INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAT017

Thèse de doctorat

Improving IoT Data Stream Analytics Using Summarization Techniques

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 Dénomination (Sigle)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 5 juin 2020, par

MAROUA BAHRI

Composition du Jury :

Albert Bifet Professor, Télécom Paris	Co-directeur de thèse
Silviu Maniu Associate Professor, Université Paris-Sud	Co-directeur de thèse
João Gama Professor, University of Porto	Président
Cédric Gouy-Pailler Engineer-Researcher, CEA-LIST	Examinateur
Ons Jelassi Researcher, Télécom Paris	Examinateur
Moamar Sayed-Mouchaweh Professor, Ecole Nationale Supérieure des Mines de Douai	Rapporteur
Mauro Sozio Associate Professor, Télécom Paris	Examinateur
Maguelonne Teisseire Professor, TETIS, INRAE	Rapporteur

Improving IoT Data Stream Analytics Using Summarization Techniques

Maroua Bahri

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

Télécom Paris
Institut Polytechnique de Paris

Supervisors:
Albert Bifet
Silviu Maniu

Examiners:
João Gama
Cédric Gouy-Pailler
Ons Jelassi
Moamar Sayed-Mouchaweh
Mauro Sozio
Maguelonne Teisseire

Paris, 2020

Abstract

With the evolution of technology, the use of smart Internet-of-Things (IoT) devices, sensors, and social networks result in an overwhelming volume of IoT data streams, generated daily from several applications, that can be transformed into valuable information through machine learning tasks. In practice, multiple critical issues arise in order to extract useful knowledge from these evolving data streams, mainly that the stream needs to be efficiently handled and processed. In this context, this thesis aims to improve the performance (in terms of memory and time) of existing data mining algorithms on streams. We focus on the classification task in the streaming framework. The task is challenging on streams, principally due to the high – and increasing – data dimensionality, in addition to the potentially infinite amount of data. The two aspects make the classification task harder.

The first part of the thesis surveys the current state-of-the-art of the classification and dimensionality reduction techniques as applied to the stream setting, by providing an updated view of the most recent works in this vibrant area.

In the second part, we detail our contributions to the field of classification in streams, by developing novel approaches based on summarization techniques aiming to reduce the computational resource of existing classifiers with no – or minor – loss of classification accuracy. To address high-dimensional data streams and make classifiers efficient, we incorporate an internal preprocessing step that consists in reducing the dimensionality of input data incrementally before feeding them to the learning stage. We present several approaches applied to several classifications tasks: Naive Bayes which is enhanced with sketches and hashing trick, k-NN by using compressed sensing and UMAP, and also integrate them in ensemble methods.

I dedicate to my sweet loving

Father, Mother & Siblings

whose unconditional love, encouragement, affection, and prays of day and night make me able to get such success and honor. Without them, I would never get through this.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ARF	**A**daptive **R**andom **F**orest
ADWIN	**AD**aptive **WIN**dowing
CS	**C**ompressed **S**ensing
CMS	**C**ount-**M**in **S**ketch
DR	**D**imensionality **R**eduction
HT	**H**ashing **T**rick
HAT	**H**oeffding **A**daptive **T**ree
IoT	**I**nternet of Things
JL	**J**ohnson-**L**indenstrauss
k**NN**	k-**N**earest **N**eighbors
NB	**N**aive **B**ayes
RP	**R**andom **P**rojection
UMAP	**U**niform **M**anifold **A**pproximation and **P**rojection

List of Symbols

S	Stream
X_i	Instance, observation or data point, i from S
x_i^j	Value of the attribute j of instance i
a	Number of input attributes
m	Number of output attributes after reduction
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^a$	Set of a-dimensional real vectors
$\mathbb{R}^m$	Set of m-dimensional real vectors
$\mathbb{R}^{m \times a}$	Set of $m \times a$ real matrices
C	Set of class labels
W	Sliding window
d	Sketch depth
w	Sketch width
k	Number of neighbors

Part I

Introduction and Background

Data stream learning is a hot research topic in Machine Learning and Data Mining, that has motivated the development of several very efficient algorithms in the streaming setting. Our thesis deals with the elaboration of new classification approaches based on well-known summarization techniques. These proposed approaches make it possible to learn from – and make predictions on – evolving data streams using small computational resources while keeping good classification performance.

This part presents the necessary background concerning our thesis and is composed of two chapters:

- Chapter 1 provides an overview of the context and motivation of the thesis, followed by our principal contributions.

- Chapter 2 gives the necessary background regarding the streaming framework, its challenges and limitations. We cover the definition of the basic concepts of the stream context, such as the processing and summarization techniques for data streams. In this chapter, we also review the state-of-the-art of streaming classification and dimensionality reduction that are relevant to this thesis.

1

Introduction

Contents

1.1 Context and Motivation

Artificial Intelligence (AI) is defined as field of study in *Computer Science* that aims to produce smart machines capable to mimic natural intelligence displayed by humans. The last few decades have witnessed a tremendous pace in the pervasiveness of technology that invades our world in all dimensions and keeps skyrocketing. This evolution includes, more and more, systems and applications that continuously generate vast amounts of data in an open-ended way as *streams*.

This incredibly huge quantity of data, derived daily from applications in AI, includes areas such as robotics, natural language processing, and sensor analytics [1]. As an instance application, the Internet of Things (IoT) is defined as a large network of physical devices and sensors (objects) that connect, interact, and exchange data. IoT is a key component of life automation, e.g., cars, drones, airplanes, and home automation. These devices will be creating a massive quantity of big data, via real-time streams, in the near future. By the end of 2020, 31 billion of such devices will be connected, and by 2025 this number is

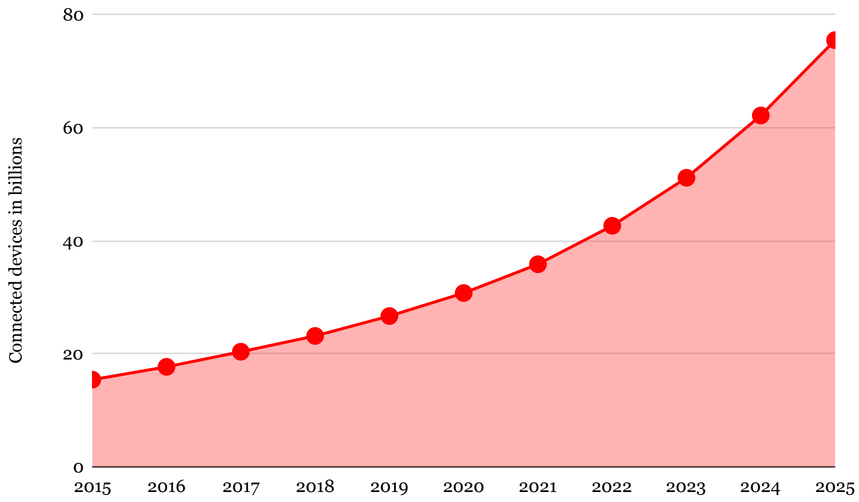


Figure 1.1: IoT connected devices from 2015 to 2025.

expected to grow, according to Statista[1], to around 75 billion of these devices that will be in use around the world, see Figure 1.1. Hence, methods and applications must be explored to cope with this tremendous flow of data that exhibit the so-called "3 *Vs*", *volume*, *velocity*, and *variability*.

There exist different ways to treat, organize and analyze this fast generated data using algorithms and tools from AI and data mining. Indeed, these techniques are often carried out by automated methods such as the ones from *machine learning*. The latter is a fundamental subset of AI that is based on the assumption that computer algorithms can learn from new data and automatically improve themselves without – or with minimal – external intervention (human intervention in general). Machine learning algorithms are characterized by learning from observations and then, make predictions using these observations in order to build appropriate models [2].

The success of IoT has motivated the field of data mining which consists in being able to extract useful knowledge by automatically acquiring the hidden insights, non-trivial, in the vast and growing sea of data made available along time. Data mining is a sub-field of machine learning which includes tasks such as classification, regression, and clustering that have been thoroughly studied over the last decades. However, traditional approaches for static datasets have some limitations when applied on dynamic data streams, hence, new approaches with novel mining techniques are necessary. In this context of IoT, mining algorithms should be able to handle the infinite and high-velocity of IoT data streams, under finite resources – in terms of time and space. More details on these challenges are provided in Section 1.2.

[1] www.statista.com/statistics/976079/number-of-iot-connected-objects-worldwide-by-type/.

To convert these data into useful knowledge, we usually use *machine learning* algorithms adapted for streaming data tasks, consisting of a data analysis process [3, 4]. In this context, the data stream mining area has become indispensable and ubiquitous in many real-world applications. Often this category of applications generates data from evolving distributions and requires real-time – or near real-time – processing.

In the data mining field, ***classification*** is one of the most popular tasks that attempts to predict the category of unlabeled – and unseen – observations by building a model based on the attribute contents of the available data. The stream classification task is considered as an active area of research in the data stream mining field, where the focus is to develop new – or improve existing – algorithms [5]. There is a number of *classifiers* that are widely used in data mining and applied in several real-world applications, such as decision trees, neural networks, k-nearest neighbors, Bayesian networks, etc. The next chapter covers, *inter alia*, a survey on the well-known and recent classification algorithms for evolving data streams.

1.2 Challenges

As mentioned above, stream classification task aims to predict *labels* – or *classes* – of new incoming unlabeled instances from the stream while updating continuously, after prediction, the models as the stream evolves to follow the current distribution of the data. The online and potentially infinite nature of data streams, which raises some critical issues and makes traditional mining algorithms fail, imposes high resource requirements to handle the dynamic behavior of evolving distributions.

While many of the following issues are common across different data stream mining applications, we address these issues in the context of *incremental classification* [6, 7, 8].

- **Evolving nature of data streams.** Any classification algorithm has to take into account the considerable evolution of data and adapt to the high speed nature, because streams often deliver observations very rapidly. Thus, algorithms must incrementally classify recent instances.

- **Processing time.** A real-time algorithm should process the incoming instances as fast as possible because the slower it is the less efficient it will be for applications that require rapid processing.

- **Unbounded memory.** Due to the huge amounts and high speed of streaming data that require an unlimited memory, any classification algorithm should have the ability to work within memory constraint by maintaining as little as possible information about processed instances and the current model(s).

- **High-dimensional data streams.** Data streams may be high-dimensional, such as those containing text documents. For such kinds of data, distances between instances grow very fast which can potentially impact any classifier's performance.

Table 1.1: Comparison between static and stream data.

	Static data	Stream data
Access	random	sequential
Number of passes	multiple pass	single pass
Processing time	unbounded	restricted
Memory usage	unbounded	restricted
Type of result	accurate	approximate
Environment	static	dynamic (evolving)

- **Concept drift.** One crucial issue when dealing with a very large stream is the fact that the underlying distribution of the data can change at any moment, a phenomenon known as *concept drift*. We direct the reader to [9] for a survey of this concept. The drift can change the classifier results over time. To cope with the new trends of data that must be detected at the same time as their appearance, a drift detection mechanism is usually coupled with learning algorithms.

In the stream setting, a crucial question arises about how to process infinite data over time while addressing the stream framework requirements with *minimal costs*?

These above-mentioned challenges are of special significance in the stream classification. We notice that stream mining techniques must differ from the traditional ones for static datasets. To handle these challenges, classification algorithms must incorporate an incremental strategy that permits such processing requirements presented in Section 2.2. Table 1.1 presents a comparison of environments for both static and stream data [10].

In addition to the overwhelming volume of data, its *dimensionality* is increasing considerably and poses a critical challenge in many domains, such as biology (omics data[2]) [11, 12] and spam email filtering [13] (classify an email as spam or not, based on the email text content). These high-dimensional data may contain many redundant or irrelevant features that can be reduced to a smaller set of relevant combinations extracted from the input feature set without a significant loss of information. In order to handle such kind of data adequately at least cost possible, a pre-processing step is imperative to filter relevant features and therefore allow cost and resource savings with data stream mining algorithms. To do so, synopsis or statistics can be constructed from instances in the stream using ***summarization*** techniques (e.g., *sketches* by keeping frequencies of data), selecting a part of incoming data without reducing the number of features (i.e., sampling), or by applying dimensionality reduction (DR) to reduce the number of features. Naturally, the choice of a suitable technique depends on the problem being solved [14].

[2] Omics data refer to the data from biological fields ending by -omics, e.g., genomics, metabolomics.

