INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

LABORATORIO DE CÓMPUTO INTELIGENTE

# AN UNCONVENTIONAL MODEL BASED DATA STREAMING PROCESSING

THESIS

TO OBTAIN THE DEGREE OF
DOCTOR IN COMPUTER SCIENCE

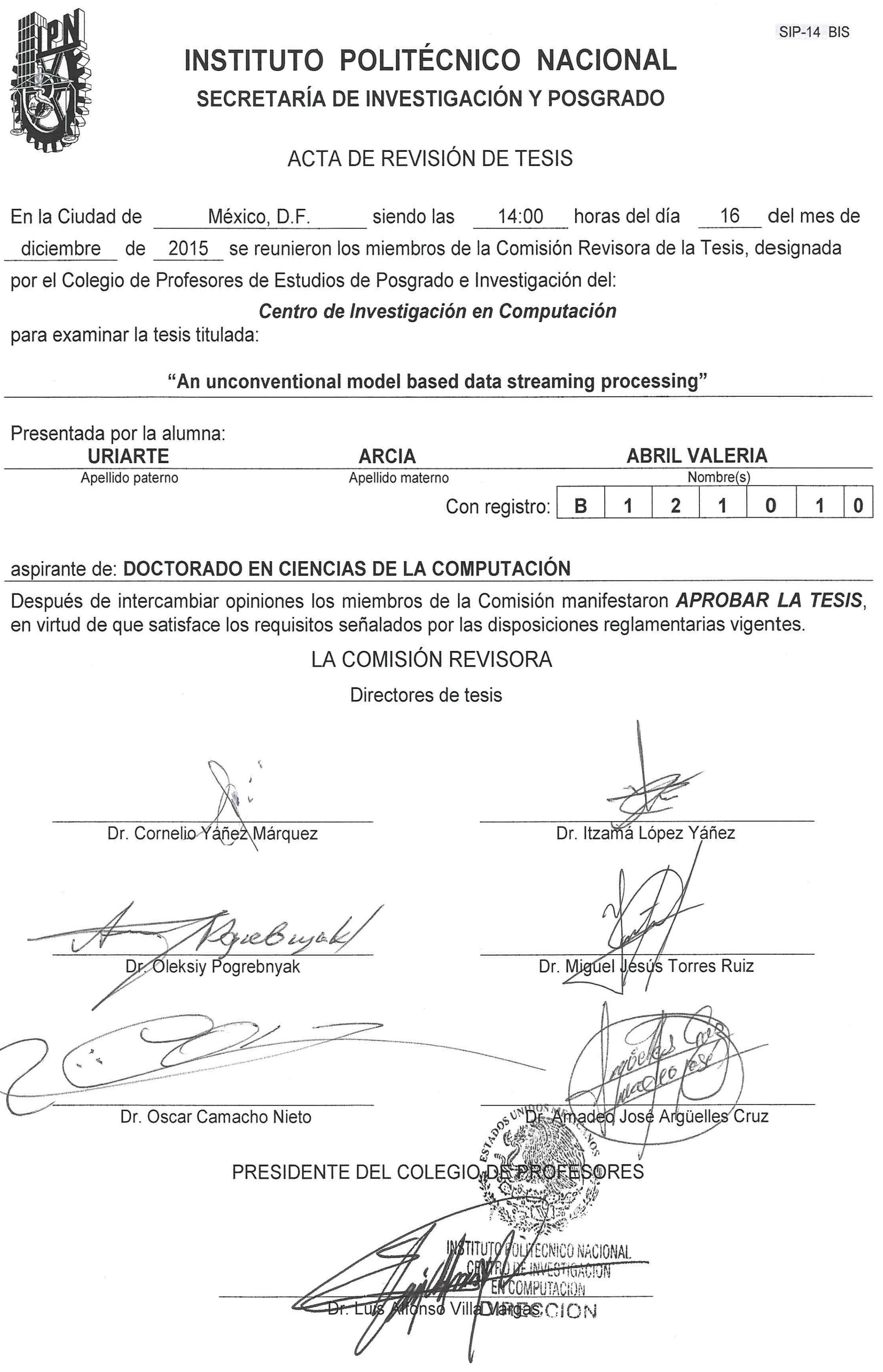
SUBMITTED BY

ABRIL VALERIA URIARTE ARCIA

THESIS ADVISORS

DR. CORNELIO YÁÑEZ MÁRQUEZ
DR. ITZAMÁ LÓPEZ YÁÑEZ

MÉXICO, D. F. JUNE, 2016



SIP-14 BIS

**INSTITUTO POLITÉCNICO NACIONAL**

**SECRETARÍA DE INVESTIGACIÓN Y POSGRADO**

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 14:00 horas del día 16 del mes de diciembre de 2015 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

***Centro de Investigación en Computación***

para examinar la tesis titulada:

**"An unconventional model based data streaming processing"**

Presentada por la alumna:

| | | |
|---|---|---|
| **URIARTE** | **ARCIA** | **ABRIL VALERIA** |
| Apellido paterno | Apellido materno | Nombre(s) |

Con registro:

| | | | | | | |
|---|---|---|---|---|---|---|
| B | 1 | 2 | 1 | 0 | 1 | 0 |

aspirante de: **DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de tesis

| | |
|---|---|
| [illegible] | [illegible] |
| Dr. Cornelio Yáñez Márquez | Dr. Itzamá López Yáñez |
| [illegible] | [illegible] |
| Dr. Oleksiy Pogrebnyak | Dr. Miguel Jesús Torres Ruiz |
| [illegible] | [illegible] |
| Dr. Oscar Camacho Nieto | Dr. Amadeo José Argüelles Cruz |

ESTADOS UNIDOS MEXICANOS

PRESIDENTE DEL COLEGIO DE PROFESORES

[illegible]

INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACION
EN COMPUTACION
DIRECCION

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 03 del mes Junio del año 2016, el (la) que suscribe Abril Valeria Uriarte Arcia alumno (a) del Programa de Doctorado en Ciencias de la Computación con número de registro B121010, adscrito a Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Cornelio Yáñez Márquez y cede los derechos del trabajo intitulado An unconventional model based data streaming processing, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección abriluriarte@yahoo.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

[illegible]

Nombre y firma

# Resumen

*Esta tesis está dedicada al diseño de una metodología novedosa para la tarea de clasificación de patrones sobre un flujo continuo de datos, basada en un clasificador asociativo. Hoy en día vivimos en una sociedad de la información, donde grandes cantidad de datos son generados por diversas áreas del conocimiento. La necesidad de extraer información útil de este flujo masivo de datos de una manera eficiente, abre una gran oportunidad para proponer nuevos métodos que permitan modelar y predecir el comportamiento de dichos datos.*

*En un* data stream*, los datos llegan de forma continua y a una gran velocidad; por tanto los algoritmos desarrollados para tratar este tipo de flujo, a diferencia de los algoritmos tradicionales, deben respetar ciertas restricciones como son: trabajar con una cantidad limitada de tiempo, utilizar poca memoria y los datos deben ser examinados una sola vez. También es importante que dichos algoritmos sean capaces de detectar cambios en la distribución que genera los datos (*concept drift*), conceptos recurrentes y surgimientos de nuevas clases.*

*Con el fin de abordar los aspectos antes mencionados, una nueva metodología basada en el clasificador Gamma se presenta en este trabajo de tesis. Este es un clasificador basado en instancias, cuya principal ventaja para trabajar con* data streams *es su fácil adaptación que puede lograrse mediante la simple adición o extracción de ejemplos del conjunto de aprendizaje sin tener que hacer costosas adaptaciones al modelo. La metodología propuesta combina el clasificador Gamma con un enfoque de ventana deslizante. Tres diferentes métodos para la actualización de la ventana deslizantes fueron utilizados en este trabajo: ventana deslizante de tamaño fijo, actualización por control estadístico y actualización por similitud del operador Gamma, siendo este último método una propuesta original de este trabajo de tesis.*

*Este método utiliza el operador $\gamma$ de similitud para calcular el grado de semejanza entre los patrones de la ventana, y con base en este grado de semejanza, se seleccionan los elementos a ser eliminados de la ventana.*

*Para un análisis más exhaustivo de la metodología, además de las pruebas realizadas con bancos de datos reales, se realizaron pruebas con bancos de datos sintéticos. Los resultados mostraron que la metodología exhibe resultados competitivos al ser comparada con otros algoritmos comúnmente utilizados para la clasificación de patrones sobre* data streams*. También se realizó un estudio comparativo de los tres métodos para actualizar la ventana deslizante, mostrando que el enfoque de actualización usando el operador Gamma de similitud mostraba mejores resultados que los otros.*

# Abstract

*This thesis is dedicated to the design of a novel methodology for the task of pattern classification over a continuous data stream based on an associative classifier. In today information society, huge amounts of data are generated from diverse fields of knowledge. The need to extract valuable information from these massive continuous data streams in an efficient manner opens an opportunity to proposed new methods to model and predict the behavior of stream data.*

*In a data stream context, data arrives continuously at high speed, therefore the algorithms developed to address this context, unlike traditional ones, must meet some constraints: work with a limited amount of time, use a limited amount of memory, and one or only few passes over the data. It is also important that such algorithms are capable of detecting the changes over time in the distribution that generated the data (concept drift), recurrent concepts and novel class detection.*

*In order to address the aforementioned issues, a methodology based on the Gamma classifier is presented in this thesis. This is an instance based classifier, whose main advantage for data streams context is that adaptation can easily be achieved by simply adding and/or removing examples to the learning set without complex model adaptation. The introduced methodology combines the Gamma classifier with a sliding window approach. Three different methods to update the sliding window were used: fixed size window, statistical controlled process, and Gamma similarity update; being the last one an original proposal introduced in this thesis. This method uses the $\gamma$ similarity operator to calculated a degree of likeness and based on this degree the elements to be forgotten from the window are selected.*

*Experiments on real and synthetic datasets were conducted to assess the efficiency of the proposed methodology. The results show that the methodology exhibits competi-*

*tive results when compared with other state of the art data stream classifiers. A comparative study of the three sliding window updating methods used is presented, showing that the similarity based approach achieved better results that the other.*

# Acknowledgment

This thesis would have never been possible without the contribution of many persons to whom I have the pleasure of expressing my gratitude.

I want to thank my parents, for their unconditional support and dedication; without them it would have been impossible to achieve the goals that until today I have reached; they have been a beacon that guides me to not lose my horizon and a safe harbor that I always want to go back; thank you for everything. To my sisters, Maya and Maite, for being my partners in crime, to understand my selfish motives and despite all always be by my side. To my grandmother Tere for being an integral part in my development as a person, I will always be grateful. To my grandmother Chepita for her eternal concern and for your prayers.

To all my friends, for your words of encouragement. To those who even in the distance have been with me and that despite how difficult it is to part, have always driven me to continue to achieve my goals. To all the people I've met in Mexico, for making me feel at home. Thanks to you I have always prevented loneliness to invade me. Especially, I would like to thank Mark and his family for always making me feel loved and welcome in your family. You all have been an important pillar for the culmination of this effort.

I am deeply indebted to my supervisors, Dr. Cornelio Yáñez Márquez and Dr. Itzamá López Yáñez, for your academic guidance and opportune advice. I would have never been able to complete this without you. To all my fellows in the Intelligent Computing Laboratory for their help, support, interest and valuable insights on my research. To my thesis committee for their valuable contributions and criticisms which have enriched this work.

I would like to thank Dr. João Gama at the University of Porto that welcomed me to visit LIAAD and for the insightful observations on my research. Your contribution was of great help to improve this work.

Finally, I would like to thank the Secretaría de Relaciones Exteriores de México, Instituto Politécnico Nacional (Secretaría Académica, COFAA, SIP and CIC), CONACyT, Universidad Nacional de Ingeniería (Nicaragua), for the support received to develop this work.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In the current document of thesis, a novel methodology for the task of pattern classification over a continuous data stream based on the Gamma classifier is presented.

## 1.1 Background

In recent years, technological advances have promoted the generation of a vast amount of information. Data is being generated from a variety of real-world applications, such as stock market transactions, credit card fraud detection, telecommunication networks, email spam filter, climate monitoring, and medical images analysis, among others. In a study performed by IDC (International Data Corporation) [1], the digital universe in 2013 was estimated in 4.4 zettabytes or 4.4 trillion gigabytes; from 2013 to 2020, the digital universe will grow by a factor of 10, meaning that the generated data will reach 44 trillion gigabytes by 2020. In 2013, only 22% of the information in the digital universe would be a candidate for analysis; less than 5% of that was actually analyzed. Another important fact presented in this study is that in 2013, the available storage capacity could hold just 33% of the information generated by the digital universe. By 2020, it is projected that the storage capacity may only store 5% 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. To address this problem data stream models have recently arose; models capable of handling flows of data that are so big that make it impossible to store them. During the last decades, machine learning algorithms have focused on batch learning where the whole dataset is available for training, but during the second part of the nineties, data stream algorithms started to become popular due to some publications that introduced the problem and proposed solutions to meet the challenges presented by this new classification scheme [2–4].

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 [5]. Thus, the major difficulties in data stream classification are related to memory space, processing and classification time, and change/evolution of the target concepts. In this context, classification algorithms have to be able to extract knowledge with only one, or few, passes over the data in a swift manner and using as little resources as possible.

A data stream model has different requirements from the traditional batch learning that is performed over static datasets where the algorithms can afford to read the data several times. When the input data is very large or even possibly infinite, not all the data can be loaded in memory and an off-line processing of the data is no longer feasible. Some of the most important constraints for data stream models are in [6, 7]:

1. *Process an example at a time:* in data stream scenarios examples arrive one after another; each one have to be inspected to decide if it will be used or ignored. Once it has been ignored there is no chance to retrieve it again. However, an algorithm can remember internally some examples for a short time frame, while respecting constraint 2.

2. *Use a limited amount of memory:* the amount of data arriving is extremely large, and potentially infinite. Thus, it will be impossible to store all the data in memory. Without a specific mechanism to limit the use of memory, it will be exhausted. Only statistics, summaries and the current model should be stored in memory. This constraint can be relaxed if external storage is used, but this need to be done with consideration of constraint 3.

3. *Work in a limited amount of time:* each example needs to be processed in real time. A data stream classification algorithm must process each element as fast as it arrives, or there will be data loss.

4. *Concept change:* the distribution generating data items can change over time. Thus, data from the past may become irrelevant, and even harmful, for the current model.

5. *Be ready to predict at any point:* the process of generating/updating the model should be as efficient as possible. The most desirable case is that the model is directly manipulated in memory by the algorithm as it processes examples, rather than having to re-compute the model based on running statistics.

The extensive research on data stream algorithms performed over the last years has produced a very large variety of techniques to address these types of problems. Currently, the main approaches use to handle data stream scenarios are the following:

- Concept drift detection methods: [8–11]
- Decision models: [12–24].
- Ensemble methods: [25–34].
- Instance based models: [5, 35, 36].
- Clustering methods: [37–41].
- Artificial neural network models: [42].
- Support Vector Machines models: [43–47].
- Frequent pattern mining methods: [48–52].
- Fuzzy models: [53–56].
- Other approaches: [57–59].

As we can see, several approaches have been used to address data stream scenarios, but to our knowledge no model based on an associative approach has been applied to deal with this kind of problems.

The associative approach is based on the associative memories, whose pioneering model was proposed by Steinbuch in 1961 [60], the *Learnmatrix*, a heteroassociative memory that works as a binary patterns classifier. In 1969, Willshaw, Buneman and Longuet-Higgins presented the *Correlograph* [61], an optical device capable of behaving as an associative memory. 1972 witnessed an intense research activity in associative memories with the work of James A. Anderson with his Interactive Memory [62], Teuvo Kohonen presented the Correlation Matrix Memories [63], Kaoru Nakano unveiled his Associatron [64] and Amari made some theoretical contributions about Self-organizing Nets of Threshold Elements [65]. In 1982, physicist John J. Hopfield published his model of associative memory [66]. This model was vital as it revived the interest in this field of research. During the following years a number of improvements were introduced to already existing models, but without any significant progress, until 1998 when Ritter *et al.* [67] presented their model of morphological associative memories.

The morphological associative memories served as inspiration for the development of the Alpha-Beta associative memories [68]. This model is based on two simple operations: the Alpha and Beta operators, which are comparable in simplicity to the basic operations of Boolean logic. Another important model that belongs to the associative approach is the Gamma classifier [69], which take some elements of the Alpha-Beta associative memories.

In a number of recent research, it has been shown theoretically and experimentally that, despite its simplicity, the pattern classification models based on the Alpha-Beta associative memories are highly effective and competitive with the models reported in current literature [70–76].

This thesis proposal is immersed within the research area of the associative a-pproach for data stream classification problems, particularly the used of the Gamma classifier applied to these kinds of problems.

## 1.2 Motivation

In today information society several applications such as: telecommunication, sensor networks, financial applications, and medical applications, generate massive amount of information. Current research is mainly focused on traditional method of classification, where datasets are static and data is available to the algorithms at any time; this approach is no longer feasible to handle the huge volume of information that is been generated. It is impossible to store all this wealth of information available, thus the design and implementation of new methods capable of extracting knowledge in an online way and able to adapt to changes in the data stream, are necessary.

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. Since the Alpha-Beta associative memories and other models based on these memo-ries, have demonstrated excellent performance when used to different applications, the objective of this thesis is to design a competitive model based on the associative approach, specifically the Gamma classifier, for data stream classification problems.

Another appealing Gamma classifier's feature for data stream scenarios is that it is an instance base learning (IBL) algorithm. These types of classifiers are good candidates for data stream scenarios since IBL algorithms are naturally incremental and adaptation can easily be achieved by simply adding and/or removing examples to the learning set without complex model adaptation.

## 1.3 General objective

Design, implement and test an adaptive incremental classification methodology capable of learning from a continuous data stream in situations where the underlying distribution of the concept may change over time. The methodology will also be able to deal with the time and space constraints inherent to data stream scenarios.

## 1.4 Specific objectives

1. Carry out a literature research on the state of the art including the main aspects for pattern classification on data stream scenarios.

2. Describe the characteristics, advantages and disadvantages of the state of the art algorithms that are used for pattern classification on data streams.
3. Perform a study of the datasets commonly used for the task of pattern classification on data streams.
4. Select a group of datasets, which can be used to test the proposed methodology. It is essential that the selected datasets allow a comparative study of the results of the proposed methodology and other representative models for data streams classification found in current literature.
5. Conduct experiments to test the new proposed methodology, and perform a comparative study of the experimental results obtained with those reported in current literature for data streams classification.

## 1.5 Main contributions

The main contribution of the thesis is a novel methodology for the task of pattern classification over a data stream based on an associative approach. The introduced methodology is based on the Gamma classifier combined with a sliding window approach. Three different methods to update the sliding window were used: fixed size window, statistical controlled process, and Gamma similarity update; being the last one an original proposal introduced in this thesis. This method uses the $\gamma$ similarity operator to calculated a degree of likeness and based on this degree the elements to be forgotten are selected.

## 1.6 Thesis organization

In this chapter the background, justification, objectives, and main contributions of the proposed methodology were described.

Chapter 2 presents a review of the current literature related to data stream models, where several research works addressing data stream mining problems were analyzed.

Chapter 3, materials and methods, includes the basic concepts and mathematical tools that are required for the development of this research work.

Chapter 4 is the main part of the thesis; in this chapter the theoretical basis of the proposed methodology is described and exemplified.

In Chapter 5, the experiments and results of the new proposed methodology are shown, as well as a study of the performance when compared with other classifiers.

Chapter 6 contains the conclusions and future work to be developed. Finally references are included.

# Chapter 2

# Related work

In this chapter, some of the main aspects for data stream mining are reviewed. The presented models comprise the state of the art in this area of knowledge.

Before presenting the different methods for data stream mining, some basic concepts relevant to this topic will be presented. Section 2.1 introduces some of the main characteristics of a data stream. Next, in section 2.2 an inherent property of data streams, concept drift, is discussed. Section 2.3 describes another important aspect of data streams, recurrent concept. Finally, in section 2.4 a study of the main approaches of data stream mining techniques is presented.

## 2.1 Data stream

A data stream can be seen as a continuous ordered sequence of instances arriving at a high speed. A flow of this type of data is huge and possible infinite, making impossible that it can be fully stored. It is also very difficult to process this kind of data using the current traditional data mining techniques due to the intrinsic nature of such data. Some of the main characteristics of a data stream are [7, 77]:

1. The data elements in the stream arrive on-line.
2. The system has no control over the order in which data elements arrive.
3. The data elements arrive at high speed forcing a real time processing.
4. Data streams are potentially unbounded in size.
5. Once an element from a data stream has been processed, it is discarded or archived. It cannot be retrieved easily unless it is explicitly stored in memory; ideally only samplings or summaries of the data should be stored in memory.
6. The distribution generating the data items can change over time. Thus, data from the past may become irrelevant.

Given these characteristics, a data stream model has to respect certain constrains. According to [77], the three main constrains to consider are: the amount of memory used to store information, the time to process each data element, and the time to answer the query of interest.

In order to deal with constrains imposed by data stream scenarios several techniques have been developed. These techniques usually use some kind of summarization to reduce the size of the input and thus also reduce the time and storage required to process such data. In [78], some of the main techniques used to reduce data size are introduced.

- **Approximation and randomization:** Approximation techniques return answers that are correct within some small fraction $\epsilon$ of error, while randomization allows a small probability $\delta$ of failure. In data stream frameworks both are used to obtain a correct result with a probability of $1 - \delta$ in an interval of radius $\epsilon$.

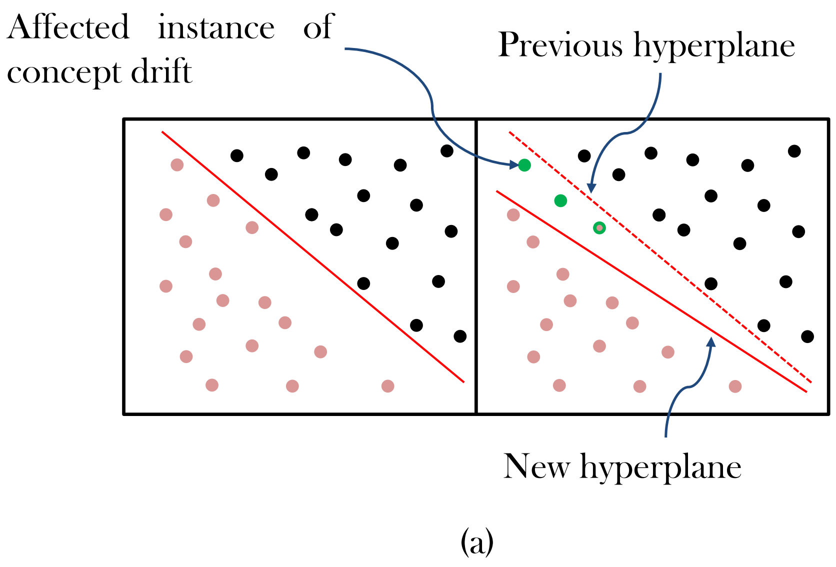
- **Time windows:** In some cases, mining over the entire past information is not necessary relevant, but only over the *recent* past. The supposition behind this assumption is that only the most recent information is relevant to the studied problem. The simplest technique is a sliding window of fixed size. A more detailed analysis of sliding windows will be presented in section 3.2.

- **Sampling:** Random sampling is possibly one of the first and simplest techniques used to reduce data size. Instead of dealing with the whole data stream, sampling selects data elements at periodic intervals. Random sampling methods can introduce errors due to the selection of data elements that are not representatives of the data stream. A better solution is the reservoir sampling presented in [79] to maintain an online random sample. The idea behind this technique is to keep a sample of size $k$, the reservoir, and replace old elements in the reservoir with new elements of the stream associated with a certain probability.

- **Synopsis, sketches and summaries:** Synopsis are compact structures that summarize data for future querying. Several synopsis methods have been used, including: wavelets [80], exponential histograms [81], and frequency moments [2], to name a few. Sketches built a statistical summary of a data stream using a small amount of memory while preserving relevant information of the data. This type of structure is generally used to estimate frequency of the most common items in a data input [2]. Cormode and Muthukrishnan [82] presented a data stream summary called count-min sketch, used for $(\epsilon, \delta)$ approximation to solve several problems in data streams such as finding quantiles and frequent items.

## 2.2 Concept drift

Data generated by most of real world applications is not static and usually changes over time. However, most of current machine learning algorithms assume that training data was generated from a stationary distribution. One of the main issues in learning from a data stream is the presence of a non-stationary probability distribution for examples generation. This change in the distribution is known in the literature as concept drift. Concept drift mean that the concept related to data that is been collected changes from time to time. It is worth noting that in this context *concept* refers to the target variable that we are trying to predict or classify.



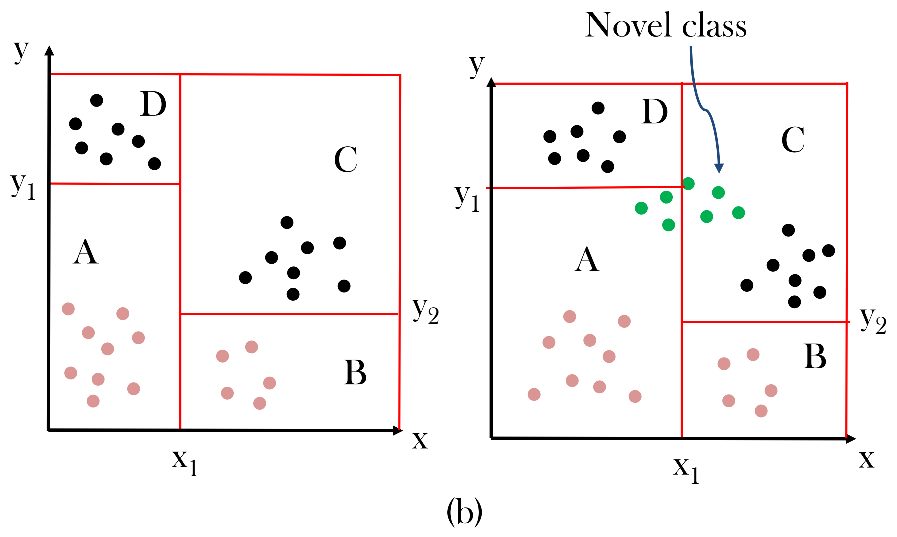


Figure 2.1: Concept Drift versus Concept Evolution

It is important to discriminate between concept drift and concept evolution. Concept drift implicates a change in the data distribution that can cause a change in the decision boundary [Fig. 2.1(a)], while concept evolution refers to the appearing of a new concept [Fig. 2.1(b)]. It is of high relevance to identify which of these problems we are facing in order to correctly address them. The first one requires a change detection method that allows data stream mining algorithms to react accordingly to the extent of the change. On the other hand concept evolution needs a novel class detection method that incorporates the new knowledge to the decision model.