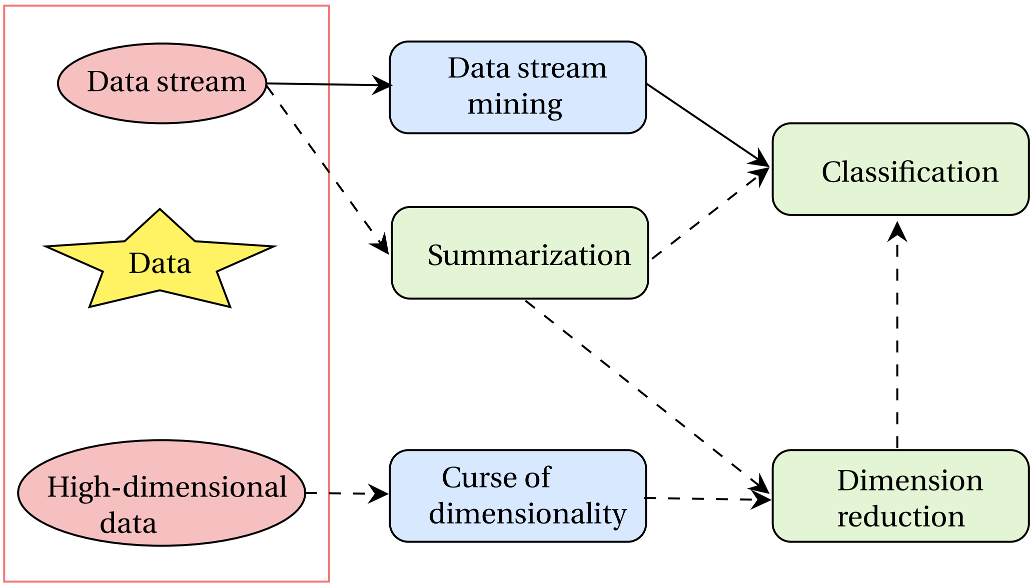


Figure 1.2: The thesis context. A data stream mining task can be applied on infinite data streams, a very popular task is classification. In order to alleviate the resource cost of a given classifier, a summarization technique is sometimes used to keep synopsis information about the stream for learning. Moreover, to overcome the curse of dimensionality, a dimensionality reduction technique can be applied on high-dimensional data as an internal pre-processing technique; then, the low-dimensional representation is fed to a classification task.

Our thesis purpose is motivated by the desired criteria, described above, for IoT data stream mining. We focus mainly on the classification task and aim to develop novel stream classification approaches to improve the performance of existing algorithms using summarization techniques. Figure 1.2 illustrates the context of this thesis.

Dimensionality reduction, embedding, and manifold learning are names for tasks that are similar in spirit. DR is defined as the projection of high-dimensional data into a low-dimensional space by reducing the input features to the most relevant ones. Indeed, DR is crucial to avoid the *curse of dimensionality* – which may increase the use of computational resources and negatively affect the predictive performance of any mining algorithm. To mitigate this drawback, several reduction techniques have been proposed, and widely investigated, in the offline setting [15, 16] to handle high-dimensional data. However, these techniques do not adhere to the strict computational resources requirements of the data stream learning framework [17, 18]. More details are provided in the next chapter.

1.3 Contributions

The main research line of this thesis addresses the aforementioned issues about mining algorithms' performance for IoT data streams. This thesis contributes to the stream mining field by introducing and exploring novel approaches that reduce the computational

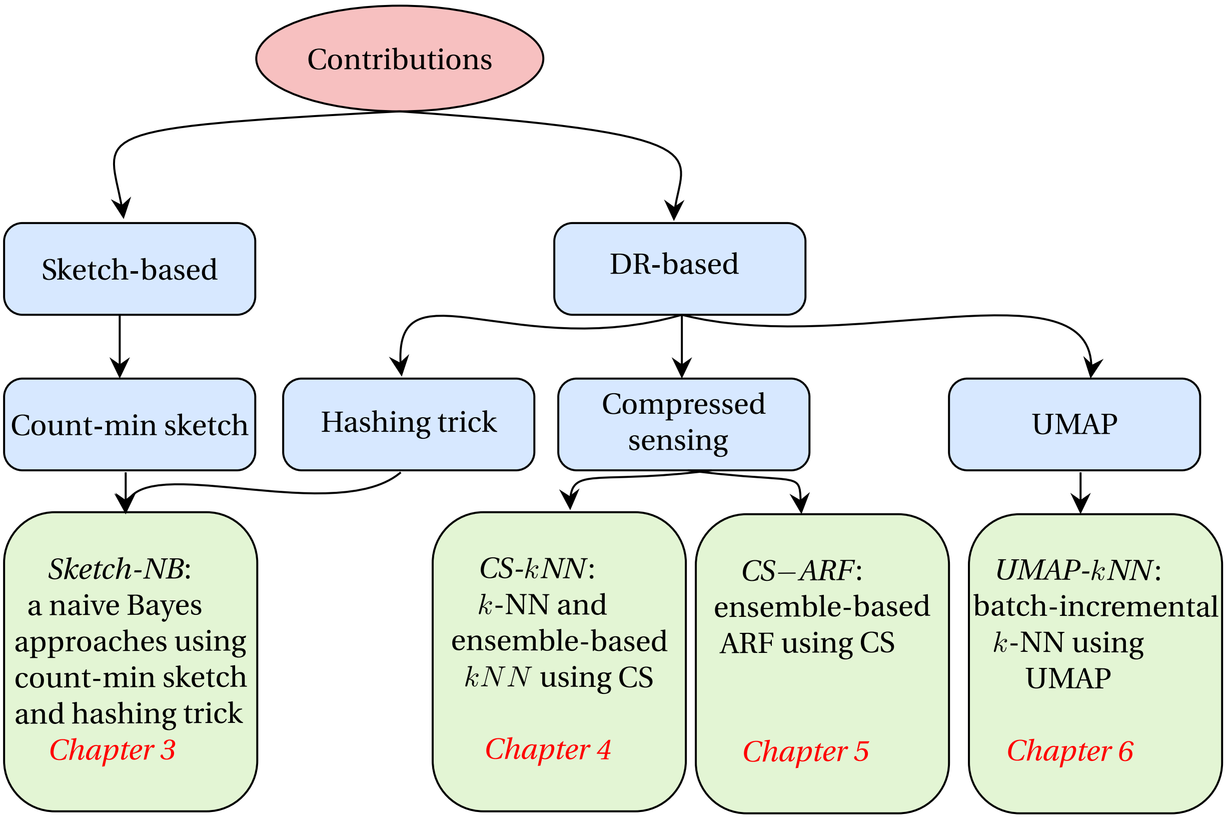

Contributions
Sketch-based
DR-based
Count-min sketch
Hashing trick
Compressed sensing
UMAP
Sketch-NB: a naive Bayes approaches using count-min sketch and hashing trick
Chapter 3
CS-kNN: k-NN and ensemble-based kNN using CS
Chapter 4
CS−ARF: ensemble-based ARF using CS
Chapter 5
UMAP-kNN: batch-incremental k-NN using UMAP
Chapter 6

Figure 1.3: Contributions of the thesis.

resources of existing algorithms while sacrificing a minimal amount of accuracy.

During this introductory, we divide the objective of the thesis in three main research questions, enumerated in the following:

- **Q1**: *How can we improve the performance of the existing classifiers in terms of computational resources while maintaining good accuracy?*
- **Q2**: *How can we do better by dynamically adapting to high-dimensional data streams?*
- **Q3**: *How can we address concept drifts, i.e., the fact that stream distribution might change over time?*

In the following, we briefly sum up our contributions and schematize them in Figure 1.3.

- In **Chapter 3**, we aim to improve the performance of naive Bayes by developing three novel approaches to make it efficient and effective with high-dimensional data.

- We study an efficient data structure, called *Count-Min Sketch (CMS)* [19], to keep synopsis (frequencies) of data in memory.
- We propose a new sketch-based naive Bayes that uses CMS to store information about the infinite amount of data streams in fixed memory size for the naive Bayes learner.
- Theoretical guarantees over the size of the CMS table are provided by extending the guarantees of the CMS technique.
- To handle high-dimensional data, we add an online pre-processing step during which data will be compressed using a rapid DR technique, such as the *hashing trick* [20], before the learning tasks. This pre-processing task makes the approach faster.
- We incorporate in the learning phase an adaptive strategy using ADaptive WINdowing (ADWIN) [21], a change detector, for the entire sketch table, in order to track drifts (change in the distribution of data over time).

- In **Chapter 4**, we focus on the k-Nearest Neighbors (kNN) algorithm [22]. We propose two approaches that aim to improve the computational costs of the kNN algorithm when dealing with high-dimensional data streams by exploring the *Compressed Sensing (CS)* [23] technique to reduce the space size.
  - We propose a new kNN algorithm to support evolving data stream classification, compressed-kNN. Our main focus consists of improving kNN resource performance by compressing input streams using the CS before applying the classification task. This will result in a huge reduction in the computational cost of the standard kNN.
  - We provide theoretical guarantees over the neighborhood preservation before and after projection using the CS technique. We therefore ensure that the result of the classification accuracy to measure the performance of compressed-kNN is almost the same as the one that would be obtained with kNN using the high-dimensional input data.
  - We also provide an ensemble technique based on Leveraging Bagging [24] where we combine several compressed-kNN results to enhance the accuracy of a single classifier.

- In **Chapter 5**, we aim to improve the performance of the new reputed ensemble-based method, Adaptive Random Forest (ARF) [25] with high-dimensional data.
  - We propose a novel ensemble approach to support high-dimensional data stream classification which aims to enhance the resource usage of the ARF ensemble method by reducing the dimensionality of the input data using the CS technique internally which are afterward fed to the ensemble members.

- In **Chapter 6**, we explore a new DR technique that has attracted a lot of attention recently: Uniform Manifold Approximation and Projection (UMAP) [26]. We use this technique to pre-process the data, leveraging the fact that it preserves the neighborhood, to improve the results of the neighborhood-based algorithm, kNN.
  - We propose an adaptation of UMAP for evolving data streams, i.e., a batch-incremental manifold learning technique. Instead of applying this batch method, UMAP, on a static dataset at once, we adapt it by using mini-batches from the stream incrementally.
  - We propose a new batch-incremental kNN algorithm for stream classification using UMAP. The core idea is to apply the kNN algorithm on mini-batches of low-dimensional data obtained from the pre-processing DR step.

1.4 Publications

Some of the research findings presented in this thesis have been presented at international conferences. In the following we provide the list of currently accepted publications:

- Maroua Bahri, Silviu Maniu, Albert Bifet. “Sketch-Based Naive Bayes Algorithms for Evolving Data Streams”. *In the IEEE International Conference on Big Data (Big Data)*, 2018, Seattle, WA, USA.

- Maroua Bahri, Albert Bifet, Silviu Maniu, Rodrigo Fernandes de Mello, Nikolaos Tziortziotis. “Compressed k-Nearest Neighbors Ensembles for Evolving Data Streams”. *In the 24th European Conference on Artificial Intelligence (ECAI)*, 2020, Santiago de Compostela, Spain.

- Maroua Bahri, Bernhard Pfahringer, Albert Bifet, Silviu Maniu. “Efficient Batch-Incremental Classification for Evolving Data Streams”. *In the Symposium on Intelligent Data Analysis (IDA)*, 2020, Lake Constance, Germany.

- Maroua Bahri, Heitor Murilo Gomes, Albert Bifet, Silviu Maniu. “CS-ARF: Compressed Adaptive Random Forests for Evolving Data Stream Classification”. *In the International Joint Conference on Neural Networks (IJCNN)*, 2020, Glasgow, UK.

- Maroua Bahri, Heitor Murilo Gomes, Albert Bifet, Silviu Maniu. “Survey on Feature Transformation Techniques for Data Streams”. *In the International Joint Conference on Artificial Intelligence (IJCAI)*, 2020, Yokohama, Japan.

- Maroua Bahri, João Gama, Albert Bifet, Silviu Maniu, Heitor Murilo Gomes. “Data Stream Analysis: Foundations, Progress in Classification and Tools”. ***under review***.

1.5 Outline

The thesis is structured in the following parts:

- The first part includes Chapter 1 – the current chapter – and Chapter 2. In this chapter, we introduced the motivation of the thesis subject followed by the main goals and contributions. In Chapter 2, we introduce the fundamental concepts related to the stream setting. We define data streams and cover the challenges imposed by their infinite and online nature. We present the methodologies used to process these data and survey the state-of-the-art of classification and summarization (mainly dimensionality reduction) techniques that handle evolving data streams.

- The second part covers the main body of the thesis which includes our main contributions. It is divided in four chapters. The order of the chapters does not necessarily follow the chronology of the research thesis itself. Chapter 3 explores the naive Bayes algorithm, count-min sketch technique, and the hashing trick. We propose three algorithms that handle efficiently high-dimensional streams and detect drifts. In Chapter 4, we focus on enhancing the computational resources of the kNN algorithm – which is very costly in practice. We present an approach that uses the compressed sensing, CS-kNN, to pre-process incrementally the stream before the classification task. We proved that the neighborhood is preserved after the projection, i.e., the accuracy is not going to be greatly impacted. We therefore used this approach as a base learner inside the Leveraging Bagging ensemble, in order to improve accuracy of the single CS-kNN. Following the same direction, Chapter 5 introduces a recent ensemble-based method, ARF, that induces diversity to the ensemble members (that are different from each other) by using different random projection matrices, one for each ensemble member. In Chapter 6, we explore a recent dimensionality reduction technique, UMAP. Motivated by its high-quality results, we extend UMAP to the stream setting by adapting a batch-incremental strategy with the kNN algorithm to obtain the UMAP-kNN approach.

- In the third part, composed of Chapter 7, we give a summary of the results achieved in this thesis, and discuss possible future developments.

- Finally, Appendix A covers the open source frameworks that have been used to develop and test the aforementioned contributions.

2

Stream Setting: Challenges, Mining and Summarization Techniques

Contents

Parts of this chapter have been the subject of the two following survey papers in separate collaborations with *Heitor Murilo Gomes*[1] and *João Gama*[2]:

- M. Bahri, A. Bifet, J. Gama, S. Maniu, H.M. Gomes. "*Data Stream Analysis: Foundations, Progress in Classification and Tools*" [27].

- M. Bahri, A. Bifet, S. Maniu, H.M. Gomes. "*Survey on Feature Transformation Techniques for Data Streams*" [28] accepted to the International Joint Conference on Artificial Intelligence (IJCAI) 2020.

2.1 Introduction

In this chapter, we provide a general overview of the background of this thesis, the data streaming setting. The problems addressed in this thesis also belong to the *supervised learning* field; in short, we are studying *data stream classification*. Handling the challenges of the stream setting may require sophisticated algorithms composed of multiple components, such as a pre-processing task to reduce the dimensionality of input data, a drift detection mechanism for concept drifts.

Other than the standard constraints of data streams, the need of more computational resources to address further requirements arises when we deal with high-dimensional data. Classification algorithms must be coupled with summarization techniques to be effective.

This chapter is organized as follows: In Section 2.2, we define the data stream setting and outline the main limitations and challenges in this area. Then, we spotlight some fundamental approaches used to keep the scalability of the stream methods, needed in order to handle continuous and potentially infinite streams. Section 2.3 is dedicated to the state-of-the-art in stream classification, by providing an overview of some well-known and new classification methods. In Section 2.4, we survey the dimensionality reduction

[1] University of Waikato, Hamilton, New Zealand.
[2] University of Porto, Porto, Portugal.

approaches designed to handle high-dimensional streaming data. We then highlight the key benefits of using these approaches for data stream classification algorithms.

2.2 Preliminaries

In the era of IoT, applications in different domains have seen an explosion of information generated from heterogeneous stream data sources every day. Hence, data stream mining has become indispensable in many real-world applications, e.g., social networks, weather forecast, network monitoring, spam emails filtering, call records, and more. As mentioned in the previous chapter, static and streaming data are different because the dynamic and changing environment of data streams makes them impossible to store or to scan multiple times due to their tremendous volume [17].

Definition 2.1 *(Data streams). Stream, incremental or online, data S, are defined as an unbounded sequence of multidimensional, sporadic, and transient observations (also called instances) made available along time. In the following, we assume that $S = X_1, \ldots, X_N, \ldots$, where each instance X_i is a vector that contains a attributes or features, denoted by $X_i = (x_i^1, \ldots, x_i^a)$ and N denotes the number of instances encountered thus far in the stream.*

In Section 1.2, we presented the main research issues encountered in the streaming framework and in this section we present some well-known manners to deal with such constraints.

2.2.1 Processing

To cope with these requirements, we can use well-established methods such as processing in one-pass and summarization (e.g., sampling) [6, 7, 29]. We describe them below.

- **One-pass constraint.** With the increasing nature of the data, it is no longer possible to examine a stream of data efficiently by using multiple passes because of its huge size and the inability to examine it more than once during the course of computation. Considering this issue, results are obtained by scanning the data stream only once and update the classification model incrementally or with the assumption that data arrive in chunks (e.g., batch-incremental algorithms).

- **Window models.** Classification results can change over time with the fact that the data change accordingly. Since scanning the stream multiple times is not allowed, therefore, the so-called moving window techniques have been proposed to capture important contents of the evolving stream. There exists three kinds of windows which are the following [30]:

  – ***Sliding window model***: Whose size is fixed to keep the last incoming observations, i.e., only the most recent observations from the data stream are stored inside the window. As time changes, the sliding window moves over the stream while keeping the same size. The set of the last elements (t) is considered as the most recent one (see Figure 2.1).

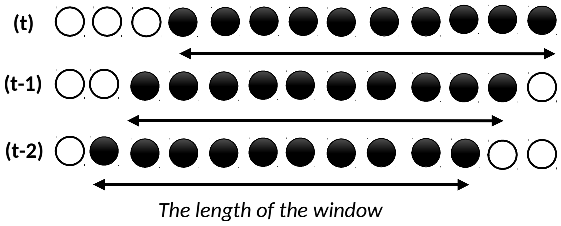


Figure 2.1: Sliding window model.

- ***Landmark window model***: In this model, the size of the window increases with time starting from a predefined instance, called *landmark*. When a new landmark is reached, all instance are removed and instances from the current landmark are kept (Figure 2.2). One issue arises in a special case when the landmark is fixed from the beginning, so the window will contain the entire stream.

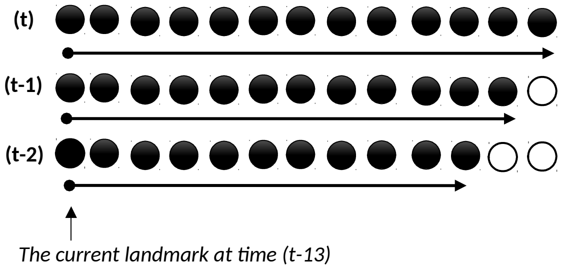


Figure 2.2: Landmark window of size 13.

- ***Damped window model***: This model is based on a fading function that periodically modifies the weights of instances. The key idea consists in assigning a weight to each instance from the stream, which is inversely proportional to its age, i.e., assign more weights to the recent arrived data. When the weight of an instance exceeds a given threshold, it will be removed from the model (see Figure 2.3).

Table 2.1: Window models comparison.

Window Model	Definition	Advantages	Disadvantages
Sliding	process the last received instances	suitable when the interest exists only in the recent instances	ignoring part of the stream
Landmark	process the entire history of the stream	suitable for one-pass classification algorithms	all the data are equally important
Damped (Fading)	assign weights to instances	suitable when the old data may affect the results	unbounded time window

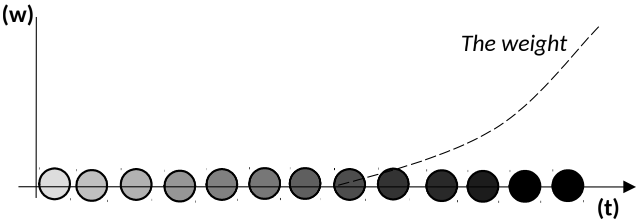


Figure 2.3: Damped window model.

Table 2.1 shows a brief description of the window models with their advantages and drawbacks. Different classification methods can be adapted to use the above models; the choice of the window model depends on the application needs [30].

The infinite nature of data streams makes them impossible to store due to resource constraints. In this context, how can we keep track of instances seen so far with *minimal information loss*?

2.2.2 Summarization

Instead of – or in conjunction with – the previous mentioned techniques, another approach is to keep only a synopsis of summary of the information constructed from stream instances. This can be achieved by either keeping a small part of the incoming or by constructing other data structures storing a synopsis of the data. In what follows, we briefly present some techniques.

- **Sampling.** Sampling methods are the most simple ones for synopsis construction in the stream framework. Storing static datasets is simple enough. In contrast, when dealing with large data streams, this is an impossible task. In this context, it is intuitively reasonable to sample the stream in order to keep some “representative” instances and thus decrease the stream size that will be stored in memory [31]. This method is easy conceptually and efficient to implement; however, the samples can be biased and not too representative.

- **Histograms.** Histograms are widely used with static datasets but their extensions to the stream framework is still a challenging task. Some techniques [32] have proposed histograms for the incremental setting to handle evolving streams. However, they do not always work because the distribution of the instances is assumed to be uniform which is not always true.
- **Sketching.** Sketch-based methods are well-known for keeping small, but approximate, synopses of data [33, 31]. They build a summary of data stream using a small amount of memory. Among them, we cite the count-min sketch [19] which is a generalization of *bloom filters* [34] used for counting items of a given item, using approximate counts while keeping sound theoretical bounds on the counts.
- **Dimensionality reduction.** Dimensionality reduction (DR) is a very popular tool to tackle high-dimensional data, another factor that makes classification algorithms expensive. It is defined as the transformation that maps instances from a high-dimensional space onto a lower-dimensional one while preserving the distances between instances [16]. Thus, instead of applying the classification algorithm on the high-dimensional data, we apply it on their small representation, in the low-dimensional space.

2.3 Stream Supervised Learning

One of the most important tasks in data mining is *classification* [5].

Definition 2.2 *(Classification problem). The problem of classification, a supervised learning task, consists in one attempts to predict a class label,* (y')*, of some unlabeled data instance* (X') *composed of a vector of attributes, by applying a generated model* $\mathbb{M}$ *(trained on labeled data* (X, y)*) [35].*

$$y' = \mathbb{M}(X'). \tag{2.1}$$

Traditionally, data mining has been performed over static datasets in the offline setting where, for the classification task, the training process is applied on a fixed size dataset and afford to read the input data several times. However, the dynamic and open-ended nature of data streams has outpaced the capability of traditional classifiers, also called *batch* classifiers, to be loaded into memory due to the technical requirements of the incremental environment [36]. The difference between classifiers for statics datasets (batch classifiers) and data stream classifiers resides in the way of how the learning and prediction are performed. Unlike batch learning, online learning must deal with data incrementally and

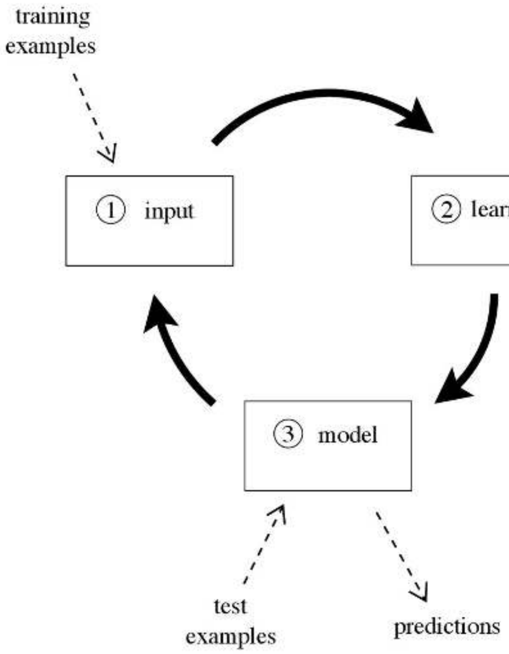


Figure 2.4: The data stream classification cycle [36].

use the one-pass processing. Moreover, and most importantly, they must use a limited amount of time to allow analysis of each instance without delay, and a limited amount of space to avoid storing huge amounts of data for the prediction task. Figure 2.4 illustrates the general model of the classification process within a data stream environment taking into account the requirements outlined above [35]:

1. The classifier receives the next training instance available from the stream (one-pass constraint).
2. The classifier processes the instance and updates the current model quickly using only a limited amount of memory.
3. The classifier predicts, in a first attempt, the class label of unlabeled instances and use them later to update the model.

A multitude of classification algorithms for static datasets that have been widely studied in the offline processing, and proved to be of limited effectiveness when dealing with evolving data streams, have been extended to work within a streaming framework [37]. A general taxonomy divides classification algorithms into four main categories 2.5: (i) frequency-based; (ii) neighborhood-based; (iii) tree-based; and (iv) ensemble-based classification algorithms.

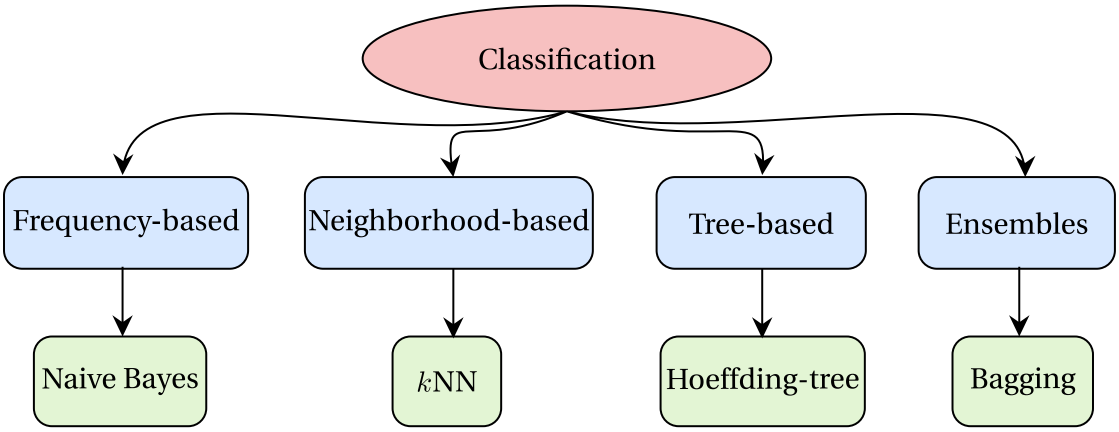


Figure 2.5: Taxonomy of classification algorithms.

2.3.1 Frequency-Based Classification

One of the most simple classifiers is Naive Bayes (NB) [38]. It uses the assumption that the attributes are all independent of each other and w.r.t. the class label, uses Bayes's theorem to compute the posterior probability of a class given the evidence (the training data). This assumption is obviously not always true in practice, yet the NB classifier has a surprisingly strong performance in real-world scenarios. Naive Bayes is a special algorithm that needs no adaptation to the stream setting because it naturally trains data incrementally thanks to its simple and easy frequency-based strategy.

2.3.2 Neighborhood-Based Classification

k-Nearest Neighbors (kNN) is a neighborhood-based algorithm that has been adapted to the data stream setting. It does not require any work during training but it uses the entire dataset to predict the class labels for test examples. The challenge with adapting kNN to the stream setting is that it is not possible to store the entire stream for the prediction phase. An envisaged solution to solve this issue is to manage the examples that are remembered so that they fit into limited memory and to merge new examples with the closest ones already in memory. Yet, searching for the nearest neighbors still costly in terms of time and memory [39]. Another new kNN method that has been proposed recently in the stream framework is Self-Adjusting Memory kNN (sam$k$NN) [40]. Sam$k$NN uses a dual-memory model to capture drifts in data streams by building an ensemble with models targeting current or former concepts.