Learning systems should also incorporate a method to forget outdated information and an incremental learning technique that takes into account concept drift and evolution. Old observations, that reflect the past behavior of the analyzed phenomena, become irrelevant for the current behavior and the learning system must forget that obsolete information. 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 [78]. Some algorithms use methods that adapt the decision model at regular intervals without taking into account if change really happened. This implies needless extra processing time. On the other hand, explicit change detection methods can indicate the point of change and quantify the extent of the change. In general, the methods for concept drift detection can be integrated with different base learners which provide flexibility to use this approach across a wide range of domains. According to [78], two main tactics are used in concept drift detection:

1. Monitoring the evolution of performance indicators using statistical techniques.

2. Monitoring distribution of two different time windows.

In [8], Gama *et al.* presented a method to detect change in the probability distribution of examples. The idea behind this method is to control the online error-rate of the algorithm. According to the authors statistical theory guarantees that if the distribution remains stationary the error will decrease, whereas if the distribution changes the error will increase. For the current context, they define a warning level and a drift level. A new context is declared, if the warning level is reached at example $k_d$ and the drift level at example $k_w$, then the algorithm learns a new model using only the examples since $k_d$. They used three distinct learning algorithms to test their method: perceptron, neural network and a decision tree.

He *et al.*, proposed a framework called ADAIN capable of incremental learning from data stream [9]. The objective of this approach is to incorporate the previously learned knowledge to improve the learning process from a new raw data and accumulate experience to support future decision-making process. They assumed data stream will arrived in chucks of size $D$. At time $t$, a new set of data $D_t$ arrives; the previous knowledge in this case include a previous hypothesis $h_{t-1}$, developed in time $t-1$ and a distribution function $P_{t-1}$ associated with data chunk $D_{t-1}$. The ADAIN framework proposed a mapping function that use $D_t$, $D_{t-1}$ and $P_{t-1}$ to calculate a new initial distribution for $D_t$. They also calculated a pseudo-error applying hypothesis $h_{t-1}$ to the new data chunk $D_t$ and use the error to refine the calculated distribution for $D_t$. In this research work the CART model was used as base learner and a multilayer perceptron as mapping function.

Another method for change detection is proposed by Kuncheva in [10]. This method assumes two consecutive windows of data and change detection relies on the estimation of the probability that data from the two windows come from different distributions. A study of two criteria to estimate the probability of the change in stream data, Kullback-Leibler distance [83] and Hotelling's T-square test for equal means [84], is presented. This work also introduces a new criterion for change detection, called semi-parametric log-likelihood detector, which trade some theoretical rigor for computation simplicity. All the criteria presented in this work, the two studied criteria and the proposed one, are standard statistical measures of discrepancy between two distributions.

In [11], a method for detecting concept drift which uses an exponentially weighted moving average (EWMA) chart to monitor the misclassification rate of a stream classifier is proposed. EWMA charts were originally proposed for detecting an increase in the mean of a sequence of random variables. In this work, a new estimator and time varying control limit are introduced to EWMA charts in order to adapt the method for concept drift detection. After a test instance is presented to the selected base classifier and the predicted class is determined to be correct or incorrect, the EWMA estimator is updated and if the obtained value is greater that a given threshold then a concept drift is declared. Generally, there is no way to control the false positive rate, where a false positive is defined as the detector flagging that concept drift has occurred, when in fact there is none. This method allows the rate of false positive concept drift detections to be controlled and kept constant over time.

## 2.3 Recurrent concept

The dynamic behavior of a data stream generated another phenomenon called recurring concept, which can be seen as a type of concept drift observed in most of real world problems. Detecting recurring concepts makes it possible to exploit previous knowledge obtained in the learning process [32]. It happens when a particular distribution of data reappears in the data stream. This means that old concepts (data distribution), which were learned by the classifier, can be forgotten due to adaptation to current distribution and after some time the old concepts appear again. If new incoming data belongs to an old distribution that has been forgotten by the learner, the learning process should be repeated and the first instances of the reappearing concept will probably be incorrectly classified.

Detecting recurrent concept in a data stream is a relative new area of research and also a very challenging task. Relearning a concept is costly and can significantly affect the performance of the mining process in a data stream. So, in addition to the previously mentioned characteristics desirable for a data stream model, such as any-time classification, one pass learning, and low memory use, the necessity of remembering historically learned concepts have to be added.

One of the first methods for recurrent concepts was introduced by Lazarescu in [85]. The method uses multiple windows for tracking concept drift. The algorithm attempts to interpret current data as well as detect, predict and quickly adapt to future changes in the concept. A usefulness based approach is used to control the forgetting mechanism to discard data from the system's memory. Each instance stored in the window is assigned three values: an age value, an evidence value, and a change indicator value, which are update each time a new instance arrives. The age value indicates the position in the window. The older the instance is, the higher the age of the instance. The evidence value is determined by how well the instance matches the current target concept description and the other instances stored in the window. The change indicator value depends on the evidence value. A newly observed instance may not match either the target concept or the rest of the instances in the window. In this case the instance is flagged as a potential indicator that a change is about to occur. Once the usefulness of the instance has been estimated, both the usefulness and age of the instance are checked to determine whether or not the instance is to be discarded. Additionally, the algorithm consistently checks the current rate of change and estimates (based on past history) what the future rate of change is likely to be. This information is used to improve the control over the size of the larger dynamic data window which acts as the memory of the system and it allows for a faster adaptation to changes in the concept. When drift is estimated, the repository of stored historical concepts is checked for recurrence. Concepts are described by averages of attributes, and similarity can be measured by any feature distance metric. If a match is found, then the concept is reset to the old definition.

In [86], a semi-supervised classification algorithm for data streams with recurring concept drifts and limited labeled data called REDLLA is presented. It is capable of labeling the unlabeled data with a clustering approach in the growing of a decision tree and reuses the unlabeled data combined with the labeled data for split-tests of the current tree. The deviations between clusters are used to identify new concepts (i.e., concept drifts) and recurring concepts at leaves. Firstly, with the incoming of stream data, unlabeled data are labeled at leaves using a clustering strategy and the information of unlabeled data is reused for the growing of the tree. Relevant statistics are stored at the leaves, such as the total number of instances, the distributions of class labels and attribute values of all available features. If the statistical count at a leaf is up to a pre-set value, a k-means clustering algorithm is instantiated to label unlabeled data. The merit of a split-test is evaluated, and correspondingly, the leaf can be replaced with a decision node and children leaves are generated. A majority-class method is used to label unlabeled data. Secondly, the recurring concept drift detection is instantiated using concept clusters maintained at leaves. During the first detection period, the concept clusters are directly stored in a concept list. Otherwise, they are used to compare with the last set of concept clusters. To measure the deviation between these two cluster sets, two variables were defined: the radius of a concept cluster and the distance between concept clusters. If a concept drift occurs, the last concept clusters are also compared with a set of history concept clusters to judge whether it is a recurrent concept or not. Thirdly, to avoid the space overflow or over-fitting with the continuously growing of the tree, a pruning mechanism is adopted when reaching a threshold. Several sub-trees from bottom to top with the roots whose classification error rates are more than 50% are cut off according to the simple pruning principle. Lastly, to track the performance of the current classification model, prediction results are evaluated periodically using a prequential estimation of the error based on the 0-1 loss function [87].

Gama and Kosina [88] present a general framework to detect changes in the data distribution by monitoring the learning process using drift detection techniques. When a change is detected, the learned model is stored in a sleep mode for possible reuse later. The proposed system uses a two-layer learning scheme. Each layer receives its own data and trains its own classifier. The first layer receives the data stream and trains a classifier using the labeled examples. For each incoming example, the current classifier predicts a class label. If the example is not labeled, the current model classifies the example and proceeds to the next example. If the example is labeled, it is possible to compute a loss function, and update the current decision model with the example. The prediction is either correct or incorrect, and an example is generated to train the second layer classifier. The example for the meta-classifier on the second layer has the same attribute values as for the classifier on the first layer, but the class label is either *True*, if the example was correctly classified, or *False*, if the example was misclassified. The decision problem on layer two is always binary. The meta-classifier is learning in parallel with layer one classifier. This way, the meta-classifier learns the regions of the instance space where the classifier performs well. The learning process of layer one is monitored using a change detection algorithm proposed in [8]. Whenever a drift is detected, the classifier and its meta-classifier are stored in a pool for further use. Furthermore, the system must decide whether to start learning a new decision model or activate one of the previously learned models. A classifier is reused when the percentage of votes of its meta-classifier exceeds some given threshold; otherwise a new one is learned.

## 2.4 Data stream mining

Nowadays, we face a vast amount of information generated continuously and at high speed. This kind of data is better modeled as a transitory data stream, rather than a static and conventional datasets. During the last years, several machine learning model have been proposed for data stream mining. In the following sections the main approaches used for mining this kind of data will be presented.

### 2.4.1 Decision models

Several machine learning models have been proposed based on logical/symbolic techniques. From these, two of them are more widely in use:

#### Decision trees

A decision tree uses a strategy of divide and conquer. Its strategy is to divide a difficult problem into several simpler sub-problems and recursively apply the same strategy to the sub-problems. A decision tree is a direct acyclic graph where each node can be a decision node or a leaf node. In the simplest model, each leaf node is labeled with a class and each decision node has a condition based on some attributes [77]. A decision tree is learned by recursively replacing leaves by test nodes, starting at the root. The attribute to test at a node is chosen by comparing all the available attributes and choosing the best one according to some heuristic measure. Currently, decision trees are one of the most popular methods for data stream classification.

The algorithms proposed by Domingos and Hulten [12, 13], have been the inspirations for several of the tree models used for data stream classification. In [12], the authors proposed a Very Fast Decision Tree learner (VFDT). It uses Hoeffding bounds, which guaranty that the tree can be learned in constant time (more precisely, in time that is worst-case proportional to the number of attributes), while being nearly identical to the trees a conventional batch learner would produce, given enough number of examples. One main problem to build the tree is to decide exactly how many examples are necessary at each node to choose the appropriate attribute to test. This problem was solved by using a statistical measure known as the Hoeffding bound. The main goal is to ensure that, with high probability, the chosen attribute using $n$ examples (where $n$ is as small as possible) is the same that would have been chosen using infinite examples. Concept-adapting Very Fast Decision Tree learner (CVFDT) presented in [13] is an extension of VFDT that adds the ability to detect and respond to changes in the example-generating process. CVFDT works by keeping an up-to-date decision tree from a window of examples. However, it does not need to learn a new model every time a new example arrives; instead, it updates the statistics at its nodes. This will statistically have no effect if the underlying concept is stationary. However, if the concept is changing, some splits that previously passed the Hoeffding test will no longer pass, because an alternative attribute now has higher gain. In this case CVFDT begins to grow an alternative sub-tree with the new best attribute as its root; when this alternative tree becomes more accurate that the old one, the old sub-tree is replaced by the new one.

In [14], Liang *et al.* addressed the problem of learning VFDT by using only positive and unlabeled samples with both certain and uncertain attributes; where several causes can be generating the uncertainty such as missing values, imprecise measurement, and privacy protection. Some binary classification problems can be better modeled as one-class classification scenarios, where target concept elements are called positive samples. For example, in the case of credit fraud detection, the user behaviors that cause bad economic effect could be looked as positive samples. Under this scenario a sample $s_t$ from the stream is represented by $s_t = (x, y, l)$ where $x$ is a tuple with $d$ attributes; $y \in \{y_0, y_1\}$ represents the class label of sample $s_t$, with $y_0$ for negative samples and $y_1$ for positive samples; $l \in \{l_0, l_1\}$ represents whether the class label of $s_t$ is available to the learner $(l_1)$ or not $(l_0)$. For the learner, only positive samples are labeled, while the rest are used as unlabeled samples. So if $l = l_1$, we are sure that the sample belongs to the target concept (positive samples), and if $l = l_0$, the true class label of $s_t$ is unknown. Based on the CVFDT algorithm [13], the authors proposed an improvement called puuCVFDT (positive and unlabeled uncertain CVFT). They transform the original CVFDT to cope with numerical and categorical data with both certain and uncertain attributes under positive and unlabeled learning scenario. To grow a puuCVFDT tree, an uncertain sample is partitioned into fractional samples and assigned to tree's nodes, starting from the tree root recursively. The sufficient statistics at each node are collected from these fractional samples and these statistics will be used to evaluate a heuristic measure for identifying the best split attribute. They proposed a measure called uncertain information gain for positive and unlabeled samples (puuIG) as splitting measures which is calculated using the subsets of fractional examples and the probability to observe the target concept in current data stream.

Li *et al.* proposed an algorithm called OcVFDT (One-class VFDT) in [15]. The proposal is based on VFDT [12] and POSC4.5 [89] using only positive labeled and unlabeled examples. In one class classification problems, the classifier is trained to differentiate one class of objects – target class – from the rest of objects. This algorithm only deals with discrete values while continuous attributes have to be discretized. Based on POSC4.5 a One-class Information Gain (OcIG) is used to measure information gain at each node and choose the best splitting attribute. The estimated probability of the observed positive samples from the data stream is used to calculate OcIG. As the true value of the probability is unknown, they enumerate nine possible values from 0.1 to 0.9., and construct a forest with nine different OcVFDTs. Then the best tree is chosen by estimating the classification performance of the trees with a chunk of validating samples.

An improvement to OcVFDTs was introduce by Qin *et al.* in [16]. The proposed method called PUVFDT (Positive and Unlabeled VFDT) uses an efficient method to estimate the parameter *PosLevel*, that represents the estimated probability of the observed positive samples from the data stream. It also introduces the ability to handle numeric attributes. The information gain measure for a splitting point is calculated using the *PosLevel* parameter and a random sample of the observed positive labeled and unlabeled instances. The standard algorithm to construct a decision tree usually assigns a constant at each leaf . In many cases, the constant is the majority class of the examples that fall at the leaf node. In this work, they explore the idea of using Positive Naive Bayes (PNB) classifiers at the leaf nodes of PUVFDT, which takes into account not only the prior distribution of the classes, but also the conditional probabilities of the attribute-values given the class label.

The problem of multi-label classification has hardly been studied for data stream scenarios. In multi-label classification, instead of a single class-label, each example can be associated with multiple labels. In [17], Read *et al.* proposed a framework for learning from and evaluating on multi-label data streams. In multi-label classification, we must deal with the extra dimension of $L$ binary labels per example, as opposed to the simple 0/1 classification accuracy measure. The Hoeffding tree [12], is one of the state-of-the-art classifiers for single-label data streams, and performs prediction by choosing the majority class of the examples that fall at each leaf. In this work, the authors extend the Hoeffding tree to deal with multi-labeled data: a multi-label Hoeffding tree. Their strategy is to construct a Hoeffding tree and an incremental majority-labelset classifier (the multi-label version of majority-class) is used as the default classifier on the leaves of the tree. Another important contribution of the proposed framework is its capacity to model and generate evolving synthetic multi-label data. The method is able to simulate dependencies between labels as found in real data, as well as any number and type of attributes, and their relationships to the label space. Furthermore, it can simulate how these dependencies and relationships evolve over time.

Most existing classification algorithms for data stream assume that the class labels are immediately available. However, for real world applications, this is no always true. In [18], Wu *et al.* propose a semi-supervised classification algorithm for data streams with concept drift and unlabeled data named SUN. This proposal uses a clustering algorithm developed from k-modes and implemented to produce concept clusters at leaves in an incremental decision tree. In this work, the gap in labeled data is filled by labelling relevant unlabeled data to improve the performance of the current learner. First it builds an incremental decision tree and several statistics are calculated. After a threshold is reached, a k-modes algorithm is instantiated at each available leaf to label unlabeled data and reuse the information of those new labeled data at the current node for future split-test. If the statistical count at this node is up to a pre-defined value, the merit of the split-test is evaluated using a heuristic method based on information gain and Hoeffding bounds. With every new data chunk new concept clusters are generated. Clusters generated over the last data chunk are kept as a set of historical concept clusters. The method uses the deviation between these old concepts clusters and new ones to distinguish concept drift from noise.

Rutkowski *et al.*, presented a decision tree for mining data streams based on the ID3 algorithm [19]. They emphasize that the key point of constructing a decision tree is to determine the best attribute to split the considered node. For data streams, the dominant problem is to establish the best attribute in each node, given that the stream is of infinite size. Given a dataset of $n$ elements in the considered node, we want to know if the best attribute computed from this dataset is also the best attribute for the whole data stream, with some fixed probability 1 - $\delta$. The main contribution of this work is a statistical test used to determine which attribute is best to split the node, with some probability given by the user. One relevant remark that the authors made is that their result should replace the Hoeffding bound, which has been used incorrectly in the CVFDT algorithm. They stress that the idea of the CVFDT algorithm published originally by Domingos and Hulten [13], in 2001 is correct, however the authors incorrectly used the Hoeffding bound in their paper. The same authors presented a similar work in [20], but in this case the method is based on a CART tree.

A fuzzy pattern tree is presented by Shaker *et al.* in [21] for binary classification from a data stream. A fuzzy pattern tree is a hierarchical, tree-like structure, whose inner nodes are marked with generalized (fuzzy) logical and arithmetic operators, whereas the leaf nodes are associated with fuzzy predicates on input attributes. A pattern tree propagates information from the leaf to the root node: a node takes the values of its descendants as input, combines them using the respective operator, and submits the output to its predecessor. It implements a recursive function that maps a combination of attribute values entered in the leaf nodes to a number in the unit interval, which will be delivered as the output by the root of the tree. Model adaptation is performed by anticipating possible local changes of the current model, and confirming these changes through statistical hypothesis testing. The basic idea of this evolving version of fuzzy pattern tree learning is to maintain an ensemble of pattern trees, consisting of an active model and a set of neighbor models. The active model is used to make predictions, while the neighbor models can be seen as anticipated adaptations: they are kept ready to replace the current model in case of a drop in performance, caused, for example, by a drift of the concept to be learned. More specifically, the set of neighbor models is always defined by the set of trees that are *close* to the current model, in the sense of being derivable from this model by means of a single *edit operation*, namely an expansion or a pruning step.

### Decision rules

A decision rule is a logic predicate of the form *if-then* that is used to represent and learn the relationships between entities, which characterize a class [90]. The rule induction process tries to generate a set of rules that is consistent with the training data. Rules can be learned from different methods such as: decision trees, grammars and direct induction algorithms.

One of the first method proposed for rule induction in an online environment was FLORA (FLOating Rough Approximation) [22]. To deal with concept drift and hidden contexts, this framework uses the following strategy: keep only a window of trusted examples, store concept descriptions and re-use them when a previous context re-appears, and finally use some heuristics to control these two functions. In the FLORA framework, a concept description or hypothesis is represented in the form of three description sets: the ADES (Accepted DEScriptors) containing description items matching only positive examples, the NDES (Negative DEScriptors) similarly summarizes the negative examples, and PDES (Potential DEScriptors) containing description items that are too general, matching positive examples, but also some negative ones. ADES, is used to classify new incoming examples, NDES is used to prevent over-generalization of ADES, and PDES acts as a reservoir of description items that are currently too general but might become relevant in the future. For each set a counter is kept, that specify how many positive and negative examples in the current window match the individual descriptions in ADES, NDES and PDES. The counters are updated with any addition to or deletion from a window and are used to decide when to move an item from one set to another or when to drop it from the hypotheses. In any case, items are retained only if they cover at least one (positive or negative) example in the current window. In this work they also present FLORA2, an improved version which introduces two heuristics to detect concept drift and acts accordingly by shrinking the window if concept drift occur, and keeping the window size fixed when the concept seems stable.

In 2006 Ferrer-Troyano *et al.*, presented FACIL (Fast and Adaptive Classifier by Incremental Learning) [23], 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. Within the rule learning process, a rule is said to be consistent when it does not cover any negative example (examples with different label). The core of this method lies in avoiding specific rules and allowing rules that may be inconsistent by linking them to positive and negative examples which are very near one another. The purity of a rule is the ratio between the number of positive examples that it covers and the total number of covered examples, positive and negative. When a threshold is reached, the examples associated with the rule are used to generate new positive and negative consistent rules. The classification task is performed using consistent and inconsistent rules, the former classify new test examples by covering while the latter classify them by distance similarly to the nearest neighbor algorithm. In addition, the system provides an explicit forgetting mechanism that is executed when the examples are older than an user defined threshold. Implicit forgetting is performed by removing examples that are no longer relevant as they do not enforce any concept description boundary.

In a more recent work, Tomczak and Gonczarek introduce a method for decision rule extraction called graph-based rules inducer [24], to support medical interviews in diabetes treatment. The knowledge extraction method that they proposed uses a graph representation to aggregate data and a forgetting factor to discard old examples. Each arriving example can be seen as a most specific rule that have $D$ conditions and a decision; therefore, they can be represented by a graph. This method uses one graph for positive examples and other for negative ones, separately. Nodes of the graphs are associated with inputs and outputs and arcs denote logical relations. Both graphs have the same set of vertices but they can have different sets of arcs. Arc's weights in the positive and negative graphs are associated with numbers of occurrences of all pairs of inputs and an output in the example. However, positive and negative graphs reflect only aggregated examples and they can be barely, if at all, called rule-based knowledge. Prior to describing rule induction based on both graphs, they proposed a manner of evaluating pairs of vertices and paths. The coverage and accuracy measures were used for evaluation. The coverage measure says about the generality of the rule while the accuracy measure expresses the specialization of the rule. However, coverage and accuracy are only suitable to measure the generalization and specialization of a rule separately; hence to reach a balance between generalization and specialization a synthetic criterion is also proposed. Finally, the proposed method uses an algorithm to reduce the search space. The algorithm creates a new graph, from the positive and negative graphs, in which an arc is either negative or positive. Each path that has only positive or negative weights is regarded as an admissible rule and only rules with a quality criterion greater than a given value $\theta$ are taken into account. The procedure for classification of a new example is as follows. First, all possible sub-paths of the example (note: example is treated as a path) should be found. However, only sub-paths that are either positive or negative are considered. Second, between the considered paths, the sub-path with the highest quality value is chosen.

### 2.4.2 Ensemble methods

Ensembles methods are a combination of several models whose individual predictions are combined - generally by average or voting techniques - to output a final prediction [7]. The idea behind an ensemble classifier is based on the concept that different learning algorithms explore different search spaces and evaluations of the hypothesis [77]. The use of ensemble methods in data stream contexts has increased, primarily because their modularity provides a natural way of adapting to changes.

Abdulsalam *et al.*, proposed a classifier based on random forests [25]. Random forests are an ensemble classification technique that grows a given number of decision trees. The decision trees used for the ensemble, in this work, are constructed according to the algorithm presented by Domingos and Hulten in [12]. The dynamic algorithm builds a user-defined number of trees using a selected block of the incoming data stream. The size of the selected block varies depending on the properties of the stream. The tree-building strategy passes down the incoming records to the tree, until they reach the frontier nodes. Every time a frontier node has seen a block of records of user-defined size, both the Hoeffding and Gini tests are applied. If the Hoeffding bound test is satisfied, then the frontier node is transformed into an internal node with an inequality based on the best attribute and split point given by the Gini index tests. The two children of this node become new frontier nodes. The proposed method is able to handle intermittent arrival of labeled and unlabeled instances and it is also able to judge whether the quality of the model is good enough for deployment, or whether further labeled records are required. Some drawbacks of this method are: it only handles numerical values or ordinal attributes for which the maximum and minimum values of each attribute are known; also, it assumes that the records are approximately uniformly distributed across all classes.

In [26], Brzezinski and Stefanowski analyze how the characteristics of incremental and block processing can be combined to produce new types of ensemble classifiers. They pointed out that ensemble classifiers can be categorized in two approaches:

block-based ensembles and online ensembles. The block-based ensembles are designed to work in environments were examples arrive in portions, called blocks or chunks. Such approaches are designed to cope mainly with gradual concept drifts. On the other hand, online ensembles are designed to learn in environments where labels are available after each incoming example; this gives them the possibility of reacting to concept drift much faster than block-based ensembles. The authors claim that the periodical introduction of new candidate classifiers and incremental updates of component classifiers should improve the ensemble's reactions to both sudden and gradual drifts in reasonable balance with computational costs. They propose three general strategies for adapting block-based ensembles to online environments. The first strategy converts a data block into a sliding window. Instead of evaluating component classifiers every $d$ examples, ensemble members are weighted after each example using the last $d$ training instances. The second strategy involves using an incremental classifier as an extension of a block-based ensemble. The ensemble works exactly like in the original algorithm but an additional online learner, which is trained with each incoming example, is taken into account during component voting. The last strategy uses a drift detector attached to an online learner which triggers component reweighting. The three presented strategies tackle different aspects of reacting to drifts in online environments. The experimental analysis of each strategy led to the creation of a new online algorithm, called Online Accuracy Updated Ensemble (OAUE), which tries to combine the best aspects of these strategies. OAUE maintains a weighted set of decision trees and predicts the class of incoming examples by aggregating the predictions of components using a weighted voting rule. Each component classifier is weighted based on its error and every $d$ examples a new classifier is created which will substitute the weakest classifier of the ensemble. Very Fast Decision Trees proposed by Domingos and Hulten [12] were used as base learners for the ensemble.

Ditzler and Polikar addressed the problem of detecting concept drift in a dataset with class imbalance [27]. They proposed two ensemble-based approaches for learning from imbalanced data streams. Both approaches are based on a previous work of the authors, called LEARN++.NSE [91], which train a new classifier for each batch of data. Classification is then obtained using a weighted majority voting, where weights are based on the averaged errors of the classifiers over recent environments. The first approach combines LEARN++.NSE with the Synthetic Minority class Oversampling Technique (SMOTE), an oversampling method that strategically generates synthetic data for the minority class. The second approach replaces LEARN++.NSE class independent raw classification error with a new penalty constraint to balance predictive accuracy on all classes by rewarding classifiers that perform well on both minority and majority class data. Second, it substitutes SMOTE with a bagging-based algorithm that makes strategic use of existing minority data. A sub-ensemble of classifiers is generated, each using all of the minority class data and a randomly sampled subset of the majority class. In this under sampling technique majority class data is not discarded, as each ensemble member is trained on a different bootstrap sample of the majority class data. In both approaches regression and classification trees were used as base learner in the ensemble.