2.3.3 Tree-Based Classification

Several tree-based algorithms have been proposed to handle evolving data streams [41, 42, 43]. A well-known decision tree learner for the data streams is the Hoeffding tree algorithm [41], also known as Very Fast Decision Tree (VFDT). It is an incremental, anytime decision tree induction algorithm that uses the Hoeffding bound to select the optimal splitting attributes. However, this learner assumes that the distribution generating instances does not change over time. So, to cope with an evolving data stream, a drift detection algorithm is usually coupled with it. In [44] an adaptive algorithm was proposed, Hoeffding Adaptive Tree (HAT), extending the VFDT to deal with concept drifts. It uses the ADaptive WINdowing (ADWIN) [21], a change detector and estimator, to monitor the performance of branches on the tree and to replace them with new branches when their accuracy decreases if the new branches are more accurate. These algorithms require more memory as the tree expands and grows; they also sacrifice computational speed due to the time spent in choosing the optimal attribute to split.

2.3.4 Ensembles-Based Classification

Ensemble learning is receiving increased attention for data stream learning due to its potential to greatly improve learning performance [25, 45]. Ensemble-based methods, used for classification tasks, can easily be adapted to the stream setting because of their high learning performance and their flexibility to be integrated with drift detection strategies. Unlike single classifiers, ensemble-based methods predict by combining the predictions of several classifiers. Several empirical and theoretical studies have shown the reasoning that combining multiple "weak" individual classifiers leads to better predictive performance than a single classifier [44, 46, 47], as illustrated in Figure 2.6. Ensembles have several advantages over single classifier methods, such as: (i) *robustness*: they are easy to scale and parallelize; (ii) *concept drift*: they can adapt to change quickly by resetting or updating current under-performing model of the ensemble; and (iii) *high-predictive performance*: they therefore usually generate more accurate concept descriptions.

An extensive review about the related work is provided in [48], and we present briefly here the well-known and the recent ones. *Online Bagging* [49] is a streaming version of Bagging [46] which generates k models trained on different samples. Different from batch Bagging where samples are produced with replacement, online Bagging selects weighted samples sampled from a Poisson(1) distribution. Leveraging Bagging (LB) [24] is based on the online Bagging method. In order to increase the accuracy, LB handles drifts using ADWIN [21], where if a change is detected, the worst classifier is erased and a new one is added to the ensemble. LB also induces more diversity to the ensemble via randomization. Adaptive Random Forests (ARF) [25] is a recent extension of Random Forest method to handle evolving data streams. ARF uses Hoeffding tree as a base learner where attributes are randomly selected during training. It is coupled with a drift detection scheme using ADWIN on each ensemble member where we replace a tree, once a drift is detected, by an alternate tree trained on the new concept. Streaming Random Patches (SRP) [50] is also a novel method that combines random subspaces and bagging while using a strategy to detect drifts similar to the one introduced in [25].

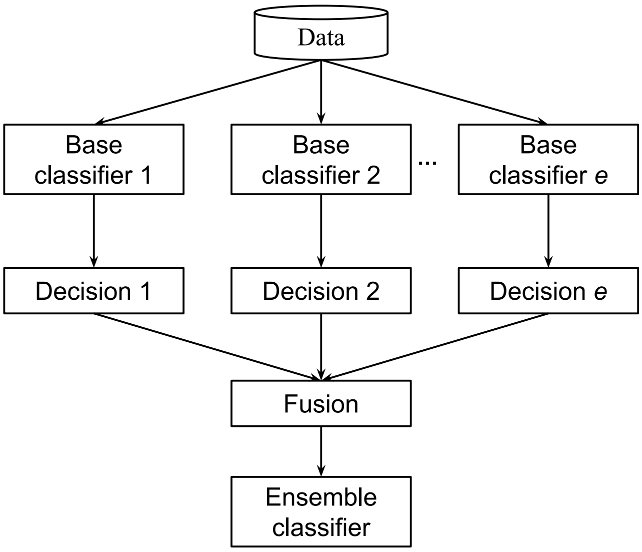


Figure 2.6: Ensemble classifier.

One notable issue related to the ensemble-based methods with evolving data streams is the massive computational demand (in terms of memory usage and running time). Ensembles require more resources than single classifiers which become significantly worse with high-dimensional data streams.

Ensemble-based methods pose a resources-accuracy tradeoff, they produce better predictive performance than single classifiers, indeed, at the price of being more costly in terms of computational resources.

2.4 Dimensionality Reduction

Definition 2.3 *(Dimensionality reduction). Given an instance X_i which is composed of a vector of a attributes $X_i = x_i^1, \ldots, x_i^a$. The DR comprises the process of finding some transformation function (or* map*) $A : \mathbb{R}^a \rightarrow \mathbb{R}^m$, where $m \ll a$, to be applied on each instance X_i from the stream S as follows:*

$$Y_i = A(X_i), \tag{2.2}$$

where $Y_i = y_i^1, \ldots, y_i^m$.

We distinguish two main different dimensionality reduction categories: (i) *feature selection* which consists in selecting a subset of the input features, i.e., the most relevant and non-redundant features, without operating any sort of data transformation [51]; and (ii) *feature transformation* – also called *feature extraction* – which consists in constructing from a set of input features in high-dimensional space, a new set of features in a lower

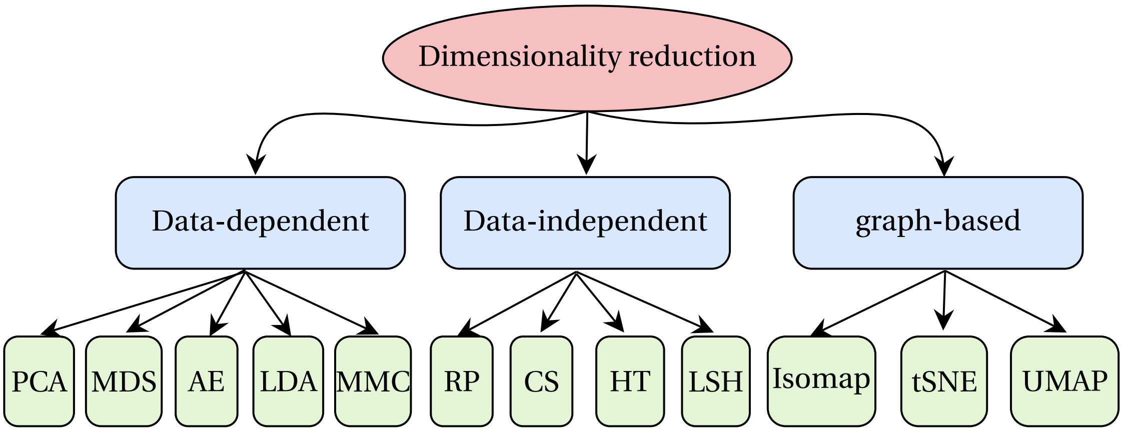


Figure 2.7: Taxonomy of dimensionality reduction techniques.

dimensional space [52].

Feature transformation and feature selection have different processing requirements. Despite the fact that both reductions are used to minimize an input feature space size, feature transformation is a DR that creates a new subset – or combinations – of features by exploiting the redundancy and noise of the input set of features, whereas feature selection is characterized by keeping the most relevant attributes from the original set of features present in the data without changing them.

Some recent surveys on streaming feature selection have been proposed [53, 54, 55, 56]. The major weakness of this category is that it could lead to a data loss. The latter may happen when, unintentionally, we remove features that might be useful for a later task (e.g., classification, visualization). In the following, we review the most crucial feature transformation techniques that are – or can be – used in the stream framework and discuss their similarities and differences. In what follows, we refer to feature transformation as dimensionality reduction.

A Taxonomy of Dimensionality Reduction. In the following, we introduce DR techniques that have been widely used in machine learning algorithms. These techniques operate by transforming and using the most relevant feature combinations, in turn reducing space and time demands; this can be crucial for applications such as classification and visualization. Traditionally, many techniques have been proposed and thoroughly used in the offline framework for static datasets, but these techniques cannot be used in the streaming framework, due to the requirements imposed by the latter (e.g., one-pass processing). Figure 2.7 shows a taxonomy that subdivides the DR techniques as follows, *data-dependent*, *data-independent*, and *graph-based* transformation techniques. The data-dependent techniques are derived from the whole data to achieve the transformation, whereas the data-independent techniques are based on random projections and do not use the input data to perform the projection. Graph-based techniques are also data-dependent that build a neighborhood graph to maintain the data structure (i.e., preserves the neighborhood after projection).

2.4.1 Data-Dependent Techniques

Data-dependent techniques construct a projection function – or matrix – from the data. This requires the presence of the entirety – or at least a part of – the dataset. In the streaming context, where data are potentially infinite, the classical techniques from this category are therefore limited, since keeping the entire data stream in memory is impractical.

Principal Components Analysis

Principal Components Analysis (PCA) is the most popular and straightforward unsupervised technique that seeks to reduce the space dimension by finding a lower-dimensional basis in which the sum of squared distances between the original data and their projections is minimized, i.e. being as close as possible to zero while maximizing the variances between the first components. Mathematically, PCA aims to find a linear mapping formed by a few orthogonal linear combinations, also called eigenvectors or PCs, from the original data that maximizes a certain cost function. However, PCA computes eigenvectors and eigenvalues from a computed covariance matrix, relying on the whole dataset. This is ineffective for streaming data since a re-estimation of the covariance matrix from scratch for new observations is unavoidable.

In this context different variations of component analysis have been adapted to the stream setting. For instance, Incremental PCA (IPCA) [57] focuses on how to update the eigenvectors of images (eigenimages) based on the previous coefficients. Candid Covariance-free Incremental PCA (CCIPCA) [58] is another extension that updates the eigenvectors incrementally and does not need to compute the covariance matrix for each new incoming instance (images) which makes it very fast. The main difference among these techniques arises in how the eigenvectors are updated. On the other hand, the common limitation concerns their application domain since both techniques deal with images as high-dimensional vectors and have not been tested on different types of data [57, 58]. Ross, Lim, Lin, and Yang proposed a batch-incremental PCA that deals with a set of new instances each time a batch is complete. However, this approach is not suited for instance-incremental learning (i.e., processing instances one by one incrementally). Mitliagkas, Caramanis, and Jain proposed a memory-limited streaming PCA that attempts to make vanilla PCA incremental and computation-efficient with high-dimensional data where samples are drawn from a Gaussian spiked covariance model. A more recent work [61] proposes a single-pass randomized PCA technique that iteratively update the subspace's orthonormal basis matrix within an accuracy-performance tradeoff. Yu, Gu, Li, Liu, and Li claim that this technique works well in many applications, albeit it has been evaluated on a single image dataset.

The above PCA techniques apply to data stream mining algorithms to alleviate their computation costs. For instance, Feng, Yan, Ai-ping, and Quan-yuan proposed an efficient online classification algorithm, FIKOCFrame, that uses a PCA variant, fast iterative kernel PCA [63], to incrementally reduce the dimensionality before classification.

Cardot and Degras proposed recently a comparative review of the incremental PCA approaches where they provide guidance for selecting the appropriate approach based on their accuracy and computation resources (time and memory).

Multi-Dimensional Scaling

Multi-Dimensional Scaling (MDS) [65] is a well-known unsupervised technique used for embedding. It projects a given distance matrix into a non-linear lower-dimensional space while preserving the similarity among instances. Nevertheless, this technique is computationally expensive with large datasets and non-scalable because it requires the entire data distance matrix. To cope with this issue, some incremental versions have been proposed to alleviate the computational requirements.

Incremental MDS (iMDS) technique, proposed by Agarwal, Phillips, Daumé III, and Venkatasubramanian, keeps some distance preservation using the so-called *out of sample* mapping without the need of reconstructing the whole matrix. A more recent work by Zhang, Huang, Mueller, and Yoo proposed a new version of MDS for high-dimensional data, named *sc*MDS. It is a batch-incremental technique where authors introduced a realignment matrix for each batch to overcome the concept drift that may occur because each batch may have a different feature bases. Nevertheless, the efficiency of this batch-incremental technique depends on the size of the batch.

Auto-Encoder

Auto-Encoders (AEs) [68] are a family of Neural Networks (NNs) which are designed for unsupervised learning, for learning a low-dimensional representation of a high-dimensional dataset, where the input is the same as the output. An AE has two main components, (i) the *encoder* step, during which the input data are compressed into a latent space representation; and (ii) the *decoder* step where the input data are reproduced from this new representation. Vincent, Larochelle, Bengio, and Manzagol introduced the denoising AE (DAE), a variant of AE, that extracts features by adding perturbations to the input data and then attempts to reconstruct the original data. Zhou, Sohn, and Lee proposed an online DAE that adaptively uses incremental feature augmentation, depending on the already existing features, to track drifts. However, this work does not address the convergence properties of the training task (the hyperparameters configuration used to construct the network, e.g., the number of epochs) that are crucial in the stream setting.

Unlike other algorithms, NNs naturally handle incremental learning tasks [71]. While dealing with data streams, NNs learn by passing the data in smaller chunks (Mini-Batch Gradient Descent) or an instance at a time (Stochastic Gradient Descent). Using this way, each instance is going to be processed only once without being stored. The advantage of using this kind of technique is that it is not limited to linear transformations. Non-linearities are introduced using non-linear activation functions, NNs are therefore more flexible. Nevertheless, this high-quality results that this family of learners offers come at the price of slow learning speed due to the infinite nature of data and the large parameter space needed.

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) [72], also known as Fisher Discriminant Analysis (FDA), is a linear transformation technique. Contrary to the techniques mentioned earlier, LDA performs a supervised reduction that takes into account the class labels of instances by looking for efficient discrimination of data in a way to maximize the separation of the existing categories (class labels), while other techniques, e.g. PCA, aim at an efficient representation. However, when dealing with evolving data streams, the set of labels of instances may be unknown at each learning stage because new classes may appear (concept evolution) [73].

One way to cope with this issue is to update the discriminant eigenspace when a new class arrives, as introduced in the Incremental LDA (ILDA) approach [74]. Another streaming extension of LDA has been proposed, called IDR/QR [75]. It applies a singular value decomposition suitable for large datasets that uses less computational cost than ILDA. Kim, Stenger, Kittler, and Cipolla proposed an ILDA that incrementally updates the discriminant components using a different criterion. They claim to be more efficient in terms of time and memory than the previous approaches.

Maximum Margin Criterion

Maximum Margin Criterion (MMC) [77] is a supervised feature extractor technique based on the same representation of LDA while maximizing a different objective function. To overcome the limitations of MMC with streaming data, Yan, Zhang, Yan, Yang, Li, Chen, Xi, Fan, Ma, and Cheng proposed an Incremental MMC (IMMC) approach, which infers an online adaptive supervised subspace from data streams by optimizing the MMC and updating the eigenvectors of the criterion matrix incrementally. Hence, the computation of IMMC is very fast since it does not need to reconstruct the criterion matrix when new instances arrive.

The incremental formulation of the proposed algorithm is mentioned in [78] with the proof. A major drawback of this approach is its sensitivity to parameter setting.

2.4.2 Data-Independent Techniques

Data-independent techniques are mainly based on the principle of random projections. These techniques are therefore appropriate for evolving streams because they generate the projection matrices (or functions), and transform data into a low-dimensional space, independently from the input data.

Random Projection

Random projection is a powerful technique for dimensionality reduction that has been widely applied with several mining algorithms for solving numerous problems [79]. RP is based on the Johnson-Lindenstrauss (JL) Lemma 2.4.2 [80] which asserts that N instances from a Euclidean space can be projected into a lower-dimensional space of $\mathcal{O}(\log(N/\epsilon^2))$ dimensions under which pairwise distances are preserved within a multiplicative factor of $1 \pm \epsilon$ [81].

Let $\epsilon \in [0,1]$, $S = \{X_1, ..., X_N\} \in \mathbb{R}^a$. Given a number $m \geq \log(N/\epsilon^2)$, $\forall X_i, X_j \in S$ there is a linear map $A : \mathbb{R}^a \rightarrow \mathbb{R}^m$ such that:

$$(1-\epsilon)\|X_i - X_j\|_2^2 \leq \|AX_i - AX_j\|_2^2 \leq (1+\epsilon)\|X_i - X_j\|_2^2, \tag{2.3}$$

where A is a random matrix that can be generated using, e.g., a Gaussian distribution. Hence, RP offers a computationally-efficient and straightforward way to compress the dimensionality of input data rapidly while approximately preserving the pairwise distances between any two instances.

Compressed Sensing

Compressed Sensing (CS), also called compressed sampling, technique based on the principle that a data compression method has to deal with redundancy while transforming and reconstructing data [23]. Given a sparse/high-dimensional vector $X \in \mathbb{R}^a$, the goal of CS is to measure $Y \in \mathbb{R}^m$ and then reconstruct X, for $m \ll a$, as follows:

$$Y = AX, \tag{2.4}$$

where $A \in \mathbb{R}^{p \times d}$ is called a *measurement*, (*sampling*, or *sensing*) matrix.

The technique has been widely studied and used throughout different domains in the offline framework, such as image processing [82], face recognition [83], and vehicle classification [84]. The basic idea is to use orthogonal features to provably and properly represent sparse and high-dimensional vectors $X \in \mathbb{R}^a$ as well as reconstruct them from a small number of feature vectors $Y \in \mathbb{R}^m$, where $m \ll a$. Two main concepts are crucial the stream recovery with high probability [23]:

Sparsity: CS exploits the fact that data may be sparse and hence compressible in a concise representation. For an instance X with support $\text{supp}(X) = \{l : x^l \neq 0\}$, we define the ℓ_0-norm $\|X\|_0 = |supp(X)|$, so X is s-sparse if $\|X\|_0 \leq s$. The implication of sparsity is important to remove irrelevant data without much loss.

Restricted Isometry Property (RIP): A is said to respect the RIP for all s-sparse data if there exists $\epsilon \in [0, 1]$ such that:

$$(1-\epsilon)\|X\|_2^2 \leq \|AX\|_2^2 \leq (1+\epsilon)\|X\|_2^2, \tag{2.5}$$

where $X \in S$. This property holds for all s-sparse data $X \in \mathbb{R}^a$.

RP and CS are closely related. Random matrices (e.g., Bernoulli, Binomial, Gaussian) are also known to satisfy the RIP with high probability if $m = \mathcal{O}(s \log(a))$ [85], which is essentially a JL type condition on projections using the sensing matrix A. The difference is mainly in terms of how big the matrix A has to be.

Hashing Trick

Hashing trick (HT) [20], also known as feature hashing, is a fact and space-efficient technique that projects sparse instances or vectors into a lower feature space using a hash function. Given a list of keys that represents a set of features from the input instances, it computes then the hash function for each key, which will ensure its mapping to a specific cell of a fixed size vector that constitutes the new compressed instance.

An important point to make is that, generally, the quality of models changes when the size of the hash table increases. Usually, the larger the hash table size is, the better is the model. However, an optimal point can be picked which guarantees almost perfect model, while the output dimension size is not to be very large. The HT technique has the appealing properties of being very fast, simple, and sparsity preserving. A significant advantage to point out is that this technique is very memory-efficient because the output feature vector size is limited, making it a clear candidate for using, especially for online learning on streams.

Locality Sensitive Hashing

The basic idea behind the Locality Sensitive Hashing (LSH) [86] is the application of hashing functions which map, with high probability, similar instances (in the high-dimensional d-space) that have the same hash code to the same bucket. I.e., if instances are phrases that are very similar to each other, they might be different by only one or a couple of words or even one character; hence, LSH will generate very similar, ideally, identical hash codes to increase the probability of collision for those instances. LSH operates by partitioning the space with hyperplanes into disjoint regions, which are spatially proximate. A particular hyperplane is going to cut the space into two half-spaces, and arbitrarily one side is called positive "1" and the other side negative "0"; this will help in classifying the instances for that dimension. The process is iterative: the first bit in the hash code of an instance is assigned with respect to its position. Then, the process keeps cutting the space and assigning bits the same way. Therefore, we obtain the hash codes based on the bits assigned after each hyperplane.

There is an efficiency-computational resources tradeoff with this technique. To achieve good accuracy, LSH requires the use of several hash functions and consequently the memory usage increases which will slow down the reduction process and make it less suitable with large data streams. LSH technique is used in several interesting real-word applications. For instance, Netflix users with similar tastes in movies for recommendation systems, plagiarism given a body of documents, LSH allows to find similar texts. Also, in classification, e.g., classification by topic pages with similar words where pages that have similar sets of words are likely to be about the same topic.

2.4.3 Graph-Based Techniques

Graph-based techniques are also data-dependent techniques that start by constructing a graph based on instance similarities and then operate on this representation.

Isometric Mapping

Isomap [87] is a manifold learning technique that can be viewed as a combination of the principles of PCA and MDS. It starts by building a neighborhood graph on the manifold from which a geodesic distance matrix is constructed. Isomap assumes that pairwise geodesic distances are equal to Euclidean ones (obtained by applying the MDS on the resulting geodesic distance matrix) in the low-dimensional space. Since it requires the computation of pairwise distances, Isomap is thus not appropriate for the incremental setting with large datasets.

Law, Zhang, and Jain proposed a streaming version of Isomap that updates the geodesic distances and the coordinates incrementally. This technique is not fully incremental because a new instance can affect the neighborhood structure and, therefore, the geodesic matrix. Thus, there is a need to examine how this new instance interacts with the existing ones before finding its coordinates. Another incremental Isomap, denoted S-Isomap, has been proposed lately by Schoeneman, Mahapatra, Chandola, Napp, and Zola which does not recompute the whole geodesic distance matrix when a new instance arrives, but only finds its nearest neighbors (that will be used to approximate the geodesic distance between this new observation and the others already available in the batch). This approach fails when used to process data because it assumes that the data are weakly correlated, and thus unable to detect when concept drift takes place.

t-distributed Stochastic Neighbor Embedding

t-distributed Stochastic Neighbor Embedding (tSNE) [90] is one of the most prominent DR techniques in the state-of-the-art. It is a graph-based non-linear technique proposed to visualize high-dimensional data embedded in a lower-space (typically 2 or 3 dimensions) by using the insight that similar instances in the high-dimensional space should be represented by close instances in the low-dimensional space. tSNE uses a fixed parameter named perplexity similar to the number of neighbors that controls the neighborhood size of each node in the graph, which prevents it from preserving global data structure (only local). The main weakness of tSNE in our context is about the scalability, i.e. to add more instances, we need to re-run tSNE from scratch.

Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection (UMAP) [26] is a new manifold technique, similar to tSNE, that has attracted much attention recently and is built upon rigorous mathematical foundation through the Riemannian geometry. UMAP starts by constructing open balls over all instances and building simplicial complexes. The space reduction is obtained by finding a representation, in a lower-space, that closely resembles the topological structure in the original space. Given the new dimension, an equivalent fuzzy topological representation is then constructed. Afterward, UMAP optimizes it by minimizing the cross-entropy between these two fuzzy representations [26]. In addition to being faster than tSNE, UMAP offers also a better visualization quality by preserving more of the global structure. Unlike tSNE [90], UMAP has no restriction on the projected space size making it useful not only for visualization, but also as a general DR technique for mining algorithms.

2.5 Evaluation Metrics

After training the model it is crucial to validate it by evaluating the classifier and verifying its applicability. Several methods exist and the most common method is the *prequential*. The prequential evaluation [91], known also as the *interleaved test-then-train* evaluation, is a popular evaluation method applied exclusively on data streams. In the prequential evaluation, instances are used to test–or *predict* on– the current model before using them to train–or *update*– the model. To evaluate the performance of our proposed classification algorithms, we emphasize on three evaluation criteria that are strongly related.

1. *Accuracy.* The accuracy (AC) [92] of a learning algorithm is the most pertinent concern that measures the percentage (%) of correct classified instances c_i that a model makes on a dataset, it is defined in Equation (2.6):

$$AC = \frac{\sum_i c_i}{N}, \tag{2.6}$$

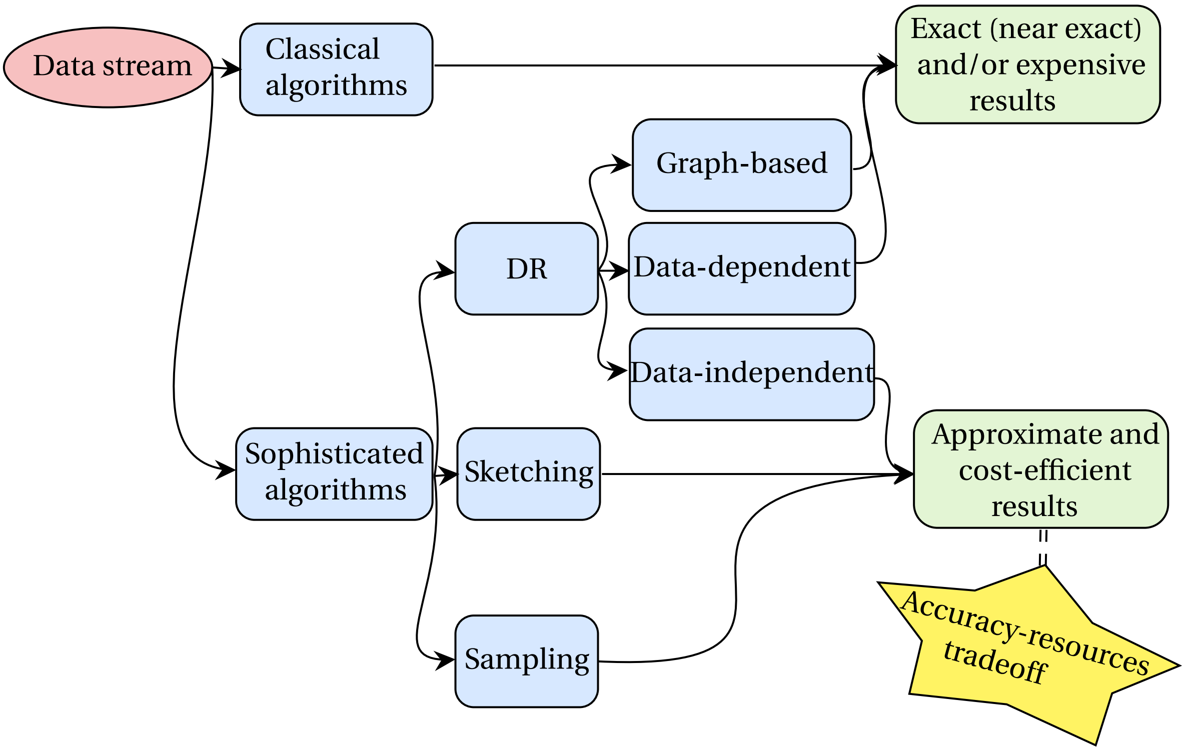


Figure 2.8: Handling data stream constraints. Traditional classification algorithms offer exact accuracy but are expensive because they process the entire stream (that may be high-dimensional). Sophisticated algorithms use an internal summarization technique(s) in order to improve their performance what leads generally to an accuracy-resources tradeoff.

where N presents the total number of instances received so far from the stream.