Farid *et al.* in [28], proposed an adaptive ensemble classifier for mining concept drift data streams. The ensemble is composed of three decision trees; ID3 and C4.5 algorithms were used as base learners in this work and a weighted voting technique for classification of each test instance. The main contribution of this work is a method for novel class detection. A similarity based clustering algorithm is used to especially measure the class distribution of the data and monitor the arrival of exceptional novel classes. It confirms the arrival of a novel class after the three classifiers show abnormal class distributions by evaluating whether the number of classified instances in a leaf node increases or decreases in comparison to a previously calculated threshold. The algorithm assigns a weight to the current decision tree based on its classification accuracy rate for the categorization of the original training instances. The instances of the stream are labeled by the ensemble classifier and then a new classifier is trained with the most recent dataset. As soon as the new classifier becomes competitive, one of the existing classifiers in the ensemble is replaced by it, if necessary. The classifier with the smallest weight reflecting the minimum classification accuracy rate is chosen for replacement.

Detection of concept drift in data streams with class imbalance is a challenge and it is a topic of intensive research. Ghazikhani *et al.*, addressed this problem in [29]. They proposed an online ensemble of neural network classifiers. The main contribution of this work is a two-layer approach for handling class imbalance and non-stationarity. The first layer handles class imbalance with a cost-sensitive learning embedded in the training phase of the neural networks. In this type of learning, the misclassification cost is different for each class and it is adjusted to have more balanced classification results. The objective function of these neural networks was modified to include this cost-sensitive feature. Another issue that they considered was diversity. Diversity is the difference between the outputs of the individual learners in the ensemble. They used different initial weights for generating diversity. The second layer implements a new method for weighting classifiers of the ensemble. The weight is increased when a classifier has a correct prediction and decreased otherwise. The main innovation is considering two separate weights for the minority and majority classes. In other words, the performance of each classifier on each class is considered separately. The outputs of the classifiers are aggregated together using a majority vote scheme.

In [30], Masud *et al.*, proposed a data stream classification method that integrates a novel class detection mechanism into traditional classifiers. In general, data stream classification techniques would be unable to detect the novel class until the classification models are trained with labeled instances of the novel class. The algorithm starts with building the initial ensemble of $M$ classifiers with the first $M$ labeled data chunks. The algorithm maintains three buffers: $buf$ keeps potential novel class instances, $U$ buffer keeps unlabeled data points until they are labeled, and $\mathcal{L}$ buffer keeps labeled instances until they are used to train a new classifier.

At each iteration, the latest data point in the stream is classified using the selected base algorithm. The authors tested the proposed mechanism using two different classifiers: decision tree and k-nearest neighbor. If the instance cannot be classified it will be pushed into $U$. When $U$ reaches a pre-define threshold, the oldest element in $U$ will be dequeued and labeled. The labeled record will be then pushed into $\mathcal{L}$. When we have $S$ instances in $\mathcal{L}$, where $S$ is the chunk size, a new classifier $L'$ is trained using the chunk. Then, the existing ensemble is updated by choosing the best $M$ classifiers from the $M + L'$ classifiers based on their accuracies on $\mathcal{L}$, and the buffer $\mathcal{L}$ is emptied to receive the next chunk of training data. An unlabeled instance is classified by taking majority vote among the classifiers in the ensemble. The method includes a clustering algorithm which builds clusters with the training data and a summary of each cluster is stored. Test instance outside the decision boundary of the generated clusters are considered outliers and candidate for a novel class. These instances are store in $buf$ for later processing. For novel class detection a measure of cohesion is proposed. This measure is calculated using the instances stored in $buf$ and the summary of the existing clusters generated from the training data. If all of the classifiers in the ensemble discover a novel class using this measure, then arrival of a novel class is declared.

In data stream environments, it may happen that only a small fraction of the data can be labeled. A method to cope with scarcity of labeled data is proposed by Masud *et al.* [31]. They presented a technique to train and update a classification model using both, labeled and unlabeled instances. Each classifier is built as a collection of micro-clusters using semi-supervised clustering, and an ensemble of these classifiers is used to categorize unlabeled data. The input stream data is divided into equal-sized chunks so that each chunk can be stored in main memory. When P% of instances in a data chunk has been labeled, a new classifier is trained from that chunk. An initial ensemble $M$ of $L$ classifiers is built using the first $L$ data chunks. The existing ensemble is used to predict the label of the test instances using majority voting. The ensemble is updated by choosing the best models based on their individual accuracies over the labeled training data of a new data chunk. The semi-supervised clustering algorithm is based on cluster-impurity measure. It creates $K$ clusters using the partially labeled training data. The clusters are then split into pure clusters, where a pure cluster contains unlabeled data points or labeled data points of only one class. The summary of each pure cluster is then saved as a micro-cluster. They combine this set of micro-clusters with labeled micro-clusters (micro-clusters that correspond to manually labeled instances) from the last $r$ contiguous chunks. Finally, a label propagation technique to propagate labels from the labeled micro-clusters to the unlabeled micro-clusters is applied. This yields a new classification model that can be used to classify unlabeled data.

Another ensemble learning algorithm called Pool and Accuracy based Stream Classification (PASC) is presented in [32]. The algorithm maintains a pool of classifiers to deal with recurring concepts, where each classifier describes a concept of the environment; it is worth noting that in this proposal only one classifier from the pool is used to classify a test instance. Two approaches are presented to select which classifier will be used: the first one uses the active classifier where an active classifier is a classifier which was updated by the previous batch of instances. The second approach uses a weighted classifier. At the beginning of the iteration, each classifier is assigned with an initial weight. After predicting the label of an instance, the weights are updated using the real and predicted label. In later iterations, an instance is classified using the classifier which has the highest weight. For sake of efficiency the weights are not updated with every instance; instead a subsample of the batch is used. The square root of the batch size was chosen as the number of instances selected to update the weights. Naïve Bayes classifiers were used as base learners. In a second phase, the algorithm updates the pool of classifiers. When a batch of instances is received, the algorithm first predicts the corresponding labels of the instances. Then, the true label of each instance is revealed. After receiving the true labels of the whole instances of the batch, the algorithm either chooses the most appropriate classifier and updates it, or creates a new classifier that suits the received data and their labels. To avoid getting redundant, there is a constraint on the maximum number of classifiers. Three different methods to update the pool were proposed: the first one uses the Bayes theorem, the second is similar to the first except that it makes some assumptions to decrease the time complexity of the method, and the third one is a heuristic method which turns out to be more efficient than the other two. Any of these methods output the best classifier of the pool and a similarity measure. If the similarity measure is more than a predefined threshold parameter, the data batch and its labels should be given to the best classifier to be updated. Otherwise, if there is a free space in the pool or a free space can be created, a new classifier will be created with the newly arrived batch and its labels. To create a free space, it is checked to see whether a merging process can be done to merge two classifiers of the pool.

In [33], a supervised neural constructivist system (SNCS) is presented. The system design is based on Michigan-style neural learning classifier system (N-LCS) that evolves a population of Multilayer Perceptrons (MLP) on-line. Michigan-style LCSs are open frameworks that foster crossbreeding between different learning paradigms. Three key aspects are required for every implementation of Michigan-style LCS: (1) a knowledge representation, based on classifiers, which enables the system to map sensorial status with actions, (2) an apportionment of credit mechanism which evaluates the classifiers, and (3) a knowledge discovery mechanism. LCSs are on-line machine learning techniques that use genetic algorithms (GAs) and an apportionment of credit mechanism to evolve a rule-based knowledge. In order to improve flexibility and obtain accurate models beyond the limitations of rule-based systems, the use of neural networks has become popular, resulting in the N-LCS. In this work, the authors proposed a Michigan-style LCS which uses MLPs as classifiers and a GA is used to evolve the topology of the MLPs. To estimate the quality of the classifier, the authors proposed a set of parameters: fitness, experience, time and accuracy. After each learning step, the experience, accuracy and fitness of the classifiers are updated in an incremental way. To maintain a compact and accurate population, once the training phase is finished, the final population is reduced deleting those classifiers that have low experience.

Saunier and Midenet proposed a traffic management application for road safety diagnosis in signalized intersections [34]. Such application requires dealing with data streams that may be subject to concept drift over various time scales. An expert provides imprecise labels based on video recordings of the traffic scenes. In order to improve the performance – overall and for each class – and the stability of learning in a stream, this paper presents an ensemble method based on incremental algorithms that rely on their sensitivity to the processing order of instances. Most incremental algorithms build different hypotheses if the training instances are presented in different orders. These hypotheses can then be combined through a voting technique. This paper explores the use of the order in which the instances are presented to create different ensembles of classifiers. The severity, or proximity to a potential car collision, is measured by the spatio-temporal distance between the interacting vehicles, and is related to their probability of collision. The aim of the proposed method is to predict the severity indicators. Naïve Bayes classifiers are used as base learner in these ensembles. The approach relies on selecting a subset of the available instances for learning. The selection criterion of labeled instances was implemented by selecting instances that are misclassified by the current hypothesis. During the active learning process the hypotheses are iteratively used for prediction and then updated with the newly labeled instances. In batch mode, the proposed method initializes $k$ hypothesis randomly drawn from the training dataset, then the rest of the dataset is presented in a random order to each hypothesis and processed according to the selection criterion. When learning online, with a data stream $S$, the order of the data instances cannot be controlled. The proposed algorithm initializes a set of diverse hypotheses on the initialization labeled dataset of size $n$. Each instance in the stream $S$ is then considered for selection by each hypothesis: since the selection criterion depends on each hypothesis, each having a different initialization, they will pick different instances from the same data stream.

### 2.4.3 Instance based models

Instances Based Learning (IBL) algorithms use specific instances during classification and prediction tasks. The primary output of IBL algorithms is a concept description as a function that maps instances to categories. An instance-based concept description includes a set of stored instances. This set of instances can change after each training instance is processed. Concept descriptions are determined by how the IBL algorithm's similarity and classification functions use the current set of saved instances [92]. IBL algorithms are derived from the well-known nearest neighbor classifier initially proposed by Cover and Hart [93]. These algorithms use similarity functions to yield matches between instances. Model-based algorithms induce a general model from data and use this model for further reasoning; contrary to this, IBL algorithms store the data and postpone the processing until a prediction is requested (which earned the name of lazy classifiers). Predictions or classifications are derived from the information provided by the stored instances.

Algorithms for data stream contexts have to be incremental, highly adaptive and capable of handling concept change. In the case of IBL algorithms, incremental learning and model adaptation is very simple, since it comes to updating the stored instances, but high computational costs arise while classifying test instances or performing a new prediction. Consequentially, IBL might be preferable in applications where the incoming number of instance is large when compared with the number of queries that have to be performed [5].

Beringer and Hüllermeier developed an IBL algorithm for data streams classification [5]. The system maintains an implicit concept description in the form of a case base (a window with a number of instances). On arrival of a new example, this is added to the case base. It is also checked whether other examples might be removed, either since they have become redundant or since they are outliers. A set $C$ of examples within a neighborhood of the recently added example are considered candidates for removal. A statistical test is also incorporated to detect abrupt change of concept. In case a change is detected, an estimate of the change is calculated in order to determine the number of examples that need to be removed from the case base. Examples to be removed are chosen at random according to a distribution which is spatially uniform but temporally skewed. The case base is updated taking into account three indicators: temporal relevance, spatial relevance and consistency.

Another model that uses the advantage of instance based learning is presented in [35]. In this work, a model named Similarity-based Data Stream Classifier (SimC) is introduced. The system maintains an implicit concept description in the form of a case base, introducing an insertion/removal policy that adapts quickly to the data tendency and maintains a representative, small set of examples and estimators. An initial model is built using the first 100 instances, which are stored in groups, where each class will be represented by certain number of groups. Once the initial model is built, it will be used to classify new instances and will be updated. The classification function is a standard $k$-NN with a default value of $k = 1$, but this parameter can also be defined by the user. The classifier uses the heterogeneous distance measure HVDM [94], which is updated with each insertion and deletion in the case base. For the appropriate selection of instances, SimC uses estimators that provide general information about each group and each instance stored in the case base: the mean instance of each group, the age of the group, usefulness, and consistency. The instance selection policy of SimC, that updates the model, relies on the combination of these estimators

Shaker and Hüllermeier presented a instance based algorithm that can be applied to classification and regression problems in data stream scenarios [36]. The proposed algorithm is called IBLStream. The basic idea is an evolving version of fuzzy pattern tree learning (eFPT) to maintain an ensemble of pattern trees, consisting of a current (active) model and a set of neighbor models. The current model is used to make predictions, while the neighbor models can be seen as anticipated adaptations: they are kept ready to replace the current model in case of a drop in performance, caused, for example, by a drift of the concept to be learned. The set of neighbor models is always defined by the set of trees that are *close* to the current model in the sense of being derivable from this model by means of a single *edit operation*, namely an expansion or a pruning step. An expansion replaces a leaf $L$ of the current tree by a tree-node. A pruning step is essentially undoing an expansion. More precisely, each inner node except the root can be replaced by one of its sibling nodes, which means that the subtree rooted by this node is lifted by one level, while the subtree rooted by the other sibling is pruned. For each time step $t$, the error rate of the current model and, likewise, of all neighbors is calculated on a sliding window consisting of the last $n$ training examples. The length of the sliding window, $n$, is a parameter of the method; as a default a value of $n = 100$ was used, which is large from the point of view of statistical hypothesis testing and small enough to enable a fast reaction to changes of the data generating process.

### 2.4.4 Clustering methods

Clustering can be defined as a procedure to partitioning a group of objects into several sub-groups, such that the similarities between elements of a sub-group are high while similarities with elements of a different sub-group are low. Most approaches use distance between instances as a metric for similarity. This strategy is prohibitive for data stream contexts as it requires a quadratic search space regarding to the number of instances.

The data stream clustering problem is defined by [77] as "maintaining a continuously consistent good clustering of the sequence of data observed so far, using a small amount of memory and time". In the early development of data stream clustering, the focus of the research was to construct a suitable data structure for data streams based on traditional methods. Some of the methods developed during this phase were: Birch algorithm, LocalSearch algorithm, Stream algorithm and CluStream algorithm [37]. As the development of data stream clustering algorithms continued the research focuses on two aspects: the first one is the clustering for irregularly distributed data; the second is how to detect outliers and exclude the interference of noisy data.

In [38], a new k-means clustering algorithm for data streams, called StreamKM++ is proposed. The stream algorithm maintains a small sketch of the input using a merge-and-reduce technique. The data is organized in a small number of samples, each representing $2^i m$ input points (for some integer $i$ and a fixed value $m$). When two samples representing the same number of input points exist an union (merge) is executed and a new sample (reduce) is created. This small sketch is called *coreset*. A coreset for a set $P$ is a small weighted set, such that for any set of $k$ cluster centers the clustering cost of the coreset is an approximation for the clustering cost of the original set $P$ with a small relative error. The coreset construction is based on the idea of the k-means++ seeding procedure introduced in [95]. To overcome some disadvantage of the k-means++ seeding procedure, a tree structure was implemented to store the *coreset* which allows the algorithm to compute the probabilities to choose the next sample point in a faster way. The obtained results show that the running time to construct the clusters was improved when compared with other clustering algorithms, but the quality of the clusters remains the same.

Wang *et al.* proposed a data stream clustering algorithm, termed Support Vector-based Stream clustering (SVStream) in [39]. Support vectors (SVs) can obtain flexible and accurate data descriptions by mapping the data into a kernel space. Support vector domain description (SVDD) is a sphere-shaped data description. By using a nonlinear transformation, SVDD can obtain a very flexible and accurate data description relying on only a small number of SVs. The algorithm searches for the smallest sphere that encloses most of the data points in the feature space, which is described by the center $\mu$ and the radius $R$. In this scheme, data points are classified into three types: *inner points*, which lie either inside or on the sphere surface, *support vectors*, which lie on the sphere surface, and *bounded support vectors*, which lie either outside or on the sphere surface. To achieve cluster labeling, an adjacency matrix $A$ is computed using the input data and the radius $R$. Clusters are defined as the connected components of the graph induced by $A$. The authors consider that the data elements of a stream arrive over time in chunks of equal size. To adapt to changes they propose a multi-sphere representation, where multiple spheres are dynamically maintained in a sphere set. When a new data chunk arrives, if a dramatic change occurs, a new sphere is created; otherwise, only the existing spheres are updated or merged to take into account the new chunk. The data elements of this new chunk are assigned with cluster labels according to the cluster boundaries constructed by the sphere set. Another important issue lies in properly discarding bounded support vector (BSVs). Since as new data arrive, more and more BSVs are generated, causing a dramatic demand for storage and computation, but cannot be directly discarded because some of them would become new SVs in later steps. To tackle this problem, a BSV decaying mechanism is proposed, which assigns each BSV with an age, through which appropriate BSVs are discarded as outliers.

In [40], a stream clustering algorithm to detect twitter spammer accounts is introduced. Three proposals were presented in this work: a modified DenStream [41], a modified StreamKM++ [38] and a combination of these two methods. The authors proposed that instead of dealing with this problem as a classification task it will be viewed as an anomaly detection task. In this anomaly detection approach, a clustering model is built on normal twitter users with all outliers being treated as spammers. DenStream is a density based clustering algorithm which uses micro-cluster synopsis designed to summarize the clusters and a pruning strategy that allows the growth of new cluster while promptly getting rid of outliers. DenStream was modified by using p-micro-clusters to model normal instances and o-micro-clusters for outlier instances. If any incoming point is not merged with a p-micro-cluster, it is classified as abnormal and placed in the outlier buffer until deletion. It also removes the function for an o-micro-cluster to become a p-micro-cluster, this modified DenStream can only cluster instances that are considered normal. StreamKM++ is a k-means based clustering algorithm which uses the concept of a coreset which is defined as a weighted subset of the input. StreamKM++ first extracts a small set of points from the data stream and uses a merge and reduce scheme to keep the coresets manageable. The proposed modified version of StreamKM++ an additional parameter $\epsilon$ was added as the threshold for a point to be considered part of a cluster and a detection scheme was implemented in order to classify points the moment they arrive. As each new point comes in, the modified StreamKM++ finds the nearest coreset cluster based on the most recent set of centers and calculates the Euclidean distance. If the distance is above the value of $\epsilon$, then the point is detected as spam and is not inserted into the nearest coreset. Using this technique, spam instances are prevented from entering into the coresets. The third proposal combines the two modified algorithms DenStream and StreamKM++. StreamKM++ algorithm is used to classify instances with zero false negatives and then outputs all predicted classes to be used by DenStream. DenStream assumes that every StreamKM++ predicted normal instance is correct, which lowers the false positive rate and by carefully choosing a moderate value for $\epsilon$, DenStream also corrects many of the false positive mistakes made by StreamKM++ and makes the final predictions.

### 2.4.5 Neural network models

Neural networks are inspired by physiological knowledge of the organization of the brain. They are structured as a set of interconnected identical units known as neurons. The interconnections are used to send signals from one neuron to the others, in either an enhanced or inhibited way. This enhancement or inhibition is obtained by adjusting connection weights [96]. For data stream scenarios, neural network-based models have been scarcely used, due to its limitation for these contexts. In a neural network, for example, a new observation causes an update of the network weights, and this influence on the network cannot simply be canceled later on; at best, it can be reduced gradually over the course of time. [35]. Moreover, updating the weights with the arrival of each new example will require high computational cost which is not a desirable characteristic for data stream algorithms. However a few works where neural networks are used for data stream scenarios are presented in the current literature. They also have been used as base classifiers for ensemble methods [29, 33].

In [42], an online perceptron classifier for non-stationary and imbalanced data streams is proposed. This classifier uses the recursive least square (RLS) adaptive filter error model. Recursive least square is a type of adaptive filter in which the parameters are adapted by minimizing the sum of the errors in all the previous instants. The algorithm is trained with gradient descent method. The error model is suitable for classifying non-stationary and imbalanced data streams. Non-stationarity is handled with the forgetting factor of the RLS error model. An error weighting ability is embedded into the RLS error model to handle class imbalance for two-class datasets. The error weight at instant $i$, would be adjusted such that the error of the class with minor data is more important than the one for the major class. The training procedure of perceptron classifier is online, i.e., the parameters of the perceptron are updated with each incoming instance.

### 2.4.6 Support vector machines

Support Vector Machines (SVMs) are linear discriminant models generally used for pattern classification and regression. SVMs rely on pre-processing the input data to represent patterns into a higher dimensional space [90]. Using a proper nonlinear mapping function patterns can be mapped to a sufficiently higher dimension, where a hyperplane that separates the data can always be found. SVMs have been used for different tasks in data streams scenarios such as sentiment analysis, fault detection and prediction, medical applications, spam detection, and multi-label classification.

Zhang *et al.* [43], presented a learning framework for data stream context based on Support Vector Machine (SVM). First, diverse training examples are categorized into four types and learning priorities are assigned to them. Four categories are defined: labeled and from the target domain (type I), labeled and from a similar domain (type II), unlabeled and from the target domain (type III), and unlabeled and from a similar domain (type IV). Then, four learning cases are derived based on the proportion and priority of the different categories of training examples: type I dominates (case 1), type III dominates (case 2), type II dominates (case 3), and type IV dominates (case 4). Finally, for each learning case, one out of four SVM-based training models is used: classical SVM, semi-supervised SVM, transfer semi-supervised SVM, and relational k-means transfer semi-supervised SVM. To select the appropriate learning model it is necessary to estimate the number of examples of each type. An estimation method based on the concept drifting probability and the labeling rate is proposed. The authors established the learning priority of the four types of training examples as follows: $typeI > typeIII > typeII > typeIV$. When a particular type dominates the training examples, examples with lower priorities will not be used for training.

Active learning for sentiment analysis on data streams has also been addressed using SVM methods. Kranjc *et al.* [44] introduced 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. The classification algorithm first learns from a labeled set of tweets, and then classified new incoming tweets from the data stream. The incoming tweets are divided into batches. The algorithm selects most suitable tweets from a first batch for hand-labeling and puts them in a query pool. This process is repeated for every following batch and every time the query pool is updated and the tweets are reordered according to how suitable for hand-labeling they are. If the user decides to conduct hand labeling is given a selected number of top tweets from the query pool. Tweets can be labeled as positive, negative or neutral. When the manual labeling process is executed, the labeled tweets are moved to a labeled pool that later will be use to re-train the model.

Smailović *et al.* [45] presented a methodology that analyzes whether the sentiments expressed in twitter feeds, which discuss selected companies and their products, can indicate their stock price changes. The authors proposed the use of a

Pegasos SVM [46] to classify the tweets as positive, negative, and neutral. Once the tweets were classified, a sentiment indicator for predictive sentiment analysis in finance, named positive sentiment probability is calculated. This indicator is the ratio between positive tweets and the number of tweets in one day. It indicates the probability that the sentiment of a randomly selected tweet on a given day is positive. Finally, the Granger causality analysis was used to correlate positive sentiment probability and stock closing price. Granger analysis is a statistical hypothesis test for discovering whether one time series is effective for forecasting another time series. In this work, the Granger test is used to check whether there is a predictive relationship between sentiments in tweets and stock closing prices.

Krawczyk and Woźniak [47] proposed a version of incremental one-class classifier for mining stationary data streams. The proposed method uses an one-class SVM (OCSVM), that assigns weights to each object according to its level of usefulness. It computes a closed boundary in a form of a hypersphere enclosing all the objects from the target class. An object belongs to the target class, if it falls within this hypersphere; otherwise it belongs to outliers. The OCSVM assigns higher weights to instances that carry useful information, while delegating lower weights to irrelevant, recurrent or noisy examples. They introduce two schemes for estimating weights for new incoming data. The first one establishes the weight based on the distance from the center of a hypersphere to the incoming example, while the other assigns higher weights to the new incoming examples. According to their experiments the first weighting method achieved better results.

### 2.4.7 Frequent pattern mining

Frequent patterns are those items, sequences or substructures that reprise in database transactions with a user specified frequency. An itemset with frequency greater than or equal to a minimum user defined threshold will be considered as a frequent pattern. [48]. One of the first works in this area was presented by Agrawal and Srikant in 1994 [49]. They considered the problem of discovering association rules between items in a large database of sales transactions. Two algorithms were proposed to detect all sets of items (itemsets) that have minimum support, i.e,. a minimal number of transactions that use the itemsets. Later, those itemsets will be used to generate rules. Since then, frequent pattern mining task has gained popularity and become an active area of research in data stream contexts.

Nori *et al.* presented an algorithm for mining closed frequent itemsets using a sliding window approach [50]. Closed frequent itemsets have smaller sizes than a complete set of frequent patterns while they contain the same information. Therefore it is more desirable to keep closed frequent itemsets rather that maintaining a complete set of frequent itemsets. In the proposed algorithm, a prefix tree based data structure is used for better maintaining the sliding window transactions and to fast computing the support of the itemsets when the window is updated. In this model only $W$ of the most recently received transactions are considered for mining purpose, where $W$ is determined by the user. When a new transaction arrives, the oldest transaction expires and departs from the window.

In [51], an approach to discover frequent patterns from data streams in the presence of concept drift is presented. This approach is based on a process called support approximation. Given a transactional data stream and a minimum support (specified by the user), the goal of this research is to discover the itemsets whose counts are above the minimum support. The authors stated that for each itemset, the correlations between attributes in the data produce a relationship of frequency between the itemset and its subsets. They call this a *support relationship*. Therefore, given some concept of a data stream, there is a corresponding set of support-relationship instances. For support relationship modeling a multi-layer perceptron was used and a genetic algorithm was used for the determination of both the suitable structure and connection weights, where the structure is considered the number of hidden-layer nodes in the neural network model. The basic idea is that a synopsis of the data stream should be maintained from the constructed support-relationship model. With a constructed support-relationship model, given $k$ total counts of some itemsets from length 1 to length $k$, the count of an itemset that grows from these itemsets can be approximated. This process is called support approximation or count approximation. The support of an itemset of size $k + 1$ whose count is unknown can be approximated if the counts of its subsets of length $k$ and below are all known. In order to do so, a dynamically synopsis that contains itemsets of the first k orders is kept. To optimize the maintenance of the synopsis a prefix tree that is ordered lexicographically was proposed as data structure.