The most accurate classifier is the one that achieves the highest accuracy and makes few mistakes when predicting the class labels of unlabeled instances.

2. *Running time.* A good classification algorithm processes instances as fast as possible once they arrive. The running time comprises any internal pre-processing (e.g., dimensionality reduction), learning, and prediction step as new instances arrive.

3. *Memory.* The memory – measured in megabytes (MB) – used by an algorithm is: (i) the memory used to store the current model(s); and/or (ii) the memory used to store some running statistics useful for the incremental processing of data streams.

2.6 Discussions

DR plays a significant role in the data stream mining area since it aims at keeping the most relevant features in order to reduce the computational cost of stream mining algorithms (Figure 2.8).

Techniques such as PCA, LDA, and MDS are the most classical ones for DR. As we mentioned before in Section 2.7, some versions of the data-dependent techniques have been proposed to deal with evolving data streams. Nevertheless, this category of techniques usually provides good accuracy when combined with stream data mining algorithms. On the other hand, data-independent techniques are naturally adapted to the evolving environment of data streams and do not suffer from the scalability problem. Moreover, using data-independent techniques is extremely fast because it is performed without including the input data content. This transformation performs as well, if not better, as data-dependent transformation because it is less sensitive to new unseen instances and could benefit from the infinite nature of streams. Sometimes data-independent schemes could destroy any interpretability in the case of visualization. Thus, as illustrated in Figure 2.8, the choice of the technique (data-dependent or data-independent) leads to an accuracy-resource tradeoff that may depend on the problem being solved and the algorithm used (e.g., when using a graph-based manner for visualization to preserve the neighborhood and the global structure of data).

2.7 Conclusion

Processing data streams is a big challenge in the data mining field because of the additional constraints created by the large volume of unbounded data. In addition to that, the high dimensionality of data streams in some domains makes the issue more challenging in the stream framework. In this chapter, we studied the basic notions of evolving data streams and discussed the different processing techniques that could be used by mining algorithms to address the the stream requirements. We provided thereafter an overview of state-of-the-art classifiers and summarization techniques proposed in the stream setting.

In the second part of the thesis, we will detail our contributions consisting in novel versions of some classifiers that use summarization techniques under the streaming framework. In the next chapter, we start by presenting our new developments of the Naive Bayes algorithm using sketching and DR techniques.

Part II

Summarization-Based Classifiers

After providing in the first part a deep overview of the literature including the concepts used in our work, we focus now on our contributions. In this second part, the major part of this thesis, we detail the developments that we have proposed in order to build our classification approaches based on different summarization techniques for evolving data streams. This part is composed of four chapters:

- Chapter 3 presents the basic concepts of the count-min sketch [19] which is used to create our classification approach based on Naive Bayes, called *SketchNB*. We attempt thereafter to increase the performance of the latter with high-dimensional data streams using a fast dimensionality reduction technique, the hashing trick. Finally, an approach on the data stream framework without has to address concept drifts. For this to happen, we incorporate an efficient change detector with strong guarantees, ADWIN [21].

- Chapter 4 explores another dimensionality reduction technique based on random projection, compressed sensing [23], and uses it in conjunction with the k-nearest neighbors classifier. The resulting algorithm, CS-kNN, aims to reduce memory and time requirements. Theoretical guarantees characterizing the similarity between the kNN neighborhoods before and after the projection are provided. To obtain more stable predictive performance, because of the stochasticity engendered by the compressed sensing, we use the CS-kNN approach as a base learner to the Leveraging Bagging ensemble-based method.

- Chapter 5 presents a similar strategy to the one described in Chapter 4. We propose an ensemble method, based on the adaptive random forest [25], which pre-processes the input instances using multiple CS random matrices to reduce the resource usage.

- Chapter 6 remains in the context of kNN and explores a new visualization technique, UMAP [26], that has attracted a lot of attention because of its high-quality results and neighborhood-preservation nature. We use UMAP as a dimensionality reduction technique in a batch-incremental manner to compress the input space size. The output is then used with the kNN instead of using high-dimensional data. Promising results are obtained with this batch-incremental UMAP-kNN approach because both methods, kNN and UMAP, are based on exploring the neighborhood of instances, using a measure of “proximity”.

3

Sketch-Based Naive Bayes

Contents

This chapter contains results from our paper [93] published at the IEEE International Conference on Big Data, (BigData) 2018 under the title "*Sketch-Based Naive Bayes Algorithms for Data Streams*" and presented as a poster at the Machine Learning Summer School (MLSS)[1] 2019.

3.1 Introduction

As mentioned in Chapter 2, several algorithms have been proposed in the literature to deal with this problem such as decision trees, naive Bayes, neural networks, or k-nearest

[1] https://smiles.skoltech.ru/mlss2019.

neighbors. To cope with the evolving nature of data streams and their challenges previously enumerated, techniques such as sketches, a class of specialized algorithms that can produce approximate results efficiently with mathematically proven error bounds, are useful to process infinite data using fewer resources.

The sketches are a data structure that stores the counts of *categorical* data, e.g., hashing a word into a particular cell using a hash function. Naive Bayes is a frequency-based algorithm that keeps counts about categorical data during the learning phase. Sketches and naive Bayes could be unified together to obtain a memory-efficient naive Bayes.

In this chapter, we focus on these important challenges in classification over data streams, by exploring the benefits of introducing a sketch technique for streams. The main focus of this work attempts to extend the stream naive Bayes algorithm to deal with massive data by storing data with high-quality approximations in a sketch table which allows both fast predictions and the use of a minimal amount of space for training (which answers **Q1**). We thereafter extend this first proposed approach to handle changes in the evolving distribution using a concept drift mechanism (**Q3**), namely ADWIN [21]. Finally, we propose a third contribution to the two stated algorithms that aims to address high dimensionality (**Q1** and **Q2**) using the hashing trick technique (HT) [20], detailed in Section 2.4.2.

The remainder of the chapter is organized as follows. In Section 3.2, we present the main components related to our contributions. In Section 3.3, we detail our proposed approaches. Section 3.4 discusses the different experiments performed on both artificial and real datasets. Finally, we draw our conclusion in Section 3.5.

3.2 Preliminaries

We assume that the data stream S contains an infinite number of *instances* $X_1, X_2, \ldots, X_N, \ldots$, where each instance is a vector containing a attributes, denoted by $X_i = (x_i^1, x_i^2, \ldots, x_i^a)$. We will denote by N the number of instances encountered thus far in the stream. The classification problem consists in assigning each instance X_i to a class $c_j \in C$, where C is the set of classes.

3.2.1 Naive Bayes Classifier

One of the most often used classifiers is Naive Bayes (NB) [38]. It uses the assumption that the attributes are all independent of each other and, w.r.t. the class label, uses Bayes's Theorem to compute the posterior probability of a class given the evidence (the training data). This assumption is obviously not always true in practice, yet the NB classifier has a surprisingly strong performance in real-world scenarios. Using Bayes's theorem one can

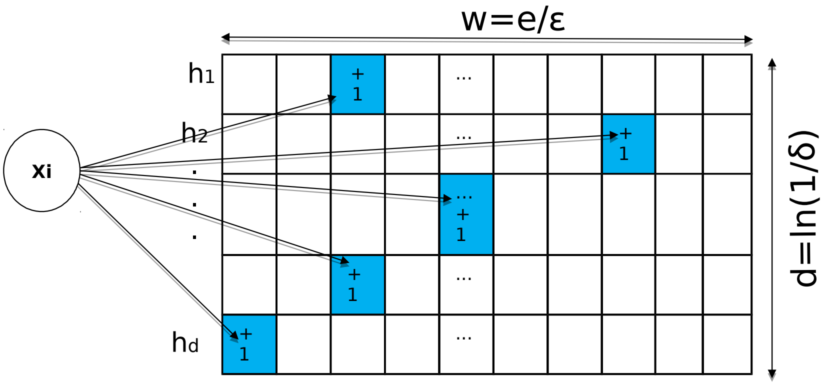


Figure 3.1: Count-min sketch of a width w and a depth d.

compute the probability of each class:

$$P(C \mid A_1, \ldots, A_a) = \frac{P(A_1, \ldots, A_a \mid C) \cdot P(C)}{P(A_1, \ldots, A_a)}, \tag{3.1}$$

where $P(C|A_1, \ldots, A_a)$ is the posterior probability of the target class given the attributes, $P(C)$ is the prior probability of the class, $P(A_1, \ldots, A_a|C)$ is the likelihood, and $P(A_1, \ldots, A_a)$ is the prior probability of attributes. Once the probabilities are computed, the class having the highest probability is chosen as the predicted class.

The training in naive Bayes is straightforward. To compute class probabilities, we estimate them as a fraction of the instances seen thus far, as follows:

- Estimate $P(C)$ as the fraction of records having $C = c_j$,

$$P(C = c_j) = \frac{Count(C = c_j)}{N}.$$

- Estimate $P(X = A_1, ..., A_a|C)$ as the fraction of records with $C = c_j$ for which $X = A_1, ..., A_a$,

$$P(X = A_i|C = c_j) = \frac{Count(X = A_i \wedge C = c_j)}{Count(C = c_j)}.$$

We use this attribute independence assumption in the following to construct our NB approach with the Count-Min Sketch (CMS) technique.

3.2.2 Count-Min Sketch

Nowadays, applications generate data at rates and volumes that cannot be reasonably stored. To cope with the vast scale of information, one way is to use *synopsis* techniques [33, 94, 95]. Among them, the CMS [19] which is a generalization of Bloom filters [34] introduced to count items of a given type using approximate counts that are theoretically sound.

Definition 3.1 *(Count-min Sketch). CMS consists of a two-dimensional array of $w \cdot d$ cells of counters, having a width w of columns, and a depth d of rows. w and d are controlled by the approximation parameters ϵ and δ, such that, with probability $1 - \delta$, the approximate counts obtained from the sketch are within an absolute error ϵ of the true counts. Each row in d is a different hash function $h_1, h_2, \ldots, h_d$; each h_i is used to determine in which of the w counters on row i a count is incremented. Figure 3.1 shows the CMS table. Depending on the ϵ and δ parameters, the CMS should be initialized using the following dimensions: $d = \frac{e}{\epsilon}$ and $w = \ln \frac{1}{\delta}$, with all cells set to 0.*

Each time a new instance arrives in the data stream S, and for each attribute and class, each hash function h_i is applied and the corresponding counter (in the range $1, \ldots, w$) is incremented. When an instance needs to be classified, the corresponding counts using the same hash functions need to be retrieved, i.e., when training the classifier, one needs to look at all cells for the attribute and the class being estimated. This is done by taking the minimum overall values in the corresponding cells. This is because, since each cell has been incremented each time an attribute has been seen, each cell represents an upper bound on the actual value. Assuming a data stream with N arrivals, let q_i be the true count of an item being estimated. It has been shown in [19] that the estimated count for an item i is at least q_i since all the inserts are non-negative, and due to collisions, the counts can be over-estimated to at most $q_i + \epsilon \cdot N$ with probability at least $1 - \delta$, i.e., an upper bound to the estimate.

Little research has focused on using efficient data structures designed to reduce memory usage, such as CMS, with standard classifiers. Kveton, Bui, Ghavamzadeh, Theocharous, Muthukrishnan, and Sun proposed three graphical model sketches algorithms that estimate the marginal and the joint probabilities within a Bayesian network. Authors experimented with the special case of Bayesian networks, naive Bayes. After analyzing them, it was proved that their proposed GMFactorSketch is the best approximation. The main idea of the approach is to use $2a - 1$ sketch tables, one for each variable and one for each variable-parent pair in the graph. Given a test example, it retrieves the approximated count for each attribute from each corresponding sketch tables to compute the conditional probabilities and to predict thereafter the class label. None of the above-mentioned approaches is efficient in terms of memory with large datasets, as evidenced in our experimental Section 3.4.

3.3 Sketch-Based Naive Bayes Algorithms

Sketch-based techniques summarize massive data streams in a limited space by using multiple hash functions to decrease the probability of having wrong counts due to collisions. The main idea behind the sketch-based approaches is the use of the CMS technique to store the stream for memory-efficiency during the learning phase.

3.3.1 *SketchNB* Algorithm

We aim to adapt the CMS [19] to the classic naive Bayes classifier by leveraging its strong theoretical guarantees. The independence assumption in NB means that each attribute can be counted separately; this simplifies the learning for a large number of attributes, and allows us to use the sketch efficiently. During the classification process, the sketch table will be used in two steps:

- *Learning*: updating the sketch table for each attribute each time a new instance arrives.
- *Prediction*: retrieving the counts of a given instance from the CMS table, and use them to compute the naive Bayes probability (see Equation 3.1).