Lee *et al.* introduced a weighted maximal frequent pattern mining over data streams based on an sliding window model (WMFP-SW) [52]. The method performs mining operations based on tree structures. The first one, WMFP-FP-tree, is a tree constructed with weighted data and only one scan over the sliding window. As window changes, a part of the tree is removed, new data is added into the tree, and then tree restructuring operations are performed. A second tree structure, called WMFP-SW-tree, is composed of a header table including items' supports (or counts), weights, and node links and is used to manage extracted information from the window and conducting subset checking operations. A global WMFP-SW-tree is constructed with transaction data composing the current window while conditional WMFP-SW-trees are generated with partial data corresponding to each item in the header table. The overall mining procedure is as follows. (1) Transactions are read as many as the defined size of the window, and they are inserted into a global WMFP-SW-tree. (2) The tree is restructured according to support (counts) descending order of the inserted items, if necessary. (3) If there occurs a mining request, WMFP-SW mining operations are performed with the restructured global tree. (4) If new transaction data are entered to data streams, transactions included in previous steps are deleted while new ones are inserted into the tree. (5) After returning to the step 2, iterate until the WMFP mining procedure is completely finished.

### 2.4.8 Fuzzy models

Fuzzy models, also known as fuzzy-rule-based or fuzzy inference systems, are systems capable of modeling the qualitative aspect of human knowledge and reasoning processes without using a precise quantitative analysis [53]. These systems are composed of five functional blocks: a rule base composed of fuzzy *if-then* rules, a database defining membership function, a decision-making unit, a fuzzification interface and a defuzzification interface. Traditionally fuzzy inference systems employ a fixed structure closely related to an expert knowledge. During the last decade several approaches have been proposed to address the online identification of the structure for fuzzy inference systems [53–55].

In [56], Angelov proposed a data-driven method called simplified evolving Takagi-Sugeno (*simpl_eTS*). The proposed method allows complete autonomous knowledge extraction from stream data. The system automatically detects the shift in data distribution of a data stream and reacts by evolving the system structure. The structure is determined by a computationally simple evolving clustering method that analyses if the arriving example covers a new area of the data space, in which case a new cluster/rule is created while a redundant cluster/rule will be deleted. The quality of the local models and the respective automatically generated fuzzy rule base can be monitored online by qualitative measures, such as utility, which accumulates the weight of the rule contribution to the overall output.

### 2.4.9 Other approaches

In previous sections several of the most common tasks in data stream contexts were presented, but since it has become a main research topic during recent years, new challenges for machine learning algorithms have arisen. Because of this numerous approaches have been proposed to address these challenges. In this section some seldom addressed aspect of data stream mining are presented. Even when these approaches are rarely use, it is important to include them in this research work to try to achieve a better understanding of the work performed in the field of data stream mining.

A method infrequently used for data stream contexts is genetic algorithms (GA). Vivekanandan and Nedunchezhian present a scalable and adaptable online genetic algorithm to extract classification rules from data streams with concept drifts in [57]. The proposed method does not use a fixed static data set for fitness calculation. Instead, it extracts a small snapshot of the training example from the current part of the data stream whenever data is required for the fitness calculation. It also builds rules for all the classes separately in a parallel independent iterative manner. In the beginning of the process the candidate rule set for all the classes is generated randomly and the incoming data stream is divided into chunks of fixed size. After fully receiving a chunk of the data stream an iteration of the GA is performed and during this process, better candidate rules for all the classes are generated based on the received chunk of the data stream. A single window can be used to store the subset of the stream received as a whole, but the accuracy of the rules produced by the algorithm may degrade. To avoid this, the method uses separate multiple windows, one for each class, to store the stream data. Each window size depends upon the overall distribution of the corresponding class. As the stream evolves, the size of each window can also be altered based on the average distribution of the classes observed from the window in the previous $k$ chunks of the stream that have been examined so far, where the constant $k$ is defined by the user. If a window of a class becomes full while loading the stream, the oldest records in that particular window will be removed to make space for the new incoming records. All the data in all the windows put together will create a snapshot of the stream whose class distribution is approximately equivalent to the overall class distribution of the whole stream and this provides a stable data set for mining.

In [58], a hormone based nearest neighbor classification algorithm for data stream classification is presented. The proposed method uses an artificial endocrine system (AES) for updating and condensing the samples in the classifier. An AES is a model that simulates the manner in which a biological endocrine system processes information. It allows cells in a large system to communicate and interact with each other forming an entire system. The algorithm only keeps the samples on the boundaries between different classes and the amount of the boundary samples is decided *a prior*. Interior samples in a class are seen as unimportant points which should be discarded in this process. Each record is seen as a position that can accommodate only one artificial endocrine cell. $K_{total}$ endocrine cells are initialized in the initial stage. With the evolution of the data stream, the endocrine cells keep on changing their positions to track the changing boundaries between different classes. Each endocrine cell has a perceiving sensor and a releaser. Only one kind of hormone can be released by an endocrine cell and the class of the hormone released was decided by the position where the cell resided in. Each endocrine cell can perceive all kinds of hormones coming from other endocrine cells. When the same kind of hormones is perceived by an endocrine cell, the hormone concentration will be accumulated. On the contrary, when different kinds of hormones are perceived, the concentration will be reduced. During classification phase the algorithm calculates the distances between the new arrived record and all of the endocrine cells. The class label of the nearest position where the endocrine cell resided in will be set to the new record.

Most of the approaches in data stream contexts assume that data comes already pre-processed or that pre-processing is an integral part of a learning algorithm. Pre-processing of data streams has been largely overlooked in this research field. In [59], the authors presented a study of adaptive pre-processing methods and as a result of their analysis a prototype approach for combining adaptive pre-processing with adaptive predictor online was proposed. The main focus of this study is to determine under what circumstances decoupling the adaptability of the pre-processor and the predictor is beneficial to prediction accuracy. According to the authors, concept drift typically refers to changing relations between the input data and the target variable. Since there is no data evolution, only labels change; such a drift typically would not require adaptation of the pre-processing and adapting the predictor would be sufficient. Data evolution is likely to require adaptation of the pre-processor because the input data representation changes. This drift in isolation, however, may not require adaptation of the predictor, if the relation between the classes does not change. In such cases, it may be sufficient to adjust the pre-processing so that new data is repositioned to appear, where the old data previously were appearing. In cases, where both types of drift take place, we may or may not need to adapt the pre-processing, but typically we would need to adapt the predictor. Changes in data may cause that both adaptations are required, but it may be optimal to execute those adaptations asynchronously, i.e., at different times after the change. They used a parametric Naïve Bayes as predictor and identified three scenarios where decoupling adaptive pre-processing and prediction may be beneficial: the two components need different amount of data for training, changes in data do not change the relation between the classes, and changes in pre-processing need to be introduced while updates to the predictor are continuously executed.

# Chapter 3

# Materials y methods

In this chapter, materials and methods used for the development of the proposed methodology are presented. First section, introduces the main aspects of the classifier used as base learner in the data stream classification methodology proposed in this research work. Section 3.2 presents some concepts related to the sliding windows approach, which will be used as one of the methods to implement concept drift detection and a forgetting mechanism in the proposed methodology. In section 3.3 a statistical adaptable method to update a sliding window is described. This will be tested as the second method to adapt Gamma classifier for concept drift detection. An evaluation method for data stream classification algorithms is depicted in section 3.4 which will be used to assess the proposed methodology. The new methodology will be compared with other data stream classification algorithms. The MOA implementation of these algorithms will be used thus a brief description of this software is presented in section 3.5. Finally, section 3.6 presents the datasets used for experimental evaluation of the methodology.

## 3.1 The Gamma classifier

### 3.1.1 Alpha and Beta operators

In this sub-section, the Alpha ($\alpha$) y Beta ($\beta$) operators are discussed. These operators are a key part in the implementation of the Gamma classifier, which will be used as base learner in the proposed methodology. The Alpha and Beta operators were introduced in [68, 97] and are the basis of the Alpha-Beta associative memories. The operators are defined in a tabular manner, as shown in tables 3.1 and 3.2. Considering two sets, A and B, defined as:

$$A = \{0, 1\} \qquad B = \{0, 1, 2\}$$

Table 3.1: Alpha operator's definition

| $\alpha : A \times A \to B$ | | |
|---|---|---|
| $x$ | $y$ | $\alpha(x,y)$ |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 2 |
| 1 | 1 | 1 |

Table 3.2: Beta operator's definition

| $\beta : B \times A \to A$ | | |
|---|---|---|
| $x$ | $y$ | $\beta(x,y)$ |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 0 | 1 |
| 2 | 1 | 1 |

### 3.1.2 $\mathcal{U}_\beta$ operator

In this sub-section the $\mathcal{U}_\beta$ operator is presented, as originally defined in [98]. The unary $\mathcal{U}_\beta$ operator receives as input an $n$-dimensional binary vector $x$ and outputs a non-negative integer number.

**Definition 3.1.1** *Let $A$ be a set defined as $A = \{0,1\}$, $n \in \mathbb{Z}^+$ and $x \in A^n$ a binary vector of size n, with the $i$-th component represented by $x_i$. Then $\mathcal{U}_\beta(x)$ is define as follows: $\mathcal{U}_\beta(x)$ is an operator that receives a $n$-dimensional binary vector as an input argument and it outputs a non-negative integer that is calculated using the following expression:*

$$\mathcal{U}_\beta(x) = \sum_{i=1}^{n} \beta(x_i, x_i)$$

**Example 3.1.1** *Let* $x = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$*; calculate* $\mathcal{U}_\beta(x)$

Given the definition 3.1.1, $\mathcal{U}_\beta(x) = \sum_{i=1}^{6} \beta(x_i, x_i)$, thus $\mathcal{U}_\beta(x) = \beta(1,1) + \beta(1,1) + \beta(0,0) + \beta(1,1) + \beta(0,0) + \beta(1,1) = 1 + 1 + 0 + 1 + 0 + 1 = 4$. Then $\mathcal{U}_\beta(x) = 4$.

**Example 3.1.2** *Let* $x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$*; calculate* $\mathcal{U}_\beta(x)$

In this case, $\mathcal{U}_\beta(x) = \sum_{i=1}^{8} \beta(x_i, x_i)$, thus $\mathcal{U}_\beta(x) = \beta(1,1) + \beta(0,0) + \beta(0,0) + \beta(1,1) + \beta(1,1) + \beta(1,1) + \beta(0,0) + \beta(1,1) = 1 + 0 + 0 + 1 + 1 + 1 + 0 + 1 = 5$. Then $\mathcal{U}_\beta(x) = 5$.

### 3.1.3 Pruning operator

Let $x \in A^n$ and $y \in A^m$, where $n, m \in \mathbb{Z}^+$ and $n < m$, be two binary vectors of size $n$ and $m$ respectively; then $y$ pruned by $x$, denoted by $y||_x$, is a $n$-dimensional binary vector whose $i-th$ component is define as follows:

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

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

**Example 3.1.3** *For instance, let* $y = \begin{pmatrix}1\\1\\1\\1\\0\\1\end{pmatrix}$, $x = \begin{pmatrix}0\\1\\0\end{pmatrix}$, $m = 6$, *and* $n = 3$ *then* $y\|_x$ *is calculated as follows:*

$(y\|_x)_1 = y_{1+6-3} = y_4$
$(y\|_x)_2 = y_{2+6-3} = y_5$
$(y\|_x)_3 = y_{3+6-3} = y_6$

$$(y\|_x) = \begin{pmatrix}1\\0\\1\end{pmatrix}$$

Thus, only the $4th$, $5th$, and $6th$ elements of $y$ will be used in $y\|_x$. Notice that the first $m - n$ components of $y$ are discarded when building $y\|_x$. As we will see in next sections the Gamma classifier works considering similarities between binary vectors so these components will not be relevant as they are always counted as different by the classifier.

### 3.1.4 Modified Johnson-Möbius code

As presented in previous subsection, the $\mathcal{U}_\beta(x)$ operator needs a binary vector as input, thus a coding method is required for numerical vectors. A simple binary conversion can introduce additive or subtractive noise and even both as shown in table 3.3. As mentioned before the Gamma classifier works based on the similarities of two vector. If we take as an example the numbers 2 and 3 in decimal, they are not equal but similar with only one unit of difference between them; while in a binary coding the difference between 001 and 010 is of two bits. In this case the binary codification introduces additive noise that can affect the classification performance. As an alternative binary codification the modified Johnson-Möbius code was used. The modified Johnson-Möbius code, proposed in [99], which is a variation on the classical Johnson-Möbius code, allows us to convert a set of real numbers into binary representations while preserving the relationship between the numbers in decimal.

Table 3.3: Example of binary and modified Johnson-Möbius codified numbers

| Decimal number | Binary number | Modified Johnson-Möbius ) |
|---|---|---|
| 0 | 000 | 0000 |
| 1 | 001 | 0001 |
| 2 | 010 | 0011 |
| 3 | 011 | 0111 |
| 4 | 100 | 1111 |

**Algorithm 3.1.1** *Modified Johnson-Möbius coding algorithm*

1. Let $R$ be a set of real number such that

$$R = \{r_1, r_2, ..., r_i, ..., r_n\}$$

where $n$ is a positive integer and let $r_i$ be the minimum number in $R$.

2. If there is one or more negative numbers in the set, create a new set from $R$ subtracting the minimum (of the set of numbers) $r_i$ from each number in $R$, leaving only non-negative real numbers

$$T = \{t_1, t_2, ..., t_i, ..., t_n\}$$

where $t_j = r_j - r_i, \forall j \in \{1, 2, ..., n\}$ and particularly $t_i = 0$.

3. Choose a fixed number $d$ such that $d \in \mathbb{Z}^+$. Truncate each number of the set $T$ to have only $d$ decimals.

4. Scale up all the numbers of the set obtained in the previous step, by multiplying all numbers by an appropriate power of 10 ($10^d$), in order to leave only non-negative integer numbers

$$E = \{e_1, e_2, ..., e_i, ..., e_m, ..., e_n\}$$

where $e_j = t_j \cdot 10^d, \forall j \in \{1, 2, ..., n\}$ and let $e_m$ be the maximum number in $E$.

5. For each $i = 1, 2, ..., n$, concatenate $(e_m - e_i)$ zeros with $e_i$ ones, where $e_m$ is the maximum non-negative integer number in $E$, and $e_i$ is the current non-negative integer number to be coded.

**Example 3.1.4** *Let* $R = \{2.4, -0.7, 1.826, 1.5\}\,; R \in \mathbb{R}$.

**Step 1:** Let $R = \{2.4, -0.7, 1.826, 1.5\}$.

**Step 2:** There is a negative number (-0.7), thus the subtraction operation has to be performed

2.4 - (-0.7) = 3.1
-0.7 - (-0.7) = 0
1.826 - (-0.7) = 2.526
1.5 - (-0.7) = 2.2

After the subtraction a transformed set is obtained: $T = \{3.1, 0.0, 2.526, 2.2\}$.

**Step 3:** A fixed number is chosen, $d = 1$ and all the number in $T$ will be truncate to have one decimal. The following set is obtained $T = \{3.1, 0.0, 2.5, 2.2\}$.

**Step 4:** Then scale up all the numbers in $T$ using $(10)^d = (10)^1$ as scale factor. The resulting set is $E = \{31, 0, 25, 22\}$, where $e_m = 31$.

**Step 5:** For each number $e_i$ in $E$, concatenate $(e_m - e_i)$ zeros with $e_i$ ones. Each number will be codified using 31 bits.

1. $e_1 = 31$, there will be (31 - 31 = 0) zeros concatenate with 31 ones.
2. $e_2 = 0$, there will be (31 - 0 = 31) zeros concatenate with 0 ones.
3. $e_3 = 25$, there will be (31 - 25 = 6) zeros concatenate with 25 ones.
4. $e_4 = 22$, there will be (31 - 22 = 9) zeros concatenate with 22 ones.

The resulting codes for all the numbers of the original set for this example are presented in table 3.4.

Table 3.4: Example of modified Johnson-Möbius codified numbers

| Number | Codified number (using modified Johnson-Möbius coding) |
|---|---|
| 31 | 1111111111111111111111111111111 |
| 0 | 0000000000000000000000000000000 |
| 25 | 0000001111111111111111111111111 |
| 31 | 0000000001111111111111111111111 |

### 3.1.5 Generalized Gamma operator

The Gamma similarity operator is based on the Alpha and Beta operations, which belong to the Alpha-Beta associative memories. This operator indicates whether two vectors are similar or not, given a degree of dissimilarity $\theta$. This parameter specifies the tolerance, when comparing two vectors, to be considered similar, however they have differences [98]. The Gamma operator is defined as follow:

**Definition 3.1.2** *Let $A$ be a set defined as $A = \{0,1\}$, two numbers $n, m \in \mathbb{Z}^+$, two binary vector of size $n$ and $m$, respectively, $x \in A^n$ and $y \in A^m$ with the i-th component of each vector represented by $x_i$ and $y_i$, respectively, and $\theta$ a non-negative integer. The generalized similarity Gamma operator $\gamma_g(x,y,\theta)$ is defined as: $\gamma_g(x,y,\theta)$ is an operator that receives two binary vectors $x$ and $y$, and a non-negative integer $\theta$ as input arguments, and the output is a binary number, for which there are two cases:*

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

$$\gamma_g(x,y,\theta) = \begin{cases} 1 & \text{if } m - \mathcal{U}_\beta[\alpha(x,y) \text{ mod}2] \leq \theta \\ \\ 0 & \text{otherwise} \end{cases}$$

*Case 2.* If $n < m$ then the output is computed according to the following expression:

$$\gamma_g(x,y,\theta) = \begin{cases} 1 & \text{if } m - \mathcal{U}_\beta[\alpha(x,y\|_x) \text{ mod}2] \leq \theta \\ \\ 0 & \text{otherwise} \end{cases}$$

where mod2 indicates the usual modulo 2 operator.

**Example 3.1.5** *Let* $x = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$, $y = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ *and $\theta = 3$; calculate $\gamma_g(x,y,\theta)$*

**Step 1:** In this example $n = m = 6$.

**Step 2:** $\alpha(x,y)$ is calculated and the following vector it is obtained: $\alpha(x,y) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 2 \\ 1 \\ 1 \end{pmatrix}$.

**Step 3:** Apply modulo 2 to each component of the vector obtained in the previous step. The following vector is obtained:

$$\alpha(x,y) \text{ mod2} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

**Step 4:** Apply the $\mathcal{U}_\beta$ operator to the vector obtained in step 3.

$$\mathcal{U}_\beta[\alpha(x,y) \text{ mod2}] = \mathcal{U}_\beta \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = 4;$$

then perform the subtraction $m - \mathcal{U}_\beta[\alpha(x,y) \text{ mod2}] = 6 - 4 = 2$; $2 \leq \theta$; thus $\gamma_g(x,y,\theta) = 1$.

**Example 3.1.6** *Let* $x = \begin{pmatrix}1\\0\\0\\0\\1\\1\\0\\1\end{pmatrix}$, $y = \begin{pmatrix}1\\0\\1\\1\\1\\0\\1\\0\end{pmatrix}$ *and* $\theta = 1$*; calculate* $\gamma_g(x, y, \theta)$

**Step 1:** In this example $n = m = 8$.

**Step 2:** $\alpha(x, y)$ is calculate and it is obtained the following vector: $\alpha(x, y) = \begin{pmatrix}1\\1\\0\\0\\1\\2\\0\\2\end{pmatrix}$.

**Step 3:** Apply modulo 2 to each component of the vector obtained in the previous step. The following vector is obtained:

$$\alpha(x, y) \text{ mod}2 = \begin{pmatrix}1\\1\\0\\0\\1\\0\\0\\0\end{pmatrix}.$$

**Step 4:** Apply the $\mathcal{U}_\beta$ operator to the vector obtained in step 3.

$$\mathcal{U}_\beta[\alpha(x,y) \bmod 2] = \mathcal{U}_\beta \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 3;$$

then perform the subtraction $m - \mathcal{U}_\beta[\alpha(x,y) \bmod 2] = 8-3 = 5$; $5 \nleq \theta$; thus $\gamma_g(x,y,\theta) = 0$

**Example 3.1.7** *Let* $x = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$, $y = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$ *and* $\theta = 0$; *calculate* $\gamma_g(x,y,\theta)$

**Step 1:** In this example $n = 3$ and $m = 4$. Then $y$ has to be truncate using the pruning operator $(y\|_x)$ as defined in section 3.1.3. The new vector is:

$(y\|_x)$ is $\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$

**Step 2:** $\alpha[x,(y\|_x)]$ is calculate and it is obtained the following vector:

$$\alpha[x,(y\|_x)] = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

**Step 3:** Apply modulo 2 to each component of the vector obtained in the previous step. The following vector is obtained:

$$\alpha[x,(y\|_x)] \bmod 2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

**Step 4:** Apply the $\mathcal{U}_\beta$ operator to the vector obtained in step 3.

$$\mathcal{U}_\beta[\alpha[x,(y\|_x)] \bmod 2] = \mathcal{U}_\beta \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 3;$$

then perform the subtraction $m - \mathcal{U}_\beta[\alpha[x,(y\|_x)] \bmod 2] = 4 - 3 = 1$; $1 \nleq \theta$; thus $\gamma_g(x,y,\theta) = 0$. Note that although the $x$ and $y$ sections that were compared are identical, the two vectors are not. As $\theta = 0$ the result is consistent, since when $\theta = 0$, it is required that both vectors are identical to obtain a result of 1. It is worth noting that this two vectors will be considered similar by the classifier if $\theta = 1$, since this value allows one bit of dissimilarity between the two vectors. In this particular case the dissimilarity bit will be the one truncated by the pruning operator.

### 3.1.6 Gamma classifier algorithm

The Gamma classifier is a high performance pattern classifier, which was originally presented in [98]. The classifier uses the similarity Gamma operator presented in sub-section 3.1.5. It also uses the modified Johnson-Möbius coding algorithm presented in sub-section 3.1.4 to code the input vectors. In this sub-section the Gamma classifier algorithm is presented in its generalized form as described in [69]. The fundamental set the Gamma classifier uses as base case (i.e., the ideal case), is defined as follows:

**Definition 3.1.3** *Let the fundamental set of the Gamma classifier be the set of patterns associated to a class, of the form $\{(\mathbf{x}^\mu, \mathbf{y}^\mu) \,|\, \mu = 1, 2, \ldots, p\}$ where $\mathbf{x}^\mu$ is a pattern and $\mathbf{y}^\mu$ is its associated class. For such a fundamental set, the following three properties hold:*

$$\mathbf{x}^i \neq \mathbf{x}^j \ \forall i,j \in \{1,2,\ldots,p\} \textit{such that } i \neq j$$

*meaning there are no repeated patterns,*

$$\mathbf{x}^i = \mathbf{x}^j \implies \mathbf{y}^i = \mathbf{y}^j \ \forall i,j \in \{1,2,\ldots,p\}$$

*which makes sure no pattern may have associated more than one class, and*

$$\mathbf{y}^i \neq \mathbf{y}^j \implies \mathbf{x}^i \neq \mathbf{x}^j \ \forall i,j \in \{1,2,\ldots,p\}$$

*that is, different classes have different patterns associated to each other.*

*Combined, these three properties imply that the fundamental set must induce a relationship between the set of patterns to the set of classes, in a way that such relationship fulfills a function characteristics.*

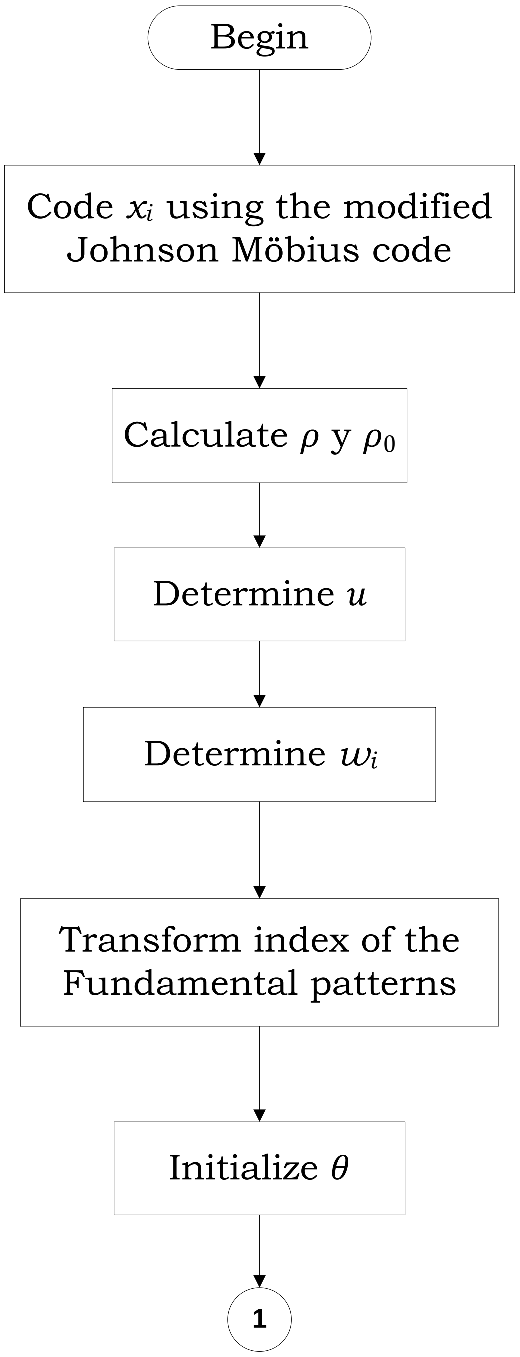

Begin
Code $x_i$ using the modified Johnson Möbius code
Calculate $\rho$ y $\rho_0$
Determine $u$
Determine $w_i$
Transform index of the Fundamental patterns
Initialize $\theta$
1

Figure 3.1: Block diagram of the Gamma classifier algorithm (first part)

**Algorithm 3.1.2** *Gamma classifier algorithm*

Let $\{(\mathbf{x}^\mu, \mathbf{y}^\mu) \,|\, \mu = 1, 2, \ldots, p\}$ be the fundamental pattern set with cardinality $p$; when a test pattern $\tilde{\mathbf{x}}$, which is an $n$-dimensional real valued vector $\tilde{\mathbf{x}} \in \mathbb{R}^n$, with $n \in \mathbb{Z}^+$, is presented to the Gamma classifier, the following step are performed:

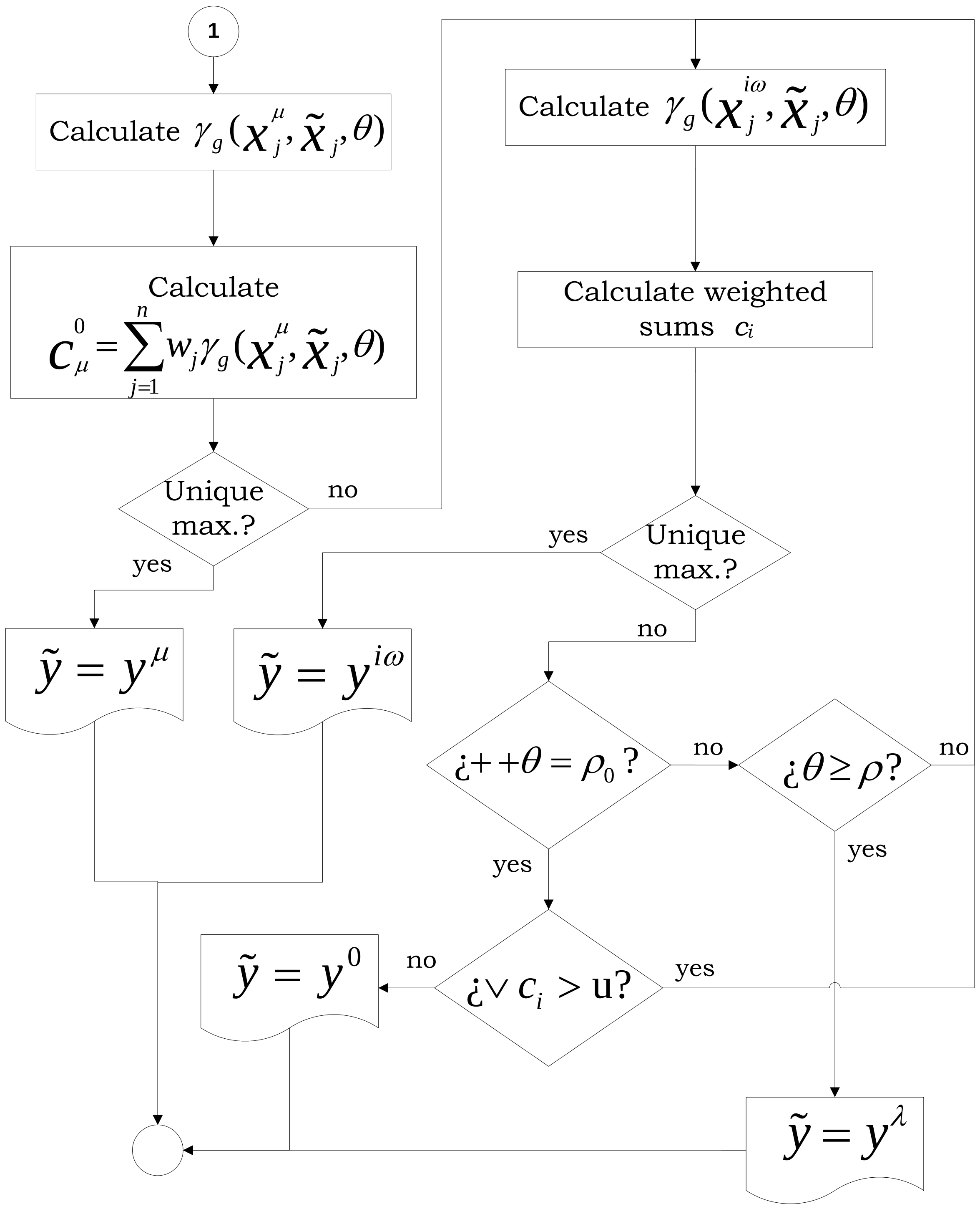

1
Calculate $\gamma_g(x_j^{\mu}, \tilde{x}_j, \theta)$
Calculate $c_{\mu}^{0} = \sum_{j=1}^{n} w_j \gamma_g(x_j^{\mu}, \tilde{x}_j, \theta)$
Unique max.?
no
yes
$\tilde{y} = y^{\mu}$
Calculate $\gamma_g(x_j^{i\omega}, \tilde{x}_j, \theta)$
Calculate weighted sums $c_i$
Unique max.?
yes
no
$\tilde{y} = y^{i\omega}$
¿$++\theta = \rho_0$?
no
¿$\theta \geq \rho$?
no
yes
yes
¿$\vee c_i > u$?
no
yes
$\tilde{y} = y^{0}$
$\tilde{y} = y^{\lambda}$

Figure 3.2: Block diagram of the Gamma classifier algorithm (second part)

1. Code the components of each pattern of the fundamental set using the modified Johnson-Möbius code, obtaining a value $e_m = \bigvee_{i=1}^{p} x_j^i$ for $j = 1, 2, , n$. With this step each component $x_j^i$ is transformed in a binary vector of dimension $e_m(j)$.

2. Convert each component of $\tilde{\mathbf{x}}$ using the modified Johnson-Möbius code, with the same parameters used to coding the fundamental set (step 1). In case that any of the components of $\tilde{\mathbf{x}}$ is greater that the corresponding $e_m$ obtained when coding the fundamental set ($\tilde{\mathbf{x}}_\xi > e_m(\xi)$), make that component equal to $e_m(\xi)$ and store the original value in the variable $mgamma_\xi$. If we obtain a negative value from any of the components once they were shifted, make that component equal to 0 and assign the value $e_m(\xi) + |\tilde{\mathbf{x}}_\xi|$ to $mgamma_\xi$.

3. Calculate the stop parameter $\rho$ and the pause parameter $\rho_0$. Depending on the problem at hand, some suggested possibilities for these parameters are:

   - $\rho = \bigwedge_{j=1}^{n} \left( \bigvee_{i=1}^{p} x_j^i \right)$.
   - $\rho = \frac{1}{n} \sum_{j=1}^{n} \left( \bigvee_{i=1}^{p} x_j^i \right)$.
   - $\rho = \bigvee_{j=1}^{n} \left( \bigvee_{i=1}^{p} x_j^i \right)$.
   - $\rho_0 = \bigwedge_{j=1}^{n} \left( \bigvee_{i=1}^{p} x_j^i \right)$, particularly if $\rho = \bigvee_{j=1}^{n} \left( \bigvee_{i=1}^{p} x_j^i \right)$.
   - $\rho_0 > \rho$, when we want to force a known class to unknown patterns.

4. Determine the pause threshold $u$. Considering that the value of this threshold depends strongly on the characteristics of the problem and the properties of the fundamental set, the following suggestions are offered as initial values:

   - $u = 0$.
   - $u = n$.

5. Determine the weights of each dimension $w_i \in \mathbb{R}^+$ for $i = 1, 2, \ldots, n$. Empirically, the following ranges as initial values are suggested:

   - Between $[1.5, 2]$ if the dimension is feature-wise linearly separable for all the classes.
   - Between $[1, 1.5]$ if the dimension is feature-wise linearly separable for some classes or feature-wise partially separable for all classes.
   - Between $[0.8, 1.2]$ if the dimension is feature-wise partially separable for some or all classes.
   - Between $(0, 0.5]$ if the dimension is feature-wise non separable.

6. Transform the index of all the patterns in the fundamental set, such that the single index that a pattern had originally (i.e., $\mathbf{x}^\mu$) will become two indexes: one for the class they belong to (i.e., $i$), and another for their position in the class (i.e., $\omega$). Under these conditions, the notation for pattern $\mathbf{x}^\mu$ will be transformed to $\mathbf{x}^{i\omega}$.

7. Initialize $\theta = 0$.

8. Compute the similarity Gamma operator $\gamma_g\left(x_j^\mu, \tilde{x}_j, \theta\right)$, according to definition 3.1.2, between each component of all the patterns in the fundamental set and the corresponding components of $\tilde{\mathbf{x}}$, considering that $mgamma_\xi$ is the dimension of the binary vector $\tilde{\mathbf{x}}_\xi$, when needed.

9. If $theta = 0$, test whether $\tilde{\mathbf{x}}$ is a fundamental pattern, by calculating the initial weighted sums $c_\mu^0$, using the results obtained in the previous step, for each fundamental pattern $\mu = 1, 2, \ldots, p$:

$$c_\mu^0 = \sum_{j=1}^{n} w_j \gamma_g\left(x_j^\mu, \tilde{x}_j, \theta\right)$$

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

$$\tilde{\mathbf{y}} = \mathbf{y}^j \text{ such that } \bigvee_{\mu=1}^{p} c_\mu^0 = c_j^0 = n$$

Otherwise, continue.

11. Compute the similarity Gamma operator $\gamma_g\left(x_j^{i\omega}, \tilde{x}_j, \theta\right)$, according to definition 3.1.2, between each component of the fundamental patterns in each class and the corresponding components of $\tilde{\mathbf{x}}$, considering that $mgamma_\xi$ is the dimension of the binary vector $\tilde{\mathbf{x}}_\xi$, when needed.

12. Compute a weighted sum $c_i$, using the results obtained in the previous step, for each class $i = 1, 2, \ldots, m$:

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

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

13. If there is more than one maximum among the different weighted sum $c_i$, increment $\theta$ by 1 and repeat steps 11 and 12 until:

    (a) There is a unique maximum among the $c_i$.
    (b) The pause parameter is reached: $\theta = \rho_0$;
    (c) The stop condition is fulfilled: $\theta \geq \rho$.

14. If the pause parameter is reached $(\theta = \rho_0)$, compare the maximum value of the weighted sums $c_i$ with the pause threshold.

(a) If $\bigvee_{i=1}^{m} c_i \leq u$ then the unknown class is assigned to the pattern:

$$C_{\tilde{\mathbf{x}}} = C_0$$

(b) If $\bigvee_{i=1}^{m} c_i > u$ then continue to step 11.

15. If there is a unique maximum, assign $\tilde{\mathbf{x}}$ to the class corresponding to such maximum:

$$\tilde{\mathbf{y}} = \mathbf{y}^j \text{ such that } \bigvee_{i=1}^{m} c_i = c_j$$

16. If the stop parameter is reached, no unique maximum was found. In such case, assign $\tilde{\mathbf{y}}$ to the class of the first maximum.

## 3.2 Sliding windows

A widespread approach for data stream mining is the use of a sliding window to handle the data. Generally, we are not interested in keeping all the information of the data stream, particularly when we have to meet the constraints of space and time required for such algorithms. This approach keeps a window with only a portion of the stream. It is useful to deal with concept drift, by eliminating those data points that came from an old concept. It can also provide a way of limiting the analyzed data stream tuples to the most relevant instances.

According to [100], the sliding window size can be fixed or variable over time:

- Sliding windows of a fixed size.
- Sliding windows of variable size.

### 3.2.1 Sliding windows of fixed size

Sliding windows of 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. This simple adaptive learning method is often used as a baseline in evaluation of new algorithms.

This method is the most straightforward tactic used for sliding windows. These types of windows are similar to first in, first out data structures. Whenever an element $i$ is observed and inserted into the window, the oldest element in the windows (i.e., $i - w$) is forgotten, where $w$ represents the window size. The parameter $w$ is assumed to be externally defined, and fixed through the execution of the algorithm. FLORA [22] is one of the earliest examples of this batch-based instance selection approach. Certain versions of the FLORA algorithm also include a mechanism that uses an adaptive window narrowing or widening according to the extent of concept drift.

Datar *et al.* [81] have considered the problem of maintaining statistics over sliding windows of fixed size. They identified a simple counting problem whose solution is a prerequisite for efficient maintenance of a variety of more complex statistical aggregates. Given a stream of bits, maintain a count of the number of 1's in the last W elements seen from the stream. They showed that, using $O(\frac{1}{\epsilon}log^2W)$ bits of memory, it is possible to estimate the number of 1's to within a factor of $1+\epsilon$. They also give a matching lower bound of $\Omega(\frac{1}{\epsilon}log^2W)$ memory bits for any deterministic or randomized algorithm. They extended their scheme to maintain the sum of the last $W$ elements of a stream of integers in a known range $[0, B]$, and provide matching upper and lower bounds for this more general problem as well.

The main challenge for the fixed size window approach, is the definition of the value for $w$. Generally, the selection for the value of $w$ is caught in a tradeoff between stability and flexibility. If the window is small, it takes care of the requirement to use low memory and the system will react quickly in phases with concept changes but in more stable phases it can affect the learner performance due to insufficient training data in the window; while a large window would produce good and stable learning results in periods of stability but cannot react quickly to concept changes. Unfortunately, the user does not know in advance the behavior of changes in the data stream so the choice for $w$ will not be optimal. And even more since the rate of change can itself changes, the optimal value of $w$ have to change too.

Another method to improve the forgetting mechanism when using a fixed size window is to consider that examples decay over time. The idea behind this approach is to choose a *decay constant* or *decay function* to weight the examples based on the assumption that the most recent data should be given a larger weight. Older examples receive smaller weights and are treated as less important by the base classifier. A decay function determines the weight of each data based on the elapsed time since the item was first observed. In [101], several decay function are presented.

### 3.2.2 Sliding windows of variable size

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.

This second approach uses windows of variable length, and generally monitors the evolution of the model's error to decide whether change has occurred. A reference algorithm is ADWIN (ADaptive sliding WINdow) presented by Bifet and Gavalda [102]. ADWIN keeps a window of variable length that reflects the change in the data. A statistical test is kept to indicate when changes occur in the data. If the test indicates a change in the distribution of the data, the window is shrunk to keep only data items that still seem to be valid. When data seems to be stationary, the window is enlarged to work on more data and reduce variance. And important characteristic of this method is that it does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique. This means that it keeps a window of length $w$ using only $O(logw)$ memory, which is important to the requirement of low memory use.

Another example of sliding window of variable size is presented in [103] which describes an online classification system that uses a fuzzy network. The system called OLIN (On Line Information Network) gets a continuous stream of non-stationary data and builds a network based on a sliding window of the latest examples. The system dynamically adapts the size of the training window and the frequency of model re-construction to the current rate of concept drift. OLIN uses the statistical significance of the difference between the training and the validation accuracy of the current model as an indicator of concept stability. OLIN adjusts dynamically the number of examples between model reconstructions by using the following heuristic: keep the current model for more examples if the concept appears to be stable and reduce drastically the size of the validation window, if a concept drift is detected. OLIN generates a new model for every new sliding window. This approach ensures accurate and relevant models over time and therefore an increase in the classification accuracy. However, the OLIN algorithm has a major drawback, which is the high cost of generating new models. OLIN does not take into account the costs involved in replacing the existing model with a new one.

## 3.3 Concept drift detection using statistical process control

This section is strongly based on the work published by Gama *et al.* in [8, 77]. The authors present a method for change detection using the probability distribution of the input examples. The method controls the online error of the algorithm. First, a warning level and a drift level are defined. Then, a new context is declared, if in a sequence of examples, the error increases reaching the warning level at example $k_w$ [Fig. 3.3(1)], and the drift level at example $k_d$ [Fig. 3.3(2)]. Finally, the algorithm learns a new model using only the examples from $k_w$ to $k_d$ [Fig. 3.3(3)].

Suppose a sequence of examples, in the form of pairs $(\overrightarrow{x_i}, y_i)$. For each example, the current decision model predicts $\widehat{y_i}$, that can be either True ($\widehat{y_i} = y_i$) or False ($\widehat{y_i} \neq y_i$). For each point $i$ in the sequence of examples, the error-rate is the probability of observing a False, $p_i$, with standard deviation given by:

$$s_i = \sqrt{\frac{p_i(1-p_i)}{i}}$$

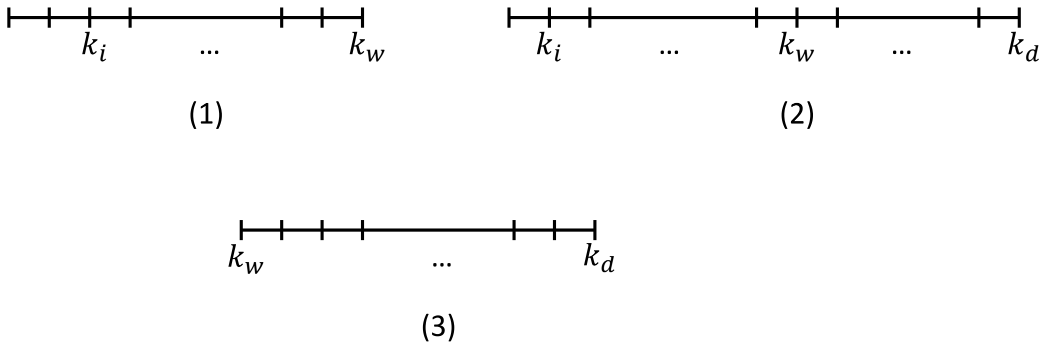


Figure 3.3: Concept drift detection using statistical process control

The drift detection method manages two registers during the training of the learning algorithm, $p_{min}$ and $s_{min}$. For each new processed example $i$, if $p_i + s_i$ is lower than $p_{min} + s_{min}$, these values are replaced by $p_i$ and $s_i$.

For a sufficiently large number of examples, the binomial distribution is closely approximated by a normal distribution with the same mean and variance. Considering that the probability distribution is unchanged when the context is static, then the $1-\alpha/2$ confidence interval for $p$ with $n > 30$ examples is approximately $p_i \pm z * s_i$. Where the parameter $z$ depends on the desired confidence level.

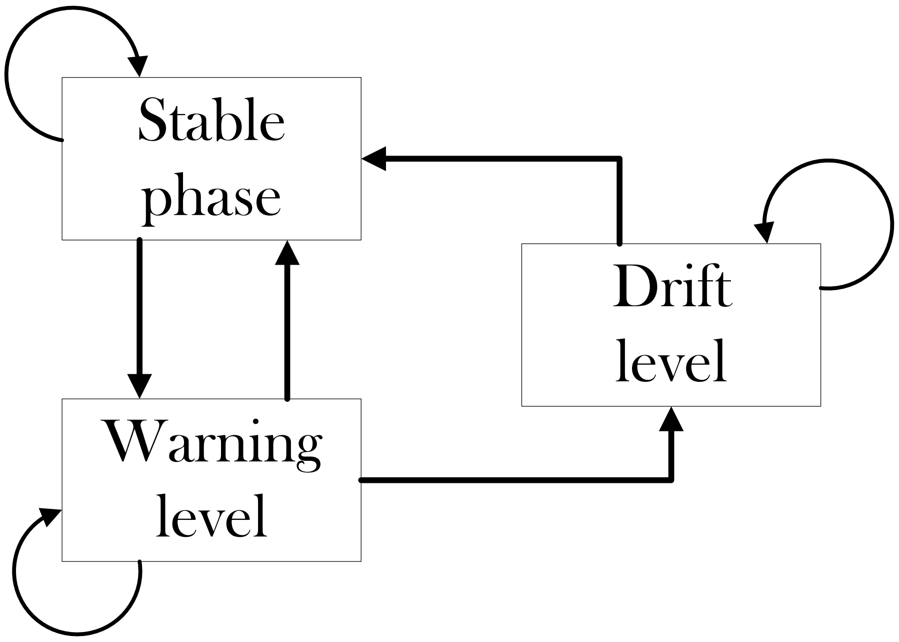


Figure 3.4: Block diagram for concept drift detection state transition

Suppose that in the sequence of examples, there is an example $j$ with correspondent $p_j$ and $s_j$. We define three possible states for the system:

- In-Control: while $p_j + s_j < p_{min} + \beta * s_{min}$. The error of the system is stable. The example $j$ is generated from the same distribution of the previous examples.
- Out-of-Control: whenever $p_j + s_j \geq p_{min} + \alpha * s_{min}$. The error is increasing, and reaches a level that is significantly higher from the past recent examples. With probability $1 - \alpha/2$ the current examples are generated from a different distribution.
- Warning: whenever the system is in between the two margins. The error is increasing but without reaching an action level. This is a not decidable state. The causes of error increase can be due to noise, drift, small inability of the decision model, among others. Thus, more examples are needed to make a decision.

Fig. 3.4 describe the state transition of the method. It is not possible to move from a stationary state to a drift state without passing the warning state. However, it is possible to move from a warning state to a stationary state. For example, we can observe an increase of the error reaching the warning level, followed by a decrease. We can assume that such situation corresponds to a false alarm, most probably due to noisy data, without changing of context.

This drift detection method provides information not only when drift occurs but also the rate of change. The distance between warning level and drift level provides such information. Small distances imply fast change rate while larger distances indicate slower changes. Considering only the warning zone, the ratio between the number of errors and the number of processed examples is an indication of the rate of change. A major characteristic is the use of the variance of the estimated error to define the action boundaries. The boundaries are not fixed but decrease as confidence in the estimated error increase. The method can be directly implemented inside on-line and incremental algorithms, or as a wrapper to batch learners.

## 3.4 Evaluating stream algorithms

An important issue for all intelligent learning algorithms is the design of a framework for evaluation and comparison with other models. In data stream contexts this is still an open topic of research and it has not been yet fully addressed. The contents of the rest of this section is mainly based on [87]. In this work, Gama *et al.* presented a very complete and well fundamented study on assessing data stream learning algorithms. The authors proposed the use of prequential error with forgetting mechanism to provide reliable errors estimators. They demonstrated that, in stationary data and for consistent learning algorithms, the holdout estimator, the prequential error and the prequential error estimated over a sliding window or using fading factors, all converge to the Bayes error [1].

[1] It is important to mention that the software MOA [107] implements most of the methods discussed in this work.

Again, let us assume a sequence of examples in the form of pairs $(x_i, y_i)$. For each example, the current decision model predicts $\hat{y}_i$, that can be either *True* $(\hat{y}_i = y_i)$ or *False* $(\hat{y}_i \neq y_i)$. For each point $i$ in the sequence, the error-rate is the probability of observing *False*, $p_i$. The authors stated that if the distribution of the examples is stationary and the examples are independent, the error rate of a consistent learning algorithm $(p_i)$ will decrease when the number of training examples, $i$, increases.

To evaluate a learning model in data stream contexts, two feasible alternatives are presented in the literature: the predictive sequential (or prequential) and the holdout. In [87], several evaluation metrics are proposed based on an accumulated sum of a $0-1$ loss function $L$ between the predicted values $(\hat{y}_i)$ and the observed values $(y_i)$.

*Holdout evaluation*, the current decision model is applied to a test set at regular time intervals (or set of examples). For a large enough holdout, the loss estimated in the holdout is an unbiased estimator. In a holdout test set with $M$ examples, the $0-1$ loss is computed as:

$$H_e(i) = \frac{1}{M}\sum_{k=1}^{M} L(y_k, \hat{y}_k) = \frac{1}{M}\sum_{k=1}^{M} e_k$$

*Predictive sequential (or prequential)*, the error of a model is computed from the sequence of examples. For each example in the stream, the actual model makes a prediction based only on the example attribute-values. We should point out that, in the prequential framework, we do not need to know the true value $y_i$ for all points in the stream. The framework can be used in situations of limited feedback by computing the loss function and $S_i$ for points where $y_i$ is known. The prequential error, computed at time $i$, is based on an accumulated sum of a loss function between the prediction and observed values:

$$P_e(i) = \frac{1}{i}\sum_{k=1}^{i} L(y_k, \hat{y}_k) = \frac{1}{i}\sum_{k=1}^{i} e_k$$

*Error estimators using sliding windows*, sliding windows are one of the most used forgetting strategies. They are used to compute statistics over the most recent past. The prequential error is computed, at time $i$, over a sliding window of size $w$ $(e_k | k \in [i-w, i])$ as:

$$P_w(i) = \frac{1}{w}\sum_{k=i-w+1}^{i} L(y_k, \hat{y}_k) = \frac{1}{w}\sum_{k=i-w+1}^{i} e_k$$

*Error estimators using fading factors*, another approach to discount older information across time consists of using fading factors. The prequential error computed at time $i$, with fading factor $\alpha$, can be written as:

$$P_\alpha(i) = \frac{1}{i}\sum_{k=1}^{i}\alpha^{i-k}L(y_k,\hat{y_k}) = \frac{1}{i}\sum_{k=1}^{i}\alpha^{i-k}e_k$$

## 3.5 MOA

Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from evolving data streams [6]. 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 will be used in this work to perform the comparative analysis between the proposed methodology and several algorithms commonly used in data stream scenarios. Several reasons were taken into account to select this platform:

- MOA has synthetic data streams generators that allow simulating potentially infinite sequences of data with different characteristics (different types of concept drifts and recurrent concepts, among others).
- It implements several popular algorithms used for data stream classification such as Hoeffding trees, AWE and ensembles methods.
- Assessing machine learning algorithms is one of the most relevant and difficult problems in data mining. MOA implements most of the evaluation metrics discusses in previous section. This will provide a general methodology to evaluate the learning algorithms in a data stream scenario.

### 3.5.1 Classifiers

This sub-section introduces some of the algorithms for data stream classification that are implemented in MOA. These algorithms will be used during the experimental phase of this work to perform a comparative analysis with the proposed methodology. The algorithms selection was made with the aim of covering the spectrum of the different approaches commonly used in data streams classification.

#### Naïve Bayes

It implements a Naïve Bayes incremental learner. It is a classifier algorithm known for its simplicity and low computational cost. Given $n_C$ different classes, the trained Naïve Bayes classifier predicts for every unlabeled instance $I$ the class $C$ to which it belongs by computing the probability of $I$ being in class $C$. This algorithm is naturally incremental: upon receiving a new example (or a batch of new examples), it simply increments the relevant counts. Predictions can be made at any time from the current counts.

### Perceptron-DDM

This implements a single perceptron classifier that performs a multiclass learning incrementally. It incorporate the drift detection method proposed by Gama *et al.* [8] to detect concept drift.

### Active classifier

Active learning focuses on learning an accurate model with as few labels as possible. Stream data poses additional challenges for active learning, since the data distribution may change over time and classifiers need to adapt. Conventional active learning strategies concentrate on querying the most uncertain instances, which are typically concentrated around the decision boundary. If changes do not occur close to the boundary, they will be missed and classifiers will fail to adapt. MOA implements four active learning strategies for stream data that explicitly handle concept drift. They are based on randomization, fixed uncertainty, dynamic allocation of labeling efforts over time, and randomization of the search space. It also contains the selective sampling strategy, which uses a variable labeling threshold.

### Hoeffding tree

A Hoeffding tree is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution generating examples does not change over time. Hoeffding trees exploit 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 observations needed to estimate some statistics within a prescribed precision (in this case, the goodness of an attribute to become a splitting attribute). Using the Hoeffding bound one can show that its output is asymptotically nearly identical to that of a non-incremental learner using infinitely many examples.