Let us start by discussing the learning process using the CMS, as described in Algorithm 1. Once an instance $X_i = (x_i^1, x_i^2 ... x_i^a)$ is received, the classification algorithm starts by updating the sketch with the counts of the attributes value by inserting each of the attribute value as $\langle k, x_i^k, c \rangle$, $k \in [0 \cdots a]$. The sketch table will thus contain the counts of the attributes values, using each of the d hash functions a times according to the number of attributes, so $O(d \cdot a)$ times in total.

Figure 3.1 shows the updating process of the sketch table. We start firstly by creating the sketch table and initializing it to zero (line 2). Given an instance X_i that belongs to a class c_j, each attribute value x_i^k is mapped to one counter in each row using the set hash functions. Each of those counts gets incremented whenever a particular similar attribute value in the same class is seen (ligne 5). Therefore, each of these numbers is going to be an upper bound. Since all of them are going to be an upper bound, only the minimum can be taken for the prediction phase using the same set of hash functions used during the update process.

Algorithm 1 Learning phase. **Symbols:** $S = \{X_1, X_2, ...\}$: labeled data stream; ϵ: epsilon; δ: delta.

```
1: function UPDATESKETCH($S, \epsilon, \delta$)
2:     CMS←0                      ▷ create the sketch with $w \cdot d$, $w = \lceil \frac{e}{\epsilon} \rceil$, $d = \lceil ln\frac{1}{\delta} \rceil$
3:     for all $X_i \in S$ do
4:         for all $x_i^k \in X_i$ do                      ▷ $k \in [1 \cdots a]$
5:             CMS[$l, h_l(< k, x_i^k, c_j >)$] +=1     ▷ $l = [1 \cdots d]$, increment the cells using $d$ hash functions in $[1..w]$
6:         end for
7:     end for
8: end function
```

Obverse that Algorithm 2 assumes that we have one stream used to present training instances, and a stream $S\prime$ for predictions; this works by presenting, at each timestamp i, an unlabeled instance X_i. To predict the class label, using the Equation (3.1) (line 5) and the counts retrieved from the CMS table (line 4), we compute an estimation of the class having the highest probability (line 7).

Theoretical guarantees about the sketch size, w and d, are provided in the following.

Algorithm 2 Prediction phase. **Symbols:** $S' = \{X_1', X_2', \ldots\}$: data stream; CMS: the count min sketch.

```
1: function EstimateSketch($S'$, CMS)
2:     for all $X_i' \in S'$ do
3:         for all $x_i^k{}' \in X_i'$ and $c_j \in C$ do                          ▷ $k \in [1 \cdots a]$
4:             $count(x_i^k{}') \leftarrow$ CMS$[l, h_l(< k, x_i^k{}', c_j >)]$    ▷ retrieve the counts from the CMS
5:             $P(x_i^1, \ldots, x_i^a \mid c_j) = \prod_{k=1}^{a} count(x_i^k{}')$    ▷ compute the probability
6:         end for
7:         $pc \leftarrow \max_{c \in C} P(x_i^1{}', \ldots, x_i^a{}' \mid c)$
8:     end for
9: end function
```

To determine the efficiency of the proposed SketchNB algorithm, we need to analyze the behavior of the sketch table since we are retrieving approximate counts from it. It has been shown that for all the inserts, the counts are non-negative and may be over-estimated because of collisions [19]. Let $f_j(x_i^k)$ be the fractional count of the attribute value x_i^k from the instance X_i in the jth class. Its estimated fractional count is denoted by $\hat{f}_j(x_i^k)$. The latter has the following guarantees: $f_j(x_i^k) \le \hat{f}_j(x_i^k)$; and with probability at least $(1-\delta)$:

$$\hat{f}_j(x_i^k) \le f_j(x_i^k) + N \cdot a \cdot \epsilon. \tag{3.2}$$

Using one sketch table with size $\left[\frac{e}{\epsilon} \times ln\frac{1}{\delta}\right]$, after the processing of N a-dimensional instances from the stream, the counts are over-estimated to within $(N \cdot a \cdot \epsilon)$ of their true values. Since we are using NB, as described in the second step of Algorithm 2, for each incoming instance, each class $c_j \in C$, and each attribute a, we are doing multiple extractions at the same time by retrieving $t = |C| \cdot a$ counts, where $|C|$ is the number of class labels:

Theorem 3.1 Given a data stream S of instances, denoted X_i, inserted in the sketch, ϵ, and δ, with probability at least $1 - \Delta = (1-\delta)^t$ we have:

$$\bigwedge_{k=1}^{t} (\hat{f}_j(x_i^k) \le f_j(x_i^k) + Na\epsilon) = \text{True}. \tag{3.3}$$

This means that we need to set the sketch according to the following setting,

$$\delta = 1 - \sqrt[t]{1-\Delta}, \tag{3.4}$$

or

$$d = \ln \frac{1}{1 - \sqrt[t]{1 - \Delta}}. \tag{3.5}$$

The above result will be used to set a crucial parameter to construct the sketch which is δ in order to fix the depth of the sketch table. The obtained δ will lead to a deeper sketch that would be able to maintain the entire stream. Consequently, we will obtain a more accurate estimation of counts by avoiding collisions.

We would like to determine the accuracy of SketchNB algorithm. The process of this algorithm is described in the pseudocode of Algorithm 2. The latter consists on retrieving, from the sketch table, the fractional counts for each attribute value, so one can apply the NB equation for each class label. In order for this to happen, we need to compute the product of the estimated fractional counts for each hash function in $[1, \ldots, \ln \frac{1}{\delta}]$.

Theorem 3.2 Let $\epsilon\prime = Na\epsilon$, and $\hat{f}_j(x_i^k)$ be the estimated fractional count of the attribute value x_i^k in the jth class. We have:

$$\prod_{k=1}^{a} \hat{f}_j(x_i^k) \leq \prod_{k=1}^{a} (f_j(x_i^k) + Na\epsilon). \tag{3.6}$$

Then, with probability at least $(1 - \Delta)$,

$$\prod_{k=1}^{a} \hat{f}_j(x_i^k) \leq \prod_{k=1}^{a} f_j(x_i^k) + \sum_{p=1}^{a} \frac{a!}{p!(a-p)!} (\epsilon\prime)^p. \tag{3.7}$$

Proof. Given an instance X_i:

$$\begin{aligned} \hat{f}_j(x_i^1) \cdot \hat{f}_j(x_i^2) \ldots \cdot \hat{f}_j(x_i^a) &\leq (f_j(x_i^1) + \epsilon\prime) \cdot (f_j(x_i^2) + \epsilon\prime) && (3.8) \\ &\ldots \cdot (f_j(x_i^a) + \epsilon\prime) \\ &= f_j(x_i^1) \ldots f_j(x_i^a) + f_j(x_i^1) \cdot \epsilon\prime \\ &+ f_j(x_i^2) \cdot \epsilon\prime \ldots + f_j(x_i^a) \cdot \epsilon\prime \\ &+ f_j(x_i^1) \cdot f_j(x_i^2) \cdot \epsilon\prime \cdots && (3.9) \end{aligned}$$

Since all the frequencies are non-negative, we know that the range of possible fractional counts is $[0, 1]$, i.e. at most 1, thus, 1 will be an upper bound to the fractional counts. So, with certainty we know that $\hat{f}_j(x_i^k) \cdot \epsilon\prime \leq \epsilon\prime$, and any product of the estimated fractional counts is always less than 1. So, by applying Pascal's triangle, we obtain:

$$\begin{aligned}\prod_{k=1}^{a} \hat{f}_j(x_i^k) &\leq \prod_{k=1}^{a} f_j(x_i^k) + f_j(x_i^1) \cdot \epsilon\prime + f_j(x_i^2) \cdot \epsilon\prime \cdots \\ &+ f_j(x_i^a) \cdot \epsilon\prime + f_j(x_i^1) \cdot f_j(x_i^2) \cdot \epsilon\prime \cdots \\ &\leq \prod_{k=1}^{a} f_j(x_i^k) + \sum_{p=1}^{a} \frac{a!}{p!(a-p)!} (\epsilon\prime)^p \\ &= \prod_{k=1}^{a} f_j(x_i^k) + \sum_{p=1}^{a} C_a^p (\epsilon\prime)^p. \end{aligned} \tag{3.10}$$

This completes the proof. □

This observation shows that, with probability $1 - \Delta$, the quantity of error due to collisions is at worst equal to $\sum\limits_{p=1}^{a} C_a^p (\epsilon\prime)^p$. This leads to the following corollary.

Corollary 1 Let E be big epsilon and N be the number of instances seen so far from the stream. An immediate result from Equation (3.10) is the following:

$$\epsilon = \frac{\sqrt[a]{aNE+1} - 1}{aN}, \tag{3.11}$$

or

$$w = \frac{eaN}{\sqrt[a]{aNE+1} - 1}. \tag{3.12}$$

Proof. Let $\epsilon\prime = aN\epsilon$. After processing N a-dimensional instances from the stream, the quantity of error caused by collisions in Theorem 3.2 must be the same as aNE according to Theorem 3.1.

$$\begin{aligned} \sum_{p=1}^{a} C_a^p (aN\epsilon)^p &= aNE \\ C_a^0 (aN\epsilon)^0 + \sum_{p=1}^{a} C_a^p (aN\epsilon)^p &= aNE + 1 \\ \sum_{p=0}^{a} C_a^p (aN\epsilon)^p 1^{a-p} &= aNE + 1 \\ (aN\epsilon + 1)^a &= aNE + 1 \\ aN\epsilon + 1 &= \sqrt[a]{aNE+1} \\ \epsilon &= \frac{\sqrt[a]{aNE+1} - 1}{aN}. \end{aligned}$$

This completes the proof. □

Our proposed SketchNB algorithm possesses strong theoretical guarantees when setting the depth and width of the sketch using Equations (3.11) and (3.4) respectively. This means that, in order to keep the guarantees overall attributes and classes, we need to set a much deeper and wider sketch table than for the case where we have only one counter.

Moreover, Equations (3.11) and (3.12) assume that the size of the stream is known. In real-world scenarios, where the stream can be infinitely increasing, this means that our sketch, along with the hash functions, will need to increase with the size of the stream. The theoretical parameters derived by us here provide good results, but they can lead to a relatively large sketch; in some cases, this may not be desirable due to space reasons. We can however perform some optimizations to the space needed. The size of the stream, N, can be too large and even infinite; instead, it can simply be a "sliding window" over which error guarantee is provided. To achieve this, we introduce a scaling constant b to the computations, in the following manner. First, we can set ϵ as follows:

$$\epsilon = b \cdot \frac{\sqrt[a]{aNE+1}-1}{aN},$$

then, we choose the width w as follows:

$$w = \frac{eaN}{b(\sqrt[a]{aNE+1}-1)}. \tag{3.13}$$

Note that the depth d still remains the same: it depends only on the parameter Δ. It is also necessary to point out that $b > 1$. When a increases, the sketch table size increases accordingly. A higher value of b reduces the width of the sketch table. In this work, b will be picked up experimentally.

3.3.2 AdaSketchNB Algorithm

One crucial issue when dealing with a very large stream is the fact that the underlying distribution of the data can change at any moment, a phenomenon known as *concept drift*, and we direct the reader to [9] for a survey on this concept.

A widely popular algorithm that handles concept drift is ADaptive WINdowing (ADWIN) [21] used in a few machine learning algorithms such as Hoeffding adaptive tree [44] and leveraging bagging [24]. The main idea of ADWIN is to maintain a variable-length window W with the most recently seen instances with the property that the window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window" [21].

In our context, it is important to incorporate a change detector mechanism. We choose ADWIN here due to its strong theoretical guarantees and good practical results.

Algorithm 3 Learning phase. **Symbols:** $S = \{X_1, X_2, \ldots\}$: data stream; ϵ: epsilon; δ: delta; W: sliding window; D: change detector; α: drift threshold.

```
1:  function UPDATEADASKETCHNB($S, \epsilon, \delta, W, \alpha$)
2:      CMS←0                ▷ create the sketch with $w \cdot d$, $w = \left[\frac{e}{\epsilon}\right]$, $d = \left[ln\frac{1}{\delta}\right]$
3:      for all $X_i \in S$ do
4:          for all $x_i^k \in X_i$ do                ▷ $k \in [1 \cdots a]$
5:              CMS[$l, h_l(< k, x_i^k, c_j >)$] +=1    ▷ $l = [1 \cdots d]$, increment the cells using $d$ hash functions in $[1..w]$
6:          end for
7:          if $c = pc$ then                ▷ true class = predicted class
8:              $W \leftarrow 1$
9:          else
10:             $W \leftarrow 0$
11:         end if
12:         if $D(\alpha)$ then                ▷ if a drift is detected
13:             CMS $\leftarrow 0$
14:         end if
15:     end for
16: end function
```

Against this background, we propose a second algorithm, AdaSketchNB described in Algorithm 3, that incorporates to the SketchNB algorithm an adaptive strategy using ADWIN change detector in the learning phase, for the entire sketch table, in order to track drifts. It works on the data distribution by detecting the change in the class label. Given a stream S, when a new training instance X_i arrives, we compare the predicted class label (under our current model) and the true class label. Then, we update the sliding window W by adding 1 if this comparison holds true, 0 otherwise (lines 7 and 9). Once a change is detected in the distribution, i.e., the newly generated instances changed from the original class distribution of the classifier built up to now, the sketch table will be re-initialized to learn the new model, corresponding to the new distribution (lines $12 - 14$). The prediction and updating phases remain the same as the basic SketchNB algorithm.