### IBL-DS

IBL-DS is an incremental learner based on instances. The system maintains an implicit concept description in the form of a case base. On arrival of a new example, this is added to the case base. It is also checked whether other examples might be removed, either since they have become redundant or since they are outliers. The case base is updated taking into account three indicators: temporal relevance, spatial relevance and consistency. It is implemented under MOA as presented in [5].

### AWE

AWE is a weighted ensemble of classifiers proposed by Wang *et al.* [104]. This ensemble method uses a weighted approach that assigns each classifier weight, such that the weight is reversely proportional to the classifier's expected error. The mean squared error was used for estimation of the classification error. The algorithm will keep an ensemble of $K$ classifiers, where $K$ is a user defined parameter. When a new chunk of data arrives, a new classifier is trained and the $K$ top weighted classifiers are kept. In MOA several classifiers can be used as base learners such as: Hoeffding trees, Naïve Bayes, Perceptrons, among others.

#### OzaBoostAdwin

OzaBoost is an incremental on-line boosting proposed by Oza and Russell [105]. This boosting algorithm uses the Poisson distribution for deciding the random probability that an example is used for training. Each base model of the ensemble is generated using a weighted training set. Each time a training example is misclassified by a base model, the Poisson distribution parameter associated with that example is increased when presented to the next base model; otherwise it is decreased. This implementation of MOA incorporates Adwin [102] to OzaBoot for adaptive windows handling.

#### SGD

SGD is a stochastic approximation of the gradient descent optimization method for minimizing an objective function. The true gradient of the function is approximated by the gradient of a single randomly picked example. MOA implements stochastic gradient descent for learning various linear models: binary class SVM, binary class logistic regression and linear regression.

#### Spegasos

Implements the stochastic variant of the Pegasos (Primal Estimated sub-GrAdient SOlver for SVM) method of Shalev-Shwartz *et al.* [46]. This method proposed a simple stochastic sub-gradient descent algorithm to solve an empirical loss minimization problem with a penalty term for the norm of the classifier that is being learned. At each iteration, a single training example is chosen at random and used to estimate a sub-gradient of the objective, and a step with pre-determined step-size is taken in the opposite direction. The algorithm is used to solve the optimization problem cast by a SVM linear kernel.

Consult [106], for further details on the implementation of theses algorithms.

## 3.6 Datasets

This section provides a brief description of the data streams used during the experimental phase of this work. The proposed solution has been tested using synthetic and real data streams. Due to the large number of examples required to evaluate data stream classification problems, just a few suitable real data streams are available. A few data streams with enough examples are hosted in [107, 108]. For further evaluation, the use of synthetic data streams generators becomes necessary. The synthetics data streams used during the experiments were generated using the

MOA framework mentioned in previous section. A summary of the main characteristics of the data streams is shown in Table 3.5. Descriptions of the data streams were taken from [107, 108] in the case of real ones and from [106] for the synthetics. Several datasets commonly present in current literature for data stream scenarios were evaluated. Only datasets with numerical attributes were selected as the classifier that will be used works by default with this type of data. No pre-processing procedures were applied to the selected datasets.

Table 3.5: Datasets characteristics

| Dataset | Instances | Attributes | Classes |
|---|---|---|---|
| Agrawal | 100,000 | 10 | 2 |
| Bank | 45,211 | 16 | 2 |
| CoverType | 581,012 | 54 | 7 |
| Electricity | 45,312 | 8 | 2 |
| Hyperplane | 100,000 | 10 | 2 |
| RandomRBF | 100,000 | 10 | 2 |

### 3.6.1 Real data streams

In this sub-section we describe the real data streams that will be used during the experimental phase of this work. The used datasets are public and available for download from [107] and [108].

*Bank marketing dataset.* The bank marketing dataset is available from the UCI Machine Learning Repository [108]. 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 to 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 [109]. 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 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. The normalized version of this dataset was used, so that the numerical values are between 0 and 1. The dataset is available from [107].

*Forest covertype dataset.* The forest covertype dataset is one of the largest datasets available from the UCI Machine Learning Repository [108]. This dataset contains information that describes forest cover types from cartographic variables. A given observation ($30 \times 30$ 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.

### 3.6.2 Synthetic data streams

In this sub-section we describe the synthetic data streams that will be used during the experimental phase of this work. All the synthetic data streams used in this work were generated using the MOA framework. This framework can be downloaded from [107].

*Agrawal data stream.* This generator was originally introduced by Agrawal *et al.* in [110]. 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. Perturbations shift 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 [106]. 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 and it is associated with a standard deviation, a class label, and a 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 [106]. For the experiments, a data stream with 100,000 data instances, 10 numerical attributes, and 2 classes was generated.

*Rotating hyperplane data stream.* A hyperplane in $d$-dimensional space is the set of points $x$ that satisfy the following expression [106]:

$$\sum_{i=1}^{d} x_i w_i = w_0 = \sum_{i=1}^{d} w_i$$

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. MOA introduces change to this data stream by adding drift to each weight attribute using this formula: $w_i = w_i + d\delta$, where $\delta$ is the probability that the direction of change is reversed and $d$ is the change applied to every example. For the experiments, a data stream with 100,000 data instances, 10 numerical attributes, and 2 classes was generated.

# Chapter 4

# Proposed solution

This chapter describes the proposed methodology for pattern classification task over a data stream based on an associative model. The proposed methodology uses the Gamma classifier as base learner combined with a sliding window approach to handle concept drift and the space and time constraints inherent to this type of scenarios. Three different methods to update the sliding windows were tested and a comparative analysis was performed.

## 4.1 Setting the problem

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

- $s_i = (x^\tau, y^\tau)$, where $x^\tau$ is an input vector of size $n$ and $y^\tau$ is the class label associate to $x^\tau$.
- $s_i = (x^\tau)$, where $x^\tau$ is an input vector of size $n$ and it is a unlabeled data element.

### 4.1.1 Challenges

*Concept drift.* In data streams the information is not static and usually changes over time. This problem is called concept drift in current literature. This phenomenon can greatly influence the performance of a learning model; mainly because the model learned from past data that may have become irrelevant and consequently the performance of the model will decay.

*Recurrent concept.* Because of the forgetting mechanism used during the data stream classification process a concept can be forgotten. If the concept reappears after some time, the learner will probably misclassify new arriving instances that belong to this old concept. A mechanism should be implemented to handle this issue.

One approach is to keep prototypes of the classes in a secondary memory using the discarded instances of the forgetting mechanism.

*Novel class.* Given the changing nature of data streams, it is possible that at some point a new class emerge. The learning algorithms need to incorporate a mechanism to include the new knowledge that is being generated by the data stream.

### 4.1.2 Goal

The main goal of this work is to design a methodology that minimizes the classification error while handling the change in the distribution of the input data, the occurrence of recurrent concepts and the emergence of new classes. The methodology will use a sliding window approach to incorporate new knowledge from the incoming data and also forget old information when data is outdated. This will allow it to handle concept drift. It also incorporates a recurrent memory where a prototype of the so-far seen classes will be stored. This memory will allow the model to keep a history of the concepts and use it in case old concepts that have been forgotten re-appear.

## 4.2 Proposed solution

The proposed solution uses the Gamma classifier as base learned and combines it with a sliding window approach, referred to as DS-Gamma. The biggest advantage that this instance base classifier provides is that adaptation to change in the behavior of the incoming data can be achieved with updating the elements in the sliding windows. Giving this characteristic, three different methods are proposed for updating such window. Each one of them implements a different combination of learning/forgetting policies. The learning (insertion)/ forgetting (removal) policies were designed to guarantee the relevance of the data, and the consistency of the current model.

In order to add flexibility to the methodology a modular approach has been adopted. Different sub-systems are used to perform each of the functions to accomplish the task of data stream classification (i.e., learning phase, forgetting mechanism, re-learning policy). Due to this features, modification of the methodology can be done in a very easy way. This gives the methodology a great flexibility for its application to a wide variety of classification problems.

To start the process of classification the methodology will wait to have $w$ labeled patterns. This parameter is the maximum size of the windows and it is user defined. A small data summary is created using these first labeled patterns. During the process of classification this data summary will be updated. The data summary contains the classes that have been seen so far, some counter and a prototype for each class. If a class is forgotten due to the forgetting mechanism, the prototype of the class will remain in the data summary, to be used in case that the class reappears (recurrent concept). This data summary is also a flexible structure that allows incorporating a new emerging class to the summary when needed (novel class detection). In this way the methodology will incorporate the new knowledge generated by the data stream to the classification process. The prototype of the class is calculated using the recursive version of the sample mean using the following expression [77]:

$$\overline{x}_i = \frac{(i-1) \times \overline{x}_{i1} + x_i}{i}$$

The methodology will have two modalities of work depending on whether the input pattern is assigned with a class label or not:

*Arriving of unlabeled patterns.* When a pattern $x^i$ arrives without an associated class label, the DS-Gamma classifier will try to classify it. If classification is successful the model will output a class label $\tilde{y}^i$ for the input pattern $x^i$. If classification was not possible the input pattern $x^i$ will be stored in a buffer as a candidate for a novel class. Fig. 4.1 depicts the process for unlabeled patterns.

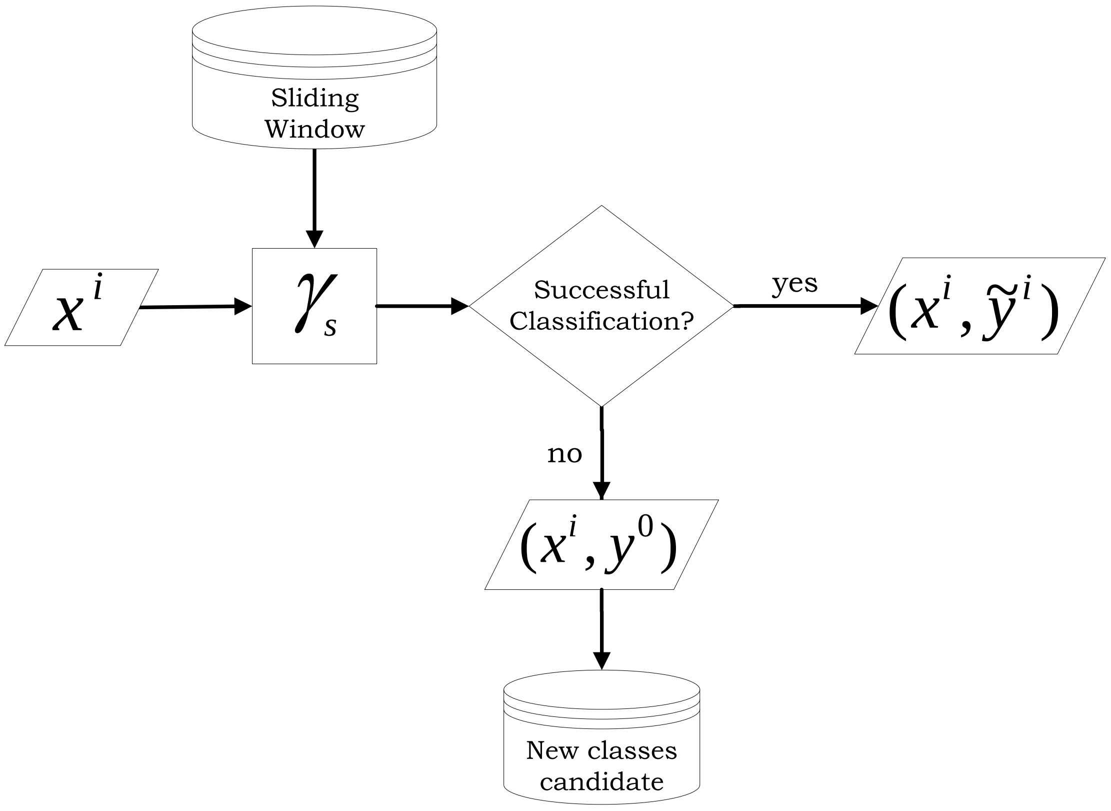


Figure 4.1: Process diagram for unlabeled records

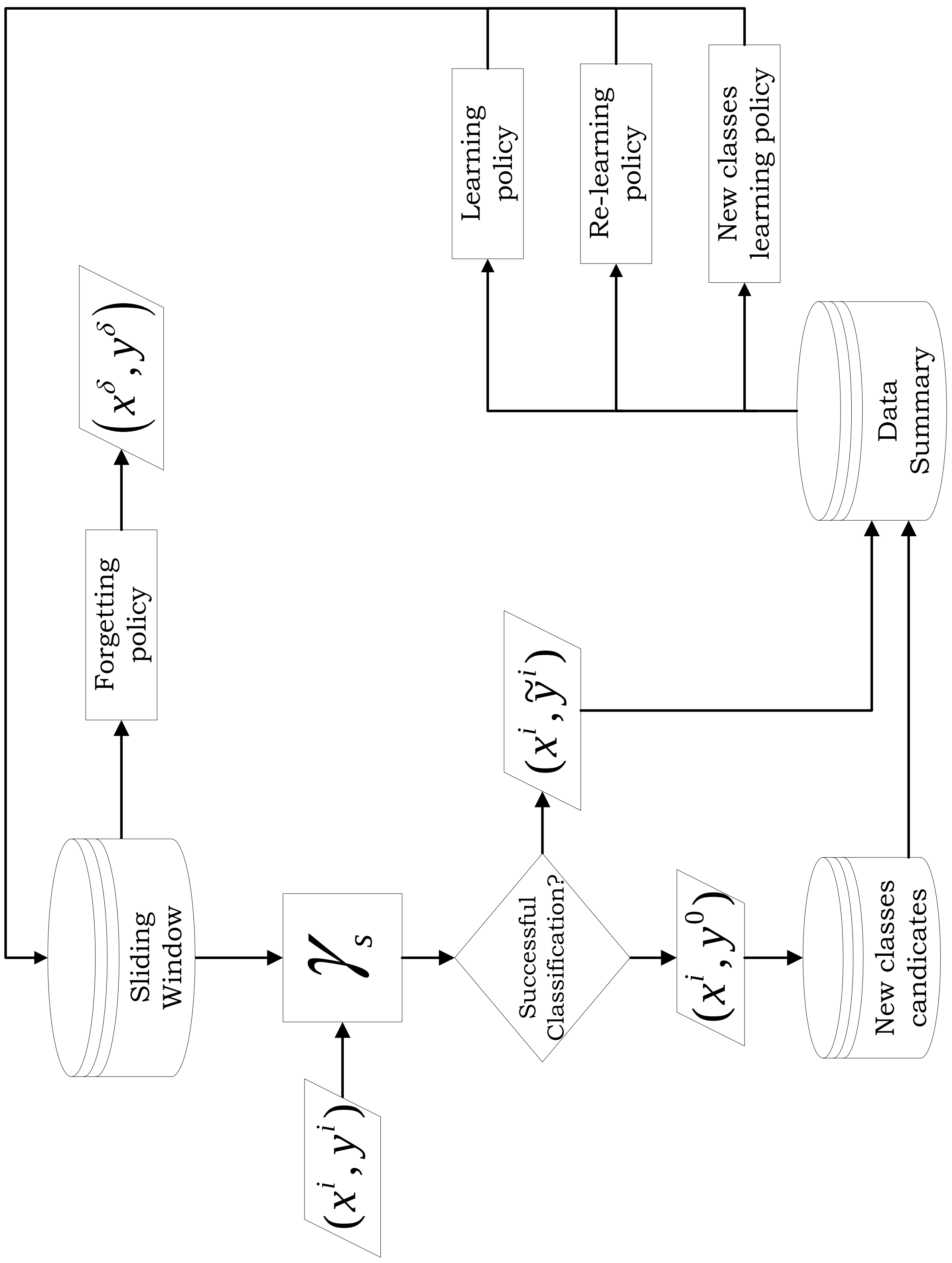


Figure 4.2: Process diagram for labeled records

*Arriving of labeled patterns.* When an input pattern $(x^i, y^i)$ arrives at time $i$ the DS-Gamma classifier will try to classify it. If classification is successful the model will output a class label $\tilde{y}^i$. The predicted class label $\tilde{y}^i$ will be used to assess the model using the prequential error over a sliding window, as described in section 3.4. It will also be used to update the sliding window and the summary data. The update will depend on the method of sliding window being used. Later in this section, the three proposed sliding window updating methods will be explained. Fig. 4.2 depicts the process for labeled patterns.

The following describes the proposed methods for updating the sliding window.

### 4.2.1 First approach: sliding window of fixed size

This first approach implements a sliding window of fixed size. This approach achieves concept drift detection in an indirect way, by considering only the most recent data. This is the simplest approach for sliding windows and it is generally presented as a base line for comparing with other methods. Algorithm 4.2.1 outlines the general steps used as learning policy. Initially, the windows is created with the first $w$ labeled records. Then the classifier attempts to classify the instance. Finally, if the instance has a class label assigned, this is used to update the window.

**Algorithm 4.2.1** Fixed size window's learning policy algorithm

```
1: Initialize window
2: for each $x^i$ do
3:     if $i < w$ then
4:         Add $x^i$ to the tail of the window.
5:     else
6:         Use $x^i$ to classify
7:         if $x^i$ is a labeled instance then
8:             Add $x^i$ to the tail of 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. The forgetting mechanisms can be divided in explicit and implicit. Explicit forgetting takes places when the examples are older than a user defined threshold. Implicit forgetting is performed by removing examples that are no longer relevant [23]. This approach assumes that the oldest information is less relevant, consequently the oldest item in the window is deleted. Algorithm 4.2.2 presents the basic forgetting policy implemented in this approach.

### 4.2.2 Second approach: sliding window using a statistical control process (SCP)

As a second approach, combining the Gamma classifier with a statistical process control to detect concept drift is proposed. This method to update a sliding window

**Algorithm 4.2.2** Fixed size window's forgetting policy algorithm

```
1: for each $x^i$ do
2:     if $i > w$ then
3:         Remove $x^{i-w+1}$ from the window.
```

was proposed by Gama *et al.* [8]. Algorithm 4.2.3 describes how the learning and forgetting mechanisms work.

**Algorithm 4.2.3** SCP algorithm

```
1: Let $(x^i, y^i)$ be the current example to classify
2: Let $\tilde{y}^i$ be the predicted class
3: Compute the mean error $p_i$ and variance $s_i$
4: if $p_i + s_i < p_{min} + s_{min}$ then
5:     $p_{min} \leftarrow p_i$
6:     $s_{min} \leftarrow s_i$
7: if $p_i + s_i < p_{min} + \beta * s_{min}$ then /* In control */
8:     $warning \leftarrow False$
9:     Add the example $(x^i, yi)$ to the sliding window
10:    if w ≥ max_window_size then
11:        Delete the oldest element in the sliding window $x^{i-w+1}$
12: else
13:    if $p_i + s_i < p_{min} + \alpha * s_{min}$ then /* Warning zone */
14:        if NOT $warning$ then
15:            $warning \leftarrow True$
16:        $buffer \leftarrow buffer \cup (x^i, y^i)$
17:        Add the example $(x^i, y^i)$ to the sliding window
18:        if w ≥ max_window_size then
19:            Delete the oldest element in the sliding window $x^{i-w+1}$
20:    else/* Out of control */
21:        Update the sliding window with the elements of $buffer$
22:        Clean the $buffer$
23:        $warning \leftarrow False$
24:        Reset $p_{min}$ and $s_{min}$
```

The method used in this section is an extension of the work presented in [77]. The error is calculated using a 0-1 loss function:

$$L(y, \hat{y}) = \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{otherwise} \end{cases}$$

### 4.2.3 Third approach: sliding window based on the Gamma similarity operator

This approach was devised to take advantage of one intrinsic characteristic of the Gamma classifier, the $\gamma$ similarity operator. Given a parameter of dissimilarity $\theta$, this operator indicates whether two vectors are similar or not.

During the classification process of a given instance $\tilde{x}$, an initial sum $c_\mu^0$ per each instance in the window and the arriving instances to be classified, was computed (see section 3.1.6). At the end of the classification process for $\tilde{x}$, this sum will hold a counter indicating how many components of the two compared instances are similar. This behavior can be exploited to compare the instances in the sliding window with the most recently classified instance and establish a grade of similarity between them. It is worth noting that this sum was already calculated during the classification process, thus it does not implicate an extra-processing step for the method.

For this work, empirically, a value of $c_\mu^0 = \sum_{j=1}^{n} \gamma_g \left( x_j^\mu, \tilde{x}_j, \theta \right) > n/2$ was chosen to consider that two instances are similar, where $n$ is the number of attributes of the instance. The instances will be considered dissimilar otherwise.

Based on the result of the sum of the similarity operator between the classified instance $\tilde{x}$ and a given instance of the sliding window $x^i$, and the classes to which each instance belongs, four types of instances was established:

- *Type 1.* This type is assigned when there is high similarity between $x^i$ and $\tilde{x}$ ($c_\mu^0 > n/2$) but they belong to different classes. These kinds of instances are always problematic for any classification algorithm, since they are similar but from different classes they will probably introduce ambiguity during the classification process. In this case $\tilde{x}$ represents the most recent knowledge and has to be incorporated to the window. Thus $x^i$ with type 1 will be a candidate to be discarded from the sliding window.

- *Type 2.* This type is assigned when there is low similarity between $x^i$ and $\tilde{x}$ ($c_\mu^0 \leq n/2$) and they belong to the same class. These instances can also be problematic because even when they belong to the same class, show few similarities between them. This situation is likely to indicate noise or an outlier. These instances will also become good candidates to be discarded from the sliding window.

- *Type 3.* This type is assigned when there is high similarity between $x^i$ and $\tilde{x}$ ($c_\mu^0 > n/2$) and they belong to the same class. These kinds of instances may not introduce uncertainty to the classification process but can represent redundant information. And given that $\tilde{x}$ represents the latest knowledge, it has to be incorporated to the window, thus the $x^i$ assigned with type 3 becomes a candidate for elimination as redundant information.

- *Type 4.* This type is assigned when there is low similarity between $x^i$ and $\tilde{x}$ ($c_\mu^0 \leq n/2$) and they belong to different classes. These are generally the most desirable kind of instances. For this method these type of instances will be the last candidate to be eliminated from the sliding window.

For this method, the instance type is considered an indicator of the level of usefulness of the instance in the sliding window and how much it can contribute to the classification process. Those instances of less help in the classification will be eliminated from the window. Fig. 4.3 shows the diagram for the process of assigning the instance types. Algorithm 4.2.4 describes the method to update a sliding window using the Gamma similarity approach.

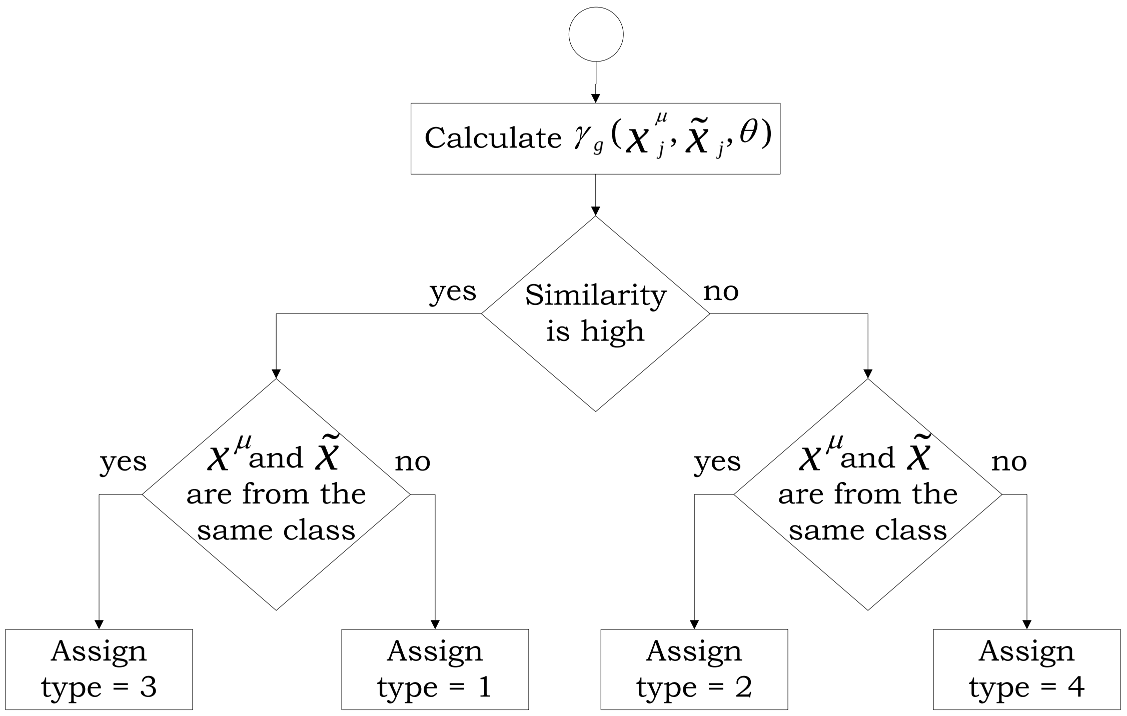

Calculate $\gamma_g(x_j^\mu, \tilde{x}_j, \theta)$
Similarity is high
yes
no
yes
$x^\mu$ and $\tilde{x}$ are from the same class
no
yes
$x^\mu$ and $\tilde{x}$ are from the same class
no
Assign type = 3
Assign type = 1
Assign type = 2
Assign type = 4

Figure 4.3: Block diagram for type assignation based on the $\gamma$ similarity operator

When this method was initially devised the hypothesis was that instances of type 1 will be the most harmful to the classification process. In order to demonstrate this assumption several experiments using the real and synthetic datasets were performed. Four rounds of experiments were performed for this method. In each round only one type of instance was selected to be removed from the window. Table 4.1 shows the results obtained from theses rounds of experiments.

**Algorithm 4.2.4** Gamma similarity algorithm

```
1: input: selected_type /* The type of instances to be deleted from the window*/
2: Let $(x^i, y^i)$ be the current example to classify
3: Let $\tilde{y}^i$ be the predicted class
4: for $j = 0$ to window_size do
5:     if $c_\mu^0 > n/2$ then
6:         if $y^i = y^j$ then
7:             $type^j \leftarrow 3$
8:         else
9:             $type^j \leftarrow 1$
10:    else
11:        if $y^i = y^j$ then
12:            $type^j \leftarrow 2$
13:        else
14:            $type^j \leftarrow 4$
15: for $j = 0$ to window_size do
16:     Select the oldest instance where $type^j = selected\_type$
17:
18: if NOT select instance of selected_type then
19:     Delete the oldest element in the sliding window $x^{i-w+1}$
```

Table 4.1: Accuracy comparison of the similarity method using different instance types

| Data stream | Agrawal | | Bank | | CoverType | | Electricity | | Hyperplane | | RandomRBF | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | $\sigma$ | Acc. | $\sigma$ | Acc. | $\sigma$ | Acc. | $\sigma$ | Acc. | $\sigma$ | Acc. | $\sigma$ |
| DS-Gamma (Fixed size) | 92.82 | 0.10 | 78.96 | 0.22 | 47.77 | 4.31 | 77.14 | - | 86.30 | 1.35 | 70.85 | 2.54 |
| DS-Gamma (SPC) | 93.36 | 0.22 | 85.33 | 0.15 | **50.67** | 10.52 | **84.11** | - | 87.19 | 1.19 | 71.53 | 2.58 |
| DS-Gamma (Similarity-Type1) | **94.42** | 0.19 | **86.07** | 0.06 | 44.01 | 8.66 | 80.18 | - | **87.31** | 1.34 | **72.19** | 2.53 |
| DS-Gamma (Similarity-Type2) | 93.95 | 0.09 | 86.01 | 0.13 | 46.88 | 7.33 | 79.00 | - | 87.16 | 1.38 | 71.56 | 2.57 |
| DS-Gamma (Similarity-Type3) | 93.91 | 0.11 | 85.87 | 0.14 | 45.28 | 6.76 | 78.36 | - | 87.19 | 1.37 | 71.48 | 2.61 |
| DS-Gamma (Similarity-Type4) | 66.40 | 0.11 | 85.69 | 0.24 | 44.54 | 7.43 | 81.45 | - | 86.92 | 1.39 | 70.09 | 2.84 |

From the results shown in table 4.1, it can be observed that, as expected, when instances of type 1 are selected to be remove from the window, the classification performance was improved. Therefore, type 1 was selected for all the experiments of this method.

# Chapter 5

# Experimental results

In this chapter the results of some of the performed experiments with the proposed methodology are presented. A comparative assessment with other representative algorithms for data stream context is performed taking into account the non-stationary distribution of the data and the memory and time constrain for this kind of algorithms.

## 5.1 Experimental design

One of the specific objectives of this work is to perform a consistent comparative analysis 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 assessment has been mainly limited to evaluating a model using different random reordering of the same static dataset ($k$-fold cross validation, leave-one-out, and bootstrap) [7]. For data stream scenarios the problem of possible infinite data raises new challenges. On the other hand, the batch evaluation 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 [6], two main approaches arise.

*Holdout.* In traditional batch learning when the volume of the data 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 subsequence $N$ instances and so forth.

*Interleaved test-then-train or prequential.* In this approach, 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 that have not been seen.

In section 3.4, several evaluation methods for data stream contexts were discussed, including the prequential error estimated over a sliding window. Since the methodology proposed in this work used a sliding window approach, this evaluation metric was chosen to assess the proposed methodology. The prequential error $P_w$ between the predicted $\hat{y}_k$ values and the observed values $y_k$ was calculated using the following expression [87] where $L$ is a $0-1$ loss function:

$$P_w(i) = \frac{1}{w}\sum_{k=i-w+1}^{i} L(y_k, \hat{y}_k) = \frac{1}{w}\sum_{k=i-w+1}^{i} e_k$$

To ensure a valid comparison of classification performance, the same conditions and validation schemes were applied in each experiment. Classification performance of each of the compared algorithms, including our proposal, was calculated using the prequential error approach. These results will be used to perform a comparative study on the performance of our proposal and other data stream classification algorithms. The selection of algorithms to be compared with our proposal, was performed with the idea of covering the broad spectrum of approaches commonly used in the current literature for data stream contexts, such as Hoeffding tree, instance based learning, Naïve Bayes with DDM (Drift Detection Method), SVM, and ensemble methods. All of these algorithms are implemented under the MOA framework. A brief description of the used algorithms can be found in section 3.5. Further details on the implementation of these algorithms can be found in [106].

For all data streams used during the experimental phase, 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 [111] for each experiment. For the synthetics data streams (agrawal, rotating hyperplane, and randomRBF), 10 data streams were generated using the MOA platform; for each one of them a different random seed was used. Performance was calculated using the prequential error introduced in section 3.4. The total time used for classification was also measured 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 4096GB of RAM.

## 5.2 Results and discussion

In this section, we present and discuss the results obtained by the DS-Gamma classifier 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 methodology to the change on the windows size when using a fixed size sliding window, which is a parameter that can largely influence the performance of the classifier. Later, this evaluation was used to select the window size for the fixed size window approach. Table 5.1 shows the average accuracy of the DS-Gamma classifier with fixed window sizes for different window sizes; best accuracy for each data stream is emphasized with boldface. Table 5.2 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 5.1.

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 in the set of experiments with a window of size equal to 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 Fig. 5.1. On the other hand the increase in the size of the windows will cause an increase in the execution time. These factors are important to take into consideration to decide in a tradeoff between time and accuracy. A size of 1000 instances was selected to perform subsequent experiments. Fig. 5.2 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 for 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.

Table 5.1: DS-Gamma (fixed size) accuracy for different window sizes

| Data stream | 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 |

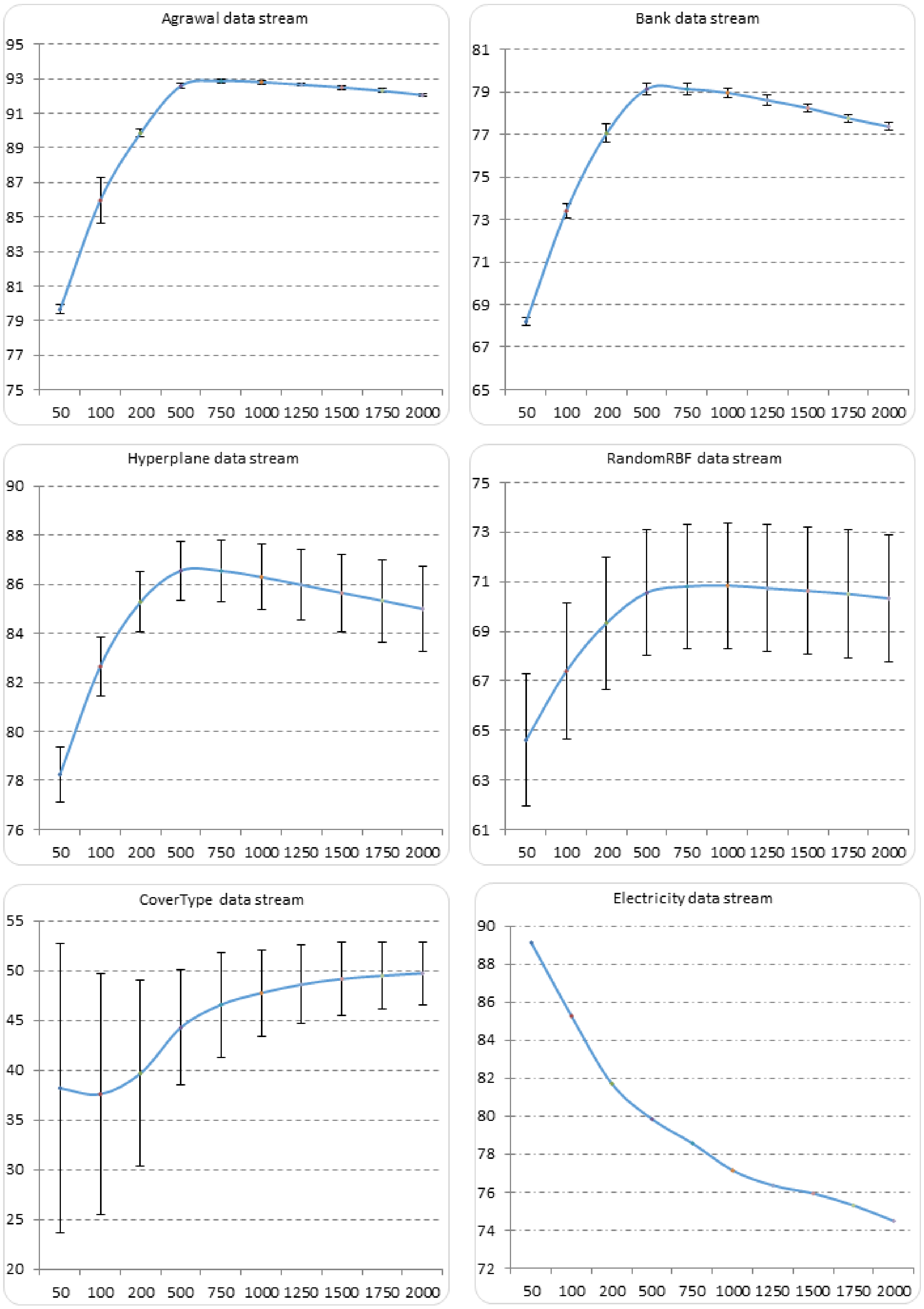


Figure 5.1: DS-Gamma (fixed size) accuracy for different window sizes

Table 5.2: DS-Gamma (fixed size) classification time (s) for different window sizes

| Data stream | 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 |

To evaluate the DS-Gamma classifier, we compared its performance with other data stream classifiers. Table 5.3 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 boxplot with the same results in Fig. 5.3. For the Agrawal data stream, the DS-Gamma classifier using the approach of similarity achieved the 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 5.4 and Fig. 5.4. 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 (12) of the Gamma classifier algorithm presented in section 3.1.2, 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.

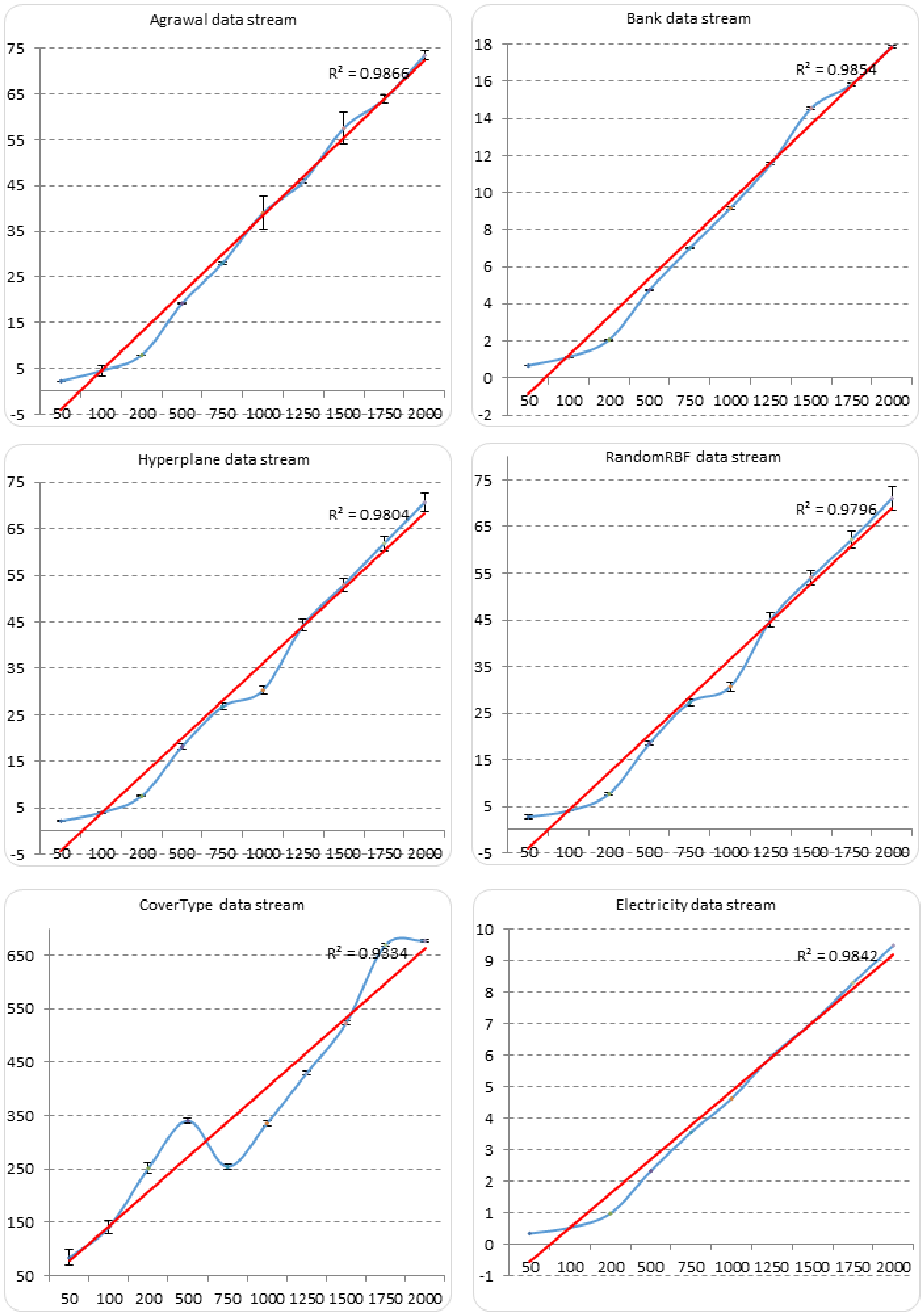


Figure 5.2: DS-Gamma (fixed size) classification time (s) for different window sizes

Table 5.3: Accuracy comparison for the DS-Gamma

| Data stream | Agrawal | | Bank | | CoverType | | Electricity | | Hyperplane | | RandomRBF | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | $\sigma$ | Acc. | $\sigma$ | Acc. | $\sigma$ | Acc. | $\sigma$ | Acc. | $\sigma$ | Acc. | $\sigma$ |
| 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 |
| SPegaso | 55.82 | 0.17 | 77.46 | 1.99 | 50.47 | 5.17 | 57.60 | - | 52.80 | 6.08 | 53.33 | 3.13 |
| DS-Gamma (Fixed size) | 92.82 | 0.10 | 78.96 | 0.22 | 47.77 | 4.31 | 77.14 | - | 86.30 | 1.35 | 70.85 | 2.54 |
| DS-Gamma (SPC) | 93.36 | 0.22 | 85.33 | 0.15 | 50.67 | 10.52 | 84.11 | - | 87.19 | 1.19 | 71.53 | 2.58 |
| DS-Gamma (Similarity) | **94.42** | 0.08 | 86.07 | 0.06 | 44.01 | 8.66 | 80.18 | - | **87.31** | 1.34 | 72.19 | 2.53 |

Table 5.4: Time (s) comparison for the DS-Gamma

| Data stream | Agrawal | | Bank | | CoverType | | Electricity | | Hyperplane | | RandomRBF | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | $\sigma$ | Time | $\sigma$ | Time | $\sigma$ | Time | $\sigma$ | Time | $\sigma$ | Time | $\sigma$ |
| 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 |
| SPegaso | 0.22 | 0.006 | 0.11 | 0.016 | 1.684 | 0.12 | 0.12 | - | 0.31 | 0.05 | 0.24 | 0.01 |
| DS-Gamma (Fixed size) | 39.00 | 3.58 | 9.16 | 0.05 | 335.03 | 12.91 | 4.63 | - | 30.28 | 0.79 | 30.72 | 0.94 |
| DS-Gamma (SPC) | 112.62 | 5.61 | 29.47 | 0,04 | 1662.78 | 444.99 | 9.04 | - | 86.24 | 3.96 | 104.83 | 3.96 |
| DS-Gamma (Similarity) | 143.13 | 0.82 | 34.25 | 0.08 | 1918.61 | 8.43 | 24.18 | - | 114.79 | 3.79 | 72.19 | 2.53 |

From the results in Table 5.3, we can observe that the performance obtained by the classifier using the SCP and the similarity approach has improved when compared to that shown by the DS-Gamma classifier using a fixed size window approach. The performance when using the new method to update the sliding window based on the similarity operator shows goods results. In 4 of the 6 tested data streams, this approach shows better results that the other two proposed methods for updating the sliding window (fixed size and SCP).

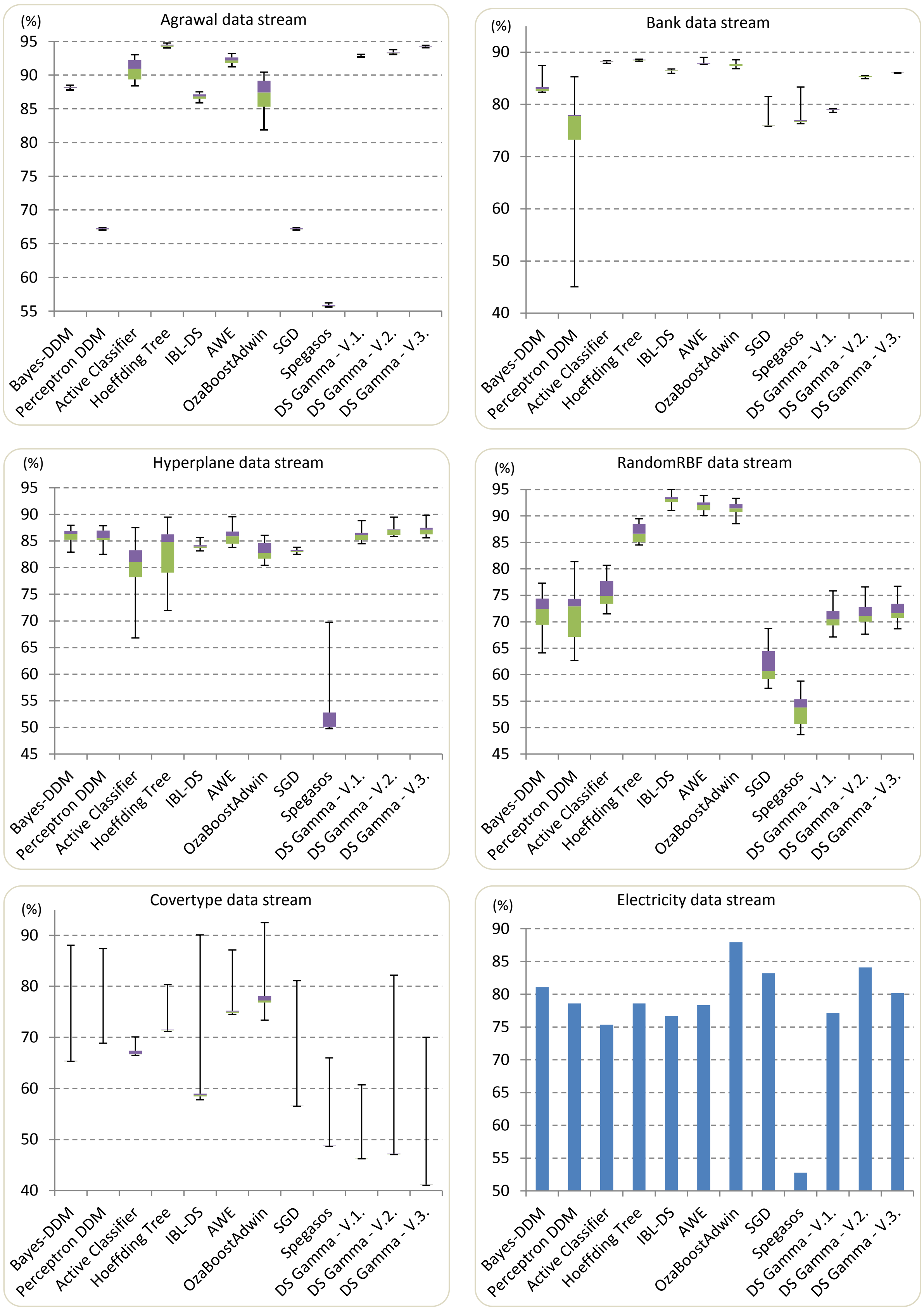


Figure 5.3: DS-Gamma accuracy comparison

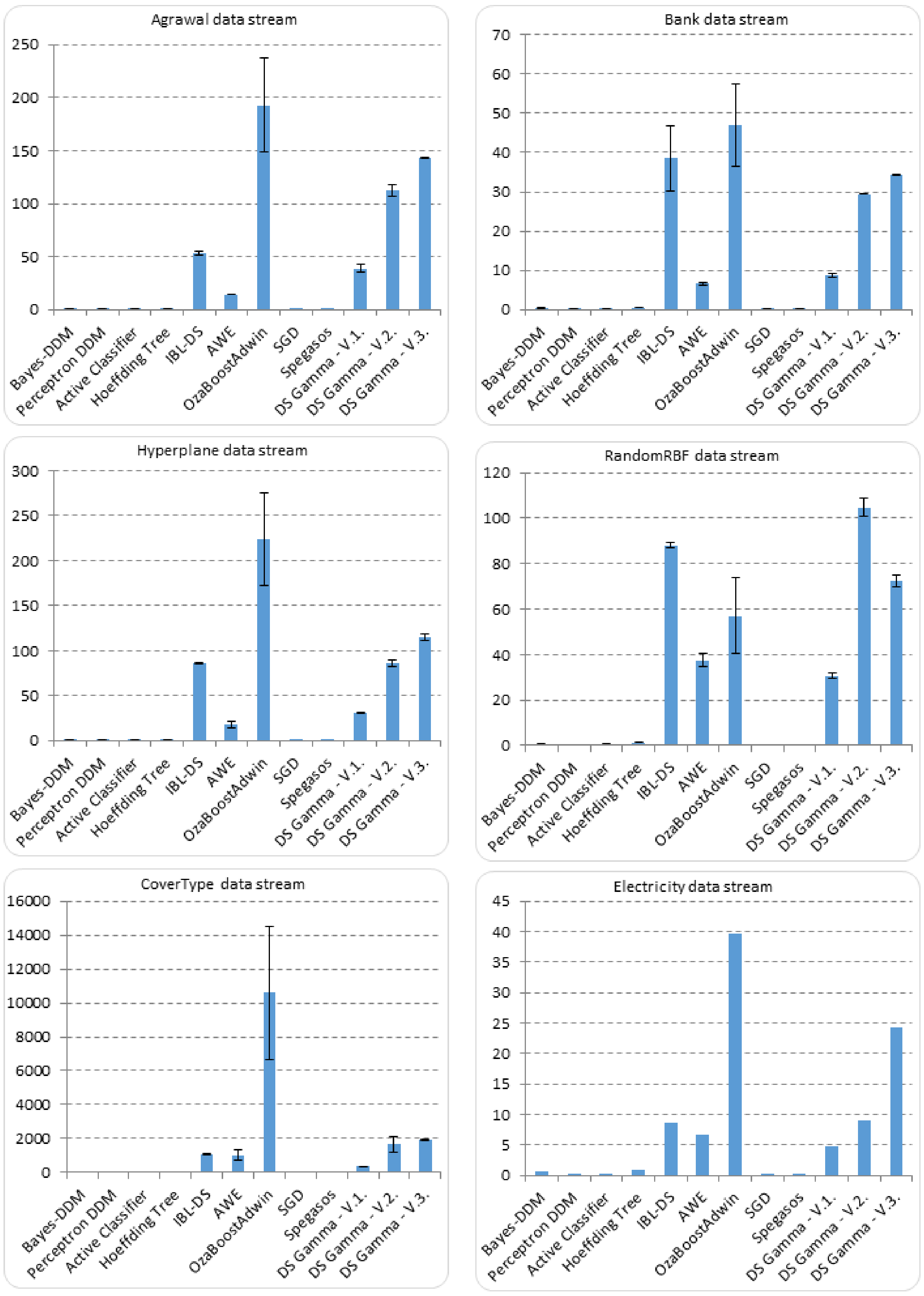


Figure 5.4: DS-Gamma classification time comparison

To statistically compare the results obtained by the DS-Gamma classifier and the other evaluated algorithms the Wilcoxon signed-ranks test was used. Tables 5.5, 5.6, and 5.7 show the values of the asymptotic significance (2-sided) obtained by the Wilcoxon test using a level of confidence $\alpha$ = 0.05. In the case of the DS-Gama with the approach of fixed size window (Table 5.5), the obtained values are all greater than $\alpha$, so the null hypothesis is not rejected and we can infer that there are no significant differences among the compared algorithms.

For the case of the DS-Gamma with SCP and similarity approach (Table 5.6 and 5.7) the only value smaller than the level of significance $\alpha$ was obtained when compared with the SPegasos classifier. In this case the null hypothesis is rejected meaning that there is a statistically significant difference between the two classifiers.

Table 5.5: Test de Wilcoxon for DS-Gamma (fixed window)

| Compared algorithms | Asymptotic significance (2-sided) |
|---|---|
| DS-Gamma - Bayes-DDM | 0.345 |
| DS-Gamma - Perceptron-DDM | 0.917 |
| DS-Gamma - Hoeffding Tree | 0.116 |
| DS-Gamma - Active Classifier | 0.463 |
| DS-Gamma - IBL-DS | 0.345 |
| DS-Gamma - AWE | 0.116 |
| DS-Gamma - OzaBoostAdwin | 0.116 |
| DS-Gamma - SGD | 0.753 |
| DS-Gamma - SPegasos | 0.116 |
| DS-Gamma - DS-Gamma (SCP) | 0.116 |
| DS-Gamma - DS-Gamma (Similarity) | 0.249 |

Table 5.6: Test de Wilcoxon for DS-Gamma (SCP)

| Compared algorithms | Asymptotic significance (2-sided) |
|---|---|
| DS-Gamma - Bayes-DDM | 0.753 |
| DS-Gamma - Perceptron-DDM | 0.600 |
| DS-Gamma - Hoeffding Tree | 0.345 |
| DS-Gamma - Active Classifier | 0.917 |
| DS-Gamma - IBL-DS | 0.463 |
| DS-Gamma - AWE | 0.249 |
| DS-Gamma - OzaBoostAdwin | 0.345 |
| DS-Gamma - SGD | 0.116 |
| DS-Gamma - SPegasos | 0.028 |
| DS-Gamma - DS-Gamma (Fixed window) | 0.116 |
| DS-Gamma - DS-Gamma (Similarity) | 0.917 |

Table 5.7: Test de Wilcoxon for DS-Gamma (fixed window)

| Compared algorithms | Asymptotic significance (2-sided) |
|---|---|
| DS-Gamma - Bayes-DDM | 0.753 |
| DS-Gamma - Perceptron-DDM | 0.463 |
| DS-Gamma - Hoeffding Tree | 0.173 |
| DS-Gamma - Active Classifier | 0.917 |
| DS-Gamma - IBL-DS | 0.463 |
| DS-Gamma - AWE | 0.345 |
| DS-Gamma - OzaBoostAdwin | 0.173 |
| DS-Gamma - SGD | 0.345 |
| DS-Gamma - SPegasos | 0.046 |
| DS-Gamma - DS-Gamma (Fixed window) | 0.249 |
| DS-Gamma - DS-Gamma (SCP) | 0.917 |

# Chapter 6

# Conclusions and future work

In this chapter we present the resulting conclusions of the research work presented in this thesis. We also propose some ideas for future work related to the research topic in this thesis. These proposals can be guideline for the development of new research works that investigate some aspects of the topic not covered in the thesis.

## 6.1 Conclusions

In this work, we describe a novel methodology for the task of pattern classification over a continuous data stream based on an associative model. The proposed methodology is a supervised pattern recognition model based on the Gamma classifier combined with different methods to update a sliding window. One of the methods is an original proposal based on the $\gamma$ similarity operator to determine the usefulness of a pattern in the sliding window. Given the reliance of the proposed methodology on the Gamma classifier as a base learner, this proposal has been given the name DS-Gamma (Data Stream Gamma) classifier.

An extensive study of the state of the art was performed and to our knowledge this is the first time that an associative approach has been used to address data stream classification problems. The main approaches used for data stream mining are the following: decision models, ensemble methods, instance based models, clustering methods, neural network models, support vector machines, frequent pattern mining, and other approaches such as fuzzy, genetic, and bioinspired algorithms, which address seldom studied aspects of data stream mining.

Decision trees are one of the most popular methods for data stream classification, followed by ensembles methods and instance based learning; for the latter, incremental learning and model adaptation is very simple, since it comes to updating the stored instances. For data stream scenarios, neural network-based models have been scarcely used, due to its limitation for these contexts, such as high computational cost and the time to update the model. On the other hand, SVMs have been used for different tasks in data stream scenarios such as sentiment analysis, fault detection and prediction, medical applications, spam detection, and multi-label classification.

Most of current machine learning algorithms assume that training data was generated from a stationary distribution; however, two of the main issues in learning from a data stream are concept drift and concept evolution, which imply non-stationary distributions. Detection of concept drift in data streams with class imbalance is a challenge and it is a topic of intensive research. Most existing classification algorithms for data streaming assume that the class labels are immediately available; however, for real world applications, this is not always true. The major difficulties in data stream classification are related to memory space, processing and classification time, and change/evolution of the target concepts. In this context, classification algorithms have to be able to extract knowledge with only one, or few, passes over the data in a swift manner and using as little resources as possible. In general terms, the modularity and simplicity of the proposed methodology is a great advantage over the other compared methods of the state of the art.

After performing a study of the datasets commonly used for the task of pattern classification on data streams, a group of 3 real and 3 synthetic datasets were selected, which were used to test the proposed methodology. The number of instances of these datasets ranges from 45,211 to 100,000, and the number of attributes goes from 8 to 54. With respect to number of classes, one of the datasets has 7 classes, while the other five datasets have 2 classes.

During the experimental phase, the proposed methodology was tested using different data stream scenarios with the aforementioned real and synthetic data streams. The proposed methodology presented competitive results in terms of performance and classification time when compared with some of the state-of-the-art algorithms for data stream classification.

Three sliding window updating methods were tested in combination with the Gamma classifier: fixed size window, statistical controlled process, and Gamma similarity update; being the last one an original proposal introduced in this thesis based on the Gamma similarity operator to determine the usefulness of the patterns in the sliding window.

For the fixed size window method, the effect of the window size over the classification accuracy and time was studied. 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. For the Gamma similarity method, it was expected to be affected by the type of instances selected to be removed from the window. Several experiments were performed selecting a different type of instance each time to observe the impact on the classification performance. From the results obtained, as expected, when instances of type 1 (high similarity, different classes) are selected to be remove from the window, the classification performance was improved.

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 forest covertype dataset. This is a heavily imbalance dataset, where the accuracy of the classifier was the worst of the compared algorithms.

Despite of the good performance of the DS-Gamma classifier, there are still some limitations to tackle:

- When the values in the training set do not induce a function, the performance of the classifier is negatively affected. It is worth noting that this issue is also true for most classification models.
- The values of some parameters such as the weights, the stop parameter and the similarity criteria, are still determined with a process of trial and error that is a very time consuming process.
- Heavily unbalanced datasets will degrade the performance of the classifier.

## 6.2 Future Work

The methodology presented in this thesis fulfilled the goals proposed at the beginning of this work. However, the results obtained have opened several lines of research that would make it possible to extent the initial proposal to address related issues. The following summarizes some of these future lines of research.

The execution time showed by the DS-Gamma classifier can be improved. An optimization of the algorithm to improve the classification time is proposed as future work.

We believed that combining the strategy of similarity to determine the usefulness of the patterns with a method to variable adapt the sliding window, such as SCP, can lead to an increase in the performance of the DS-Gamma classifier. As future work, an implementation of a method to update a sliding window combining these two approaches can be carried out.

For data stream algorithms, one important aspect is the amount of memory used. The DS-Gamma uses a fixed size amount of memory, but a detailed evaluation to exactly measure the memory consumption of the proposed methodology should be performed.

Develop an appropriate method for automatic estimation of the values for the parameters of the DS-Gamma classifier such as weights, stop parameter, and similarity criteria.

Test an updating sliding window method based on monitoring distributions on two different time-windows. One possible method is the one proposed by Kuncheva in [10], where two criteria to estimate the probability of the change between two windows, Kullback-Leibler distance [83] and Hotelling's T-square test for equal means [84] are introduced.

Considering how imbalance datasets affect the DS-Gamma classifier, a mechanism to address severe class imbalance should be integrated to the methodology to improve classification accuracy.

# References

[1] **Turner, V., Gantz, J.F., Reinsel, D. & Minton, S. (2014).** The digital universe of opportunities: rich data and the increasing value of the internet of things. Technical report, IDC Information and Data. Retrieved from http://www.emc.com/leadership/digital-universe/2014iview/index.htm

[2] **Alon, N., Matias, Y. & Szegedy, M. (1999).** The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1), 137-147.

[3] **Henzinger, M., Raghavan, P. & Rajagopalan, S. (1999).** Computing on data streams. *External Memory Algorithms*, 107-118. Boston: American Mathematical Society.

[4] **Feigenbaum, J., Kannan, S., Strauss, M. & Viswanathan, M. (1999).** An approximate L1-difference algorithm for massive data streams. *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, New York City, USA, 501-511.

[5] **Beringer, J. & Hüllermeier, E. (2007).** Efficient instance-based learning on data streams. *Intelligent Data Analysis*, 11(6), 627-650.

[6] **Bifet, A., Holmes, G., Kirkby, R. & Pfahringer, B. (2011).** *Data stream mining: a practical approach.* Technical report, University of Waikato.

[7] **Bifet, A. (2010).** *Adaptive stream mining: pattern learning and mining from evolving data streams.* Netherlands: IOS Press.

[8] **Gama, J., Medas, P., Castillo, G. & Rodrigues, P. (2004).** Learning with drift detection. *Advances in Artificial Intelligence - 17th Brazilian Symposium on Artificial Intelligence. Lecture Notes in Computer Science*, Maranhao, Brazil, 3171, 286-295

[9] **He, H., Chen, S., Li, K. & Xu, X. (2011).** Incremental learning from stream data. *IEEE Transactions on Neural Networks*, 22(12), 1901-1914.

[10] **Kuncheva, L.I. (2013).** Change detection in streaming multivariate data using likelihood detectors. *IIEEE Transactions on Knowledge and Data Engineering*, 25(5), 1175-1180.

[11] **Ross, G.J., Adams, N.M., Tasoulis, D.K. & Hand, D.J. (2012).** Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2), 191-198.

[12] **Domingos, P. & Hulten, G. (2000).** Mining high-speed data streams. *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining*, Boston, Massachusetts, USA, 71-80.

[13] **Hulten, G., Spencer, L. & Domingos, P. (2001).** Mining time-changing data streams. *Proceedings of the 7th ACM International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, USA, 97-106.

[14] **Liang, C., Zhang, Y., Shi, P. & Hu, Z. (2012).** Learning very fast decision tree from uncertain data streams with positive and unlabeled samples. *Information Sciences*, 213, 50-67.

[15] **Li, C., Zhang, Y. & Li, X. (2009).** OcVFDT: One-class very fast decision tree for one-class classiffication of data streams. *Proceeding of the 3th International Workshop on Knowledge Discovery from Sensor Data*, Paris, France, 79-86.

[16] **Qin, X., Zhang, Y., Li, C. & Li, X. (2013).** Learning from data streams with only positive and unlabeled data. *Journal of Intelligent Information Systems*, 40(3), 405-430.

[17] **Read, J., Bifet, A., Holmes, G. & Pfahringer, B. (2012).** Scalable and efficient multi-label classification for evolving data streams. *Machine Learning*, 88(1-2), 243-272.

[18] **Wu, X., Li, P. & Hu, X. (2012).** Learning from concept drifting data streams with unlabeled data. *Neurocomputing*, 92, 145-155.

[19] **Rutkowski, L., Jaworski, M., Pietruczuk, L. & Duda, P. (2014).** Decision trees for mining data streams based on the gaussian approximation.*IEEE Transactions on Knowledge and Data Engineering*, 26(1), 108-119.

[20] **Rutkowski, L., Jaworski M., Pietruczuk, L. & Duda, P. (2014).** The CART decision tree for mining data streams. *Information Sciences*, 266, 1-15.

[21] **Shaker, A., Senge, R. & Hüllermeier, E. (2013).** Evolving fuzzy pattern trees for binary classification on data streams. *Information Sciences*, 220, 34-45.

[22] **Widmer, G. & Kubat, M. (1996).** Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1), 69-101.

[23] **Ferrer-Troyano, F., Aguilar-Ruiz, J.S. & Riquelme, J.C. (2006).** Data streams classification by incremental rule learning with parameterized generalization. *Proceedings of the 2006 ACM Symposium on Applied computing*, Dijon, France, 657-661.

[24] **Tomczak, J.M. & Gonczarek, A. (2013).** Decision rules extraction from data stream in the presence of changing context for diabetes treatment. *Knowledge and Information Systems*, 34(3), 521-546.

[25] **Abdulsalam, H., Skillicorn, D.B. & Martin P. (2011).** Classification using streaming random forests. *IEEE Transactions on Knowledge and Data Engineering*, 23(1), 22-36.

[26] **Brzezinski D. & Stefanowski, J. (2014).** Combining block-based and online methods in learning ensembles from concept drifting data streams. *Information Sciences*, 265, 50-67.

[27] **Ditzler, G. & Polikar, R. (2013).** Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10), 2283-2301.

[28] **Farid, D.Md., Zhang, L., Hossain, A., Rahman C.M., Strachan, R., Sexton, G. & Dahal, K. (2013).** An adaptive ensemble classifier for mining concept drifting data streams. *Expert Systems with Applications*, 40(15), 5895-5906.

[29] **Ghazikhani, A., Monsefi, R., & Yazdi, H.S. (2013).** Ensemble of online neural networks for non-stationary and imbalanced data streams. *Neurocomputing*, 122, 535-544.

[30] **Masud, M.M., Gao, J., Khan, L., Han, J. & Thuraisingham, B. (2011)**. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transaction on Knowledge and Data Engineering*, 23(6), 859-874.

[31] **Masud, M.M., Woolam, C., Gao, J., Khan, L., Han, J., Hamlen, K.W. & Oza, N.C. (2012).** Facing the reality of data stream classification: coping with scarcity of labeled data. *Knowledge and Information Systems*, 33(1), 213-244.

[32] **Hosseini, M.J., Ahmadi, Z. & Beigy, H. (2013).** Using a classifier pool in accuracy based tracking of recurring concepts in data stream classification. *Evolving Systems*, 4(1), 43-60.

[33] **Sancho-Asensio, A., Orriols-Puig, A. & Golobardes, E. (2014).** Robust on-line neural learning classifier system for data stream classification tasks. *Soft Computing*, 1-21.

[34] **Saunier, N. & Midenet, S. (2013).** Creating ensemble classifiers through order and incremental data selection in a stream. *Pattern Analysis and Applications*, 16(3), 333-347.

[35] **Mena-Torres, D. & Aguilar-Ruiz, J.S. (2014).** A similarity-based approach for data stream classification. *Expert Systems with Applications*, 41(9), 4224-4234.

[36] **Shaker, A. & Hüllermeier, E. (2012).** BLStreams: a system for instance-based classification and regression on data streams. *Evolving Systems*, 3(4), 235-249.

[37] **Ding, S., Wu, F., Qian, J., Jia, H. & Jin, F. (2013).** Research on data stream clustering algorithms. *Artificial Intelligence Review*, 1-8.

[38] **Ackermann, M.R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C. & Sohler, C. (2012).** StreamKM++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics*, 17, 2.4:2.1-2.4:2.30.

[39] **Wang, C.D., Lai, J.H., Huang, D. & Zheng, W.S. (2013).** SVStream: A support vector-based algorithm for clustering data streams. *IEEE Transactions on Knowledge and Data Engineering*, 25(6), 1410-1424.

[40] **Miller, Z., Dickinson, B., Deitrick, W., Hu, W. & Wang, A.H. (2014).** Twitter spammer detection using data stream clustering, *Information Sciences*, 260, 64-73.

[41] **Cao, F., Ester, M., Qian, W. & Zhou, A. (2006)** Density-based clustering over an evolving data stream with noise. *Proceedings of the 6th SIAM International Conference on Data Mining*, Bethesda, Maryland, USA, 328-339.

[42] **Ghazikhani A., Monsefi, R. & Yazdi, H.S. (2013).** Recursive least square perceptron model for non-stationary and imbalanced data stream classification. *Evolving Systems*, 4(2), 119-131.

[43] **Zhang, P., Gao, B.J., Liu, P., Shi, Y. & Guo, L. (2012).** A framework for application-driven classification of data streams. *Neurocomputing*, 92, 170-182.

[44] **Kranjc, J., Smailović, J., Podpečan, V., Grčar, M., Žnidaršič, M. & Lavrač, N. (2015).** Active learning for sentiment analysis on data

streams: methodology and workflow implementation in the ClowdFlows platform. *Information Processing and Management*, 51(2), 187-203.

[45] **Smailović, J., Grčar, M., Lavrač, N. & Žnidaršič, M. (2015).** Stream based active learning for sentiment analysis in the financial domain. *Information Sciences*, 285, 181-203.

[46] **Shalev-Shwartz, S., Singer, Y., Srebro, N. & Cotter, A. (2007).** Pegasos: primal estimated sub-gradient solver for SVM. *Proceedings of the 24th International Conference on Machine Learning*, 807-814.

[47] **Krawczyk, B. & Woźniak, M. (2015).** Incremental weighted oneclass classifier for mining stationary data streams, *Journal of Computational Science*, 9, 19-25.

[48] **Nasreena, S., Azamb, M.A., Shehzada, K., Naeemc, U. & Ghazanfara, M.A. (2014).** Frequent Pattern Mining Algorithms for Finding Associated Frequent Patterns for Data Streams: A Survey. *Procedia Computer Science*, 37, 109-116.

[49] **Agrawal, R. & Srikant, R. (1994).**Fast algorithm for mining association rules in large databases. *Proceeding of the 20th International Conference on Very Large Data Bases*, San Francisco, California, USA, 487-499.

[50] **Nori, F., Deypir, M. & Sadreddini, M.H. (2013).** A sliding window based algorithm for frequent closed itemset mining over data stream. *The Journal of Systems and Software*, 86(3), 615-623.

[51] **Chao-Wei, L. & Kuen-Fang, J. (2014).**An approach of support approximation to discover frequent patterns from concept-drifting data streams based on concept learning. *Knowledge and Information Systems*, 40(3), 639-671.

[52] **Lee, G., Yun, U. & Ryu, K.H. (2014).** Sliding window based weighted maximal frequent pattern mining over data streams. *Expert Systems with Applications*, 41(2), 694-708.

[53] **Jang, J.S.R. (1993).** ANFIS : Adaptative-Network-Based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3), 665-685.

[54] **Rong, H.J., Sundararajan, N., Huang, G.B. & Saratchandran, P. (2006).** Sequential Adaptive Fuzzy Inference System (SAFIS) for nonlinear system identification and prediction. *Fuzzy Sets and Systems*, 157(9), 1260-1275.

[55] **Wang, D., Zeng, X.J. & Keane, J.A. (2010).** A structure evolving learning method for fuzzy systems. *Evolving Systems*, 1(2), 83-95.

[56] **Angelov, P. (2011).** Fuzzily connected multimodel systems evolving autonomously from data streams. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 41(4), 898-910.

[57] **Vivekanandan, P. & Nedunchezhian, R. (2011).** Mining data streams with concept drifts using genetic algorithm. *Artificial Intelligence Review*, 36(3), 163-178.

[58] **Zhao, L., Wang, L. & Xu, Q. (2012).** Data stream classification with artificial endocrine system. *Applied Intelligence*, 37(3), 390-404.

[59] **Zliobaite, I. & Gabrys, B. (2014).** Adaptive preprocessing for streaming data. *IEEE Transactions on Knowledge and Data Engineering*, 26(2), 309-321.

[60] **Steinbuch, K. (1961).** Die Lernmatrix. *Kybernetik*, 1(1), 36-45.

[61] **Willshaw, D., Buneman, O. & Longuet-Higgins, H. (1969).** Non-holographic associative memory. *Nature*, 222, 960-962.

[62] **Anderson, J.A. (1972).** A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14(3-4), 197-220.

[63] **Kohonen, T. (1972).** Correlation matrix memories. *IEEE Transactions on Computers*, C-21(4), 353-359.

[64] **Nakano, K. (1972).** Associatron - A model of associative memory. *IEEE Transactions on Systems, Man, and Cybernetics*, 2(3), 380-388.

[65] **Amari, S. (1972).** Learning patterns and pattern sequences by selforganizing nets of threshold elements. *IEEE Transactions on Computers*, C-21(11), 1197-1206.

[66] **Hopfield, J.J. (1982).** Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8), 2554-2558.

[67] **Ritter, G.X., Sussner, P. & Diaz-de-Leon, J. L. (1998).** Morphological associative memories. *IEEE Transactions on Neural Networks*, 9(2), 281-293.

[68] **Yáñez-Márquez, C. (2002).** *Memorias asociativas basadas en relaciones de orden y operadores binarios.* PhD thesis in Computer Science, Instituto Politécnico Nacional, Centro de Investigación en Computación, México D.F., México.

[69] **López-Yáñez, I. (2011).** *Teoría y aplicaciones del clasificador Gamma.* PhD thesis in Computer Science, Instituto Politécnico Nacional, Centro de Investigación en Computación, México D.F., México.

[70] **Aldape-Pérez, M., Yáñez-Márquez, C. & López-Leyva, L.O. (2006).** Feature selection using a hybrid associative classifier with masking technique. *5th Mexican International Conference on Articial Intelligence*, Apizaco, México, 151-160.

[71] **Yáñez-Márquez, C., Felipe-Riverón, E.M., López-Yáñez, I. & Flores-Carapia, R. (2006).** A novel approach to automatic color matching. *Progress in Pattern Recognition, Image Analysis and Applications. Lecture Notes in Computer Science*, 4225, 529-538.

[72] **Yáñez-Márquez, C., Cruz-Meza, M.E., Sánchez-Garfias, F.A., & López-Yáñez, I. (2007).** Using alpha-beta associative memories to learn and recall RGB images. *Advances in Neural Networks: 4th International Symposium on Neural Networks. Lecture Notes in Computer Science*, Nanjing, China, 4493, 828-833.

[73] **Godínez, I.R., López-Yáñez, I. & Yáñez-Márquez, C. (2009).** Classifying patterns in bioinformatics databases by using Alpha-Beta associative memories. *Biomedical Data and Applications, Studies in Computational Intelligence*, 224, 187-210.

[74] **López-Yáñez, I., Argüelles-Cruz, A.J., Camacho-Nieto, O. & Yáñez-Márquez, C. (2011).** Pollutants time-series prediction using the Gamma classifier. *International Journal of Computational Intelligence Systems*, 4(4), 680-711.

[75] **Aldape-Pérez, M., Yáñez-Márquez, C., Camacho-Nieto, O. & Argüelles-Cruz, A.J. (2012).** An associative memory approach to medical decision support systems. *Computer Methods and Programs in Biomedicine*, 106(3), 287-307.

[76] **Uriarte-Arcia, A.V., López-Yáñez, I. & Yáñez-Márquez, C. (2014).** One-hot vector hybrid associative classifier for medical data classification. *PLoS ONE*, 9(4), e95715.

[77] **Gama, J. (2010).** *Knowledge Discovery from Data Streams.* Chapman and Hall/CRC.

[78] **Gama, J. (2012).** A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1), 45-55.

[79] **Vitter, J.S. (1985).** Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1), 37-57.

[80] **Gilbert, A.C., Kotidis, Y., Muthukrishnan, S. & Strauss, M. (2001)** Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. *Proceedings of the 27th International Conference on Very Large Data Bases*, Rome, Italy, 79-88.

[81] **Datar, M., Gionis, A., Indyk, P. & Motwani, R.** Maintaining stream statistics over sliding windows. *Proceedings of Annual ACM SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics*, San Francisco, USA, 635-644.

[82] **Cormode, G. & Muthukrishnan, S.** An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1), 58-75.

[83] **Dasu, T., Krishnan, S., Venkatasubramanian, S. & Yi, K. (2006).** An information-theoretic approach to detecting changes in multi-dimensional data streams. *Proceedings of the Symposium on the Interface of Statistics, Computing Science, and Applications*.

[84] **Hotelling, H. (1931).** The Generalization of Students Ratio. *The Annals of Mathematical Statistics*, 2(3), 360-378.

[85] **Lazarescu, M.M. (2005).** A multi-resolution learning approach to tracking concept drift and recurrent concepts. *Proceedings of the 5th international workshop on pattern recognition in information systems*, Miami, USA, 52-62.

[86] **Li, P., Wu, X., & Hu, X. (2012).** Mining recurring concept drifts with limited labeled streaming data. *ACM Transactions on Intelligent Systems and Technology*, 3(2), 29:1-29:32.

[87] **Gama, J., Sebastião, R. & Pereira, P. (2013).** On evaluating stream learning algorithms. *Machine Learning*, 90(3), 317-346.

[88] **Gama, J. & Kosina, P. (2014).** Recurrent concepts in data streams classification. *Knowledge and Information Systems*, 40(3), 489-507.

[89] **Denis, F., Gilleron, R. & Letouzey, F. (2005).** Learning from positive and unlabeled examples. *Theoretical Computer Science*, 348(1), 70-83.

[90] **Duda, R.O., Hart, P.E. & Stork, D.G. (2001).** *Pattern Classification*. Second Edition. McGraw-Hill.

[91] **Elwell, R. & Polikar, R. (2011).** Incremental learning of concept drift in nonstationary eEnvironments. *IEEE Transactions on Neural Networks*, 22(10), 1517-1531.

[92] **Aha, D.W., Kible, D. & Albert, M.K. (1991).** Instance-based learning algorithms. *Machine Learning*, 6(1), 37-66.

[93] **Cover, T.M. & Hart, P.E. (1967).** Nearest neighbor pattern classification. *IEEE Transacions on Information Theory*, IT-13(1), 21-27.

[94] **Wilson, D.R. & Martinez, T.R. (1997).** Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6(1), 1-34.

[95] **Arthur D. & Vassilvitskii S. (2007).** K-means++: the advantages of careful seeding. *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, Louisiana, USA, 1027-1035.

[96] **Marques de Sá, J.P. (2001).** *Pattern Recognition: Concepts, Methods and Applications*. Springer.

[97] **Acevedo-Mosqueda, M.A., Yáñez-Márquez, C. & López-Yáñez, I. (2007).** AlphaBeta bidirectional associative memories: theory and applications. *Neural Processing Letters*, 26(1), 1-40.

[98] **López-Yáñez, I. (2007).** Clasificador automático de alto desempeño. Master Thesis in Computer Science. Instituto Politécnico Nacional, Centro de Investigación en Computación, México D.F., México.

[99] **Flores-Carapia, R. (2006).** Memorias asociativas Alfa-Beta basadas en el código Johnson-Möbius modificado. Master Thesis in Computer Science. Instituto Politécnico Nacional, Centro de Investigación en Computación, México D.F., México.

[100] **Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. & Bouchachia, A. (2014).** A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 44:1-44:37.

[101] **Cohen, E. & Strauss, M.J. (2006).** Maintaining time-decaying stream aggregates. *Journal of Algorithms*, 59(1), 19-36.

[102] **Bifet, A. & Gavaldà, R. (2006).** Kalman filters and adaptive windows for learning in data streams. *Discovery Science. Lecture Notes in Computer Science*, 4265, 29-40.

[103] **Last, M. (2002).** Online classification of nonstationary data streams. *Intelligent Data Analysis*, 6(2), 129-147.

[104] **Wang, H., Fan, W., Yu, P.S. & Han, J. (2003).** Mining concept-drifting data streams using ensemble classifiers. *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining*, Washington D.C., USA, 226-235.

[105] **Oza, N. & Russell, S. (2001).** Online bagging and boosting. *Proceedings of the 8th Internatinoal Workshop in Artificial Intelligence and Statistics*, Key West, Florida, USA,105-112.

[106] **Bifet, A., Kirkby, R., Kranen, P. & Reutemann, P. (2012).** Massive online analysis manual. Available for download from http://moa.cms.waikato.ac.nz/documentation/.

[107] **Massive Online Analysis (MOA).** Available for download from http://moa.cms.waikato.ac.nz.

[108] **Bache, K. & Lichman, M. (2013).** UCI machine learning repository, School of Information and Computer Science, University of California, Irvine, California, USA. Available for download from http://archive.ics.uci.edu/ml.

[109] **Harries, M. (1999).** Splice-2 comparative evaluation: electricity pricing. Technical report, The University of South Wales, 1999.

[110] **Agrawal, R., Imielinski, T. & Swami, A. (1993).** Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6), 914-925.

[111] **Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I.H. (2010).** WEKA 3: data mining software in java. Available for download from http://www.cs.waikato.ac.nz/ml/weka/.