Suppose that we have a high-dimensional data stream that we need to maintain in the sketch. Should we extract counts a number of a times – a potentially very high number – to predict the class label for each instance; do we also need to update the sketch a times?

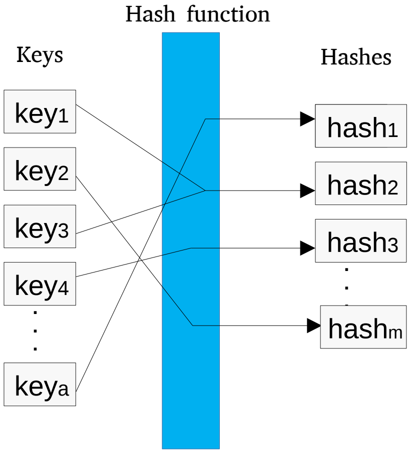


Figure 3.2: Hashing trick.

3.3.3 SketchNB$_{HT}$ and AdaSketchNB$_{HT}$ Algorithms

It is an undeniable fact that the sketches summarize massive data streams within a limited space, but hashing trick (HT) [20] can be used for further improvements with large datasets. The main idea behind the HT is presented in Figure 3.2, and explained in Section 2.4.2.

We propose SketchNB$_{HT}$ and AdaSketchNB$_{HT}$ algorithms that attempt to enhance the SketchNB and the AdaSketchNB in terms of memory and processing time while maintaining high accuracy. These approaches could perform better on high-dimensional data by integrating the HT technique to compress down the input space dimensionality. In fact, to make the analysis of high-dimensional and large datasets tractable, firstly, we reduce the size of the input dimension by applying the HT technique to instances one by one, incrementally. We obtain thereafter instances with significantly smaller dimension that will be processed by either the estimate procedure in Algorithms 2 or 3 depending on whether we are using SketchNB$_{HT}$ or AdaSketchNB$_{HT}$, respectively.

A numerical representation of each instance is obtained after applying the DR using the HT technique. So how can we update the sketch (that works with nominal data) right-after?

By applying the HT technique [20] on an input vector, we are supposed to obtain a numerical representation which prevents it from fitting to the sketch table. Instead, we are obtaining a binary vector by updating a cell only once, i.e., we do not increment the cells, we just put 1 if a cell contains 0, and it remains the same if it contains already 1. Thus, we get a discretized representation able to fit into the sketch table and also avoid wrong values due to collisions. Then, we store the instances in the sketch table in the same way as the SketchNB algorithm. Consequently, instead of updating a times the sketch table, we update it henceforth only m times, where $m \ll a$. In other words, the HT is treated as an internal pre-processing and transforming instances within the SketchNB and AdaSketchNB algorithms. Using this manner, we guarantee that out of memory error that may occur with standard classifiers for data streams will not happen using the hashing trick and the sketches.

3.4 Experimental Evaluation

We conduct several experiments to assess the classification performance of the aforementioned proposals, SketchNB, AdaSketchNB , SketchNB$_{HT}$, and AdaSketchNB$_{HT}$ algorithms. To evaluate them, we are interested in three main dimensions; the accuracy (%) as the final percentage of instances correctly classified, the memory (B), and the time (Sec) required to learn and predict from data.

3.4.1 Datasets

We use 8 synthetic and 7 real world datasets on our experiments, where 5 of them contain high-dimensional data. The synthetic datasets created using the data generators provided by MOA [97] include drifts. In the case of real datasets, we do not know whether a drift exists or not, but we still evaluate it using the change detector mechanism coupled with our proposals. Table 3.1 presents a short description of each dataset, and further details are provided in what follows.
SEA. The SEA Generator proposed by [98]. It is generated with 3 attributes and 2 decision classes with concept drift and simulates 3 gradual drifts.
RBF. The Radial Basis Function (RBF) generator creates centroids at random positions, and each one has a standard deviation, a weight and a class label. This dataset simulates drift by moving the centroids with constant speed.
LED. The LED generator originates from the CART book [99] and simulates concept drifts. It produces 24 attributes, of which 17 are irrelevant. The goal is to predict the digit displayed on the LED display. We generate also LED$_g$ that simulates 3 gradual drifts.
AGR. The AGRAWAL generator [100] creates data stream with 9 attributes and 2 classes. A factor is used to change the original value of the data. AGR is used to simulate 3 gradual drift in the generated stream.
HYP. The HYPERPLANE generator [101] used to generate streams with gradual concept drift by changing the values of its weights. We parameterize HYP with 10 attributes and a magnitude of change equals to 0.001.
Tweets. Tweets is a text generator that simulates sentiment analysis on tweets, where messages can be classified into two categories depending on whether they convey positive or negative feelings. Tweets$_1$ and Tweets$_2$ produce 1,000 and 10,000 attributes respectively.

Table 3.1: Overview of the datasets.

Dataset	#Instances	#Attributes	#Classes	Type
SEA	1,000,000	3	2	Synthetic
RBF	1,000,000	10	5	Synthetic
LED	1,000,000	24	10	Synthetic
LED_g	1,000,000	24	10	Synthetic
AGR	1,000,000	9	2	Synthetic
HYP	1,000,000	10	2	Synthetic
Tweets_1	10,000	1,000	2	Synthetic
Tweets_2	10,000	10,000	2	Synthetic
KDD99	494,020	41	23	Real
Cover	581,012	54	7	Real
Elec	45,312	8	2	Real
Poker	829,201	10	10	Real
Enron	1,702	1,054	2	Real
IMDB	120,919	1,001	2	Real
CNAE	1,080	856	9	Real

KDD99. KDD cup'99[2] for network intrusion detection. This dataset contains 41 attributes and 23 classes. It has been often used to evaluate big data streams algorithms' performance.

Cover. The forest covertype dataset obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. It contains 54 attributes and 7 classes.

Elec. The Electricity market dataset described firstly by Harries and Wales. In this marked the prices changes every 5 minutes and are affected by demand, supply, season, weather and time. It contains two possible class labels identifying the changes of the price relative to a moving average of the last 24h.

Poker. The Poker hand dataset consists of 829,201 instances and 10 attributes. Each instance of the Poker-Hand dataset is an example of a hand consisting of five playing cards.

Enron. The Enron corpus dataset is a large set of email messages that was made public during the legal investigation concerning the Enron corporation [103]. This cleaned version of Enron consists of 1,702 instances and 1,054 attributes.

IMDB. IMDB[3] movie reviews dataset was first proposed for sentiment analysis [104], where reviews have been pre-processed, and each review is encoded as a sequence of word indexes (integers).

CNAE. CNAE is the national classification of economic activities dataset, initially used in [105]. It contains 1,080 instances, each of 856 attributes, representing descriptions of

[2] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[3] http://waikato.github.io/meka/datasets/.

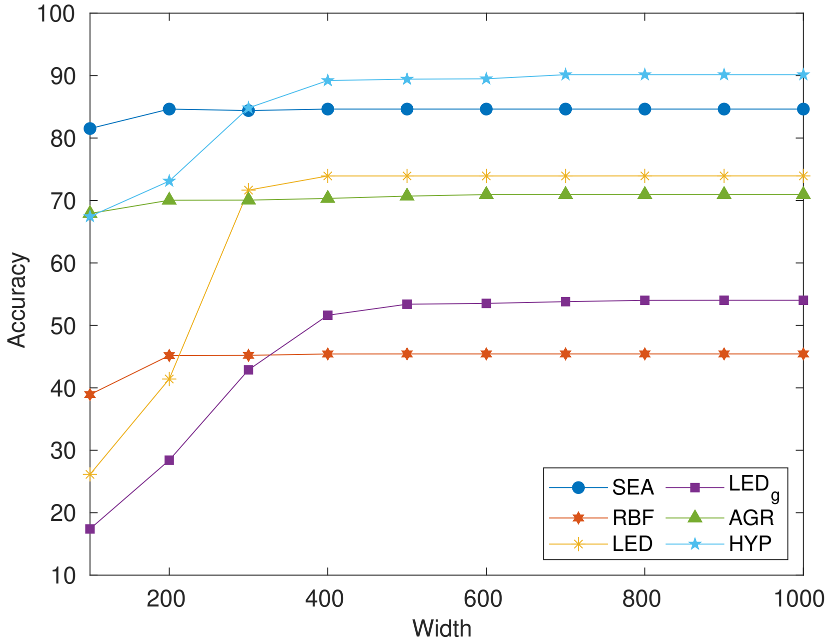


Figure 3.3: Classification accuracy with different sketch sizes for different synthetic datasets.

Brazilian companies categorized into 9 classes. The original texts were pre-processed to obtain the current highly sparse dataset.

Handling continuous attributes in data stream classifiers is a bit tricky; one way to deal with this issue is to consider that the datasets will follow a series of Gaussian distributions. We use a simpler way here to pre-process the datasets and transform all numerical attributes into discrete attributes [106]. The discretization was performed using WEKA [107], where each numerical attribute was discretized to an equal-width histogram having 10 bins.

3.4.2 Results and Discussions

The experiments were performed, implemented and evaluated in Java using the MOA framework [97] and the datasets explained in Section 3.4.1.

Despite the theoretical bounds provided on the size of the sketch, we need to optimize the sketch table parameters to allow better space usage. To do so, we use both of the synthetic and real datasets to parameterize the sketch size for each dataset, by controlling the parameter b. In order to fix the value of the constant b in Equation (3.13), we perform some experiments. We set the default values for different parameters to the sketch table, $\Delta = 0.1$, $E = 0.01$ and $N = 10^5$ for Equations (3.5) and (3.12) to fix the sketch table size, and we set the default confidence bound to ADWIN.

Figure 3.3 illustrates the appropriate width for the first 6 synthetic datasets, i.e., what is the basic width that leads to an accurate model for each dataset. We notice that for the same number of attributes and classes, e.g., LED and LED$_g$, we obtain the same width that is able

Table 3.2: Accuracy comparison of SNB, GMS, NB, kNN, ASNB, AdNB, and HAT. Bold values indicate the best results per dataset.

Dataset		Non-adaptive				Adaptive	
	SNB	GMS	NB	kNN	AdSNB	AdNB	HAT
SEA	**84.64**	70.83	**84.64**	69.95	**86.70**	**86.70**	**86.70**
RBF	44.66	32.62	**45.43**	35.14	47.14	45.91	**82.73**
LED	**73.94**	73.82	**73.94**	44.02	**73.90**	**73.90**	70.87
LED$_g$	54.02	**54.23**	54.02	43.18	72.71	**73.09**	72.60
AGR	70.95	61.67	**70.96**	68.19	79.98	82.53	**89.34**
HYP	**90.16**	81.93	**90.16**	64.18	90.88	**91.16**	81.32
Tweet$_1$	87.50	–	**89.26**	68.73	87.54	**89.26**	84.64
Tweet$_2$	74.42	–	**91.41**	77.72	74.42	**91.41**	85.38
Syn ∅	72.54	–	**75.23**	58.89	76.66	79.24	**81.73**
KDD99	89.72	97.43	95.51	**99.69**	91.17	**99.59**	98.93
Cover	62.08	50.69	62.86	**80.07**	**95.80**	83.17	86.56
Elec	66.64	63.90	66.53	**73.89**	71.70	**73.31**	72.48
Poker	56.81	56.43	58.84	**74.98**	69.22	74.58	**74.73**
Enron	75.50	–	77.44	**95.24**	84.61	85.61	**91.83**
IMDB	64.98	–	68.38	**70.42**	68.52	70.67	**70.71**
CNAE	56.30	–	**62.13**	62.13	56.30	62.13	**68.80**
Real ∅	67.43	–	70.24	**79.49**	76.76	78.43	**80.55**
O. ∅	70.16	–	**72.90**	68.50	76.71	78.86	**81.18**

to maintain the entire data stream. Therefrom we can fix the value of the constant b for each dataset experimentally reported in Table 3.4.

Tables 3.2, 3.3 and 3.4 present the results on all datasets. We compared SketchNB (SNB) and AdaSketchNB (AdSNB) classifiers to well-known state-of-the-art algorithms: the NB, the kNN with $k = 100$, the GMFactorSketch (GMS) [96] with default parameters $\epsilon = 0.01$, $\delta = 0.1$. We compared also to adaptive classifiers such as the Hoeffding Adaptive Tree (HAT) [44], and the adaptive NB (AdNB) with ADWIN. It turns out that GMS is useless and could not finish execution with some datasets which are marked by the cells with "–". We observe that the accuracy of SketchNB is almost the same when comparing to NB for all the datasets despite the use of probabilistic counts to estimate true counts. In comparison with kNN, we notice that SketchNB is more accurate on the whole set of datasets except the real ones. Such difference can be explained by the independence assumption between attributes of NB that, for some cases, does not hold true. Since GMS builds two sketch tables for each attribute, it is obvious that it will be more memory and time consuming because we are using only one big sketch. More than this, GMS cannot work with large datasets, e.g., Tweet$_1$ (see Figure 3.2).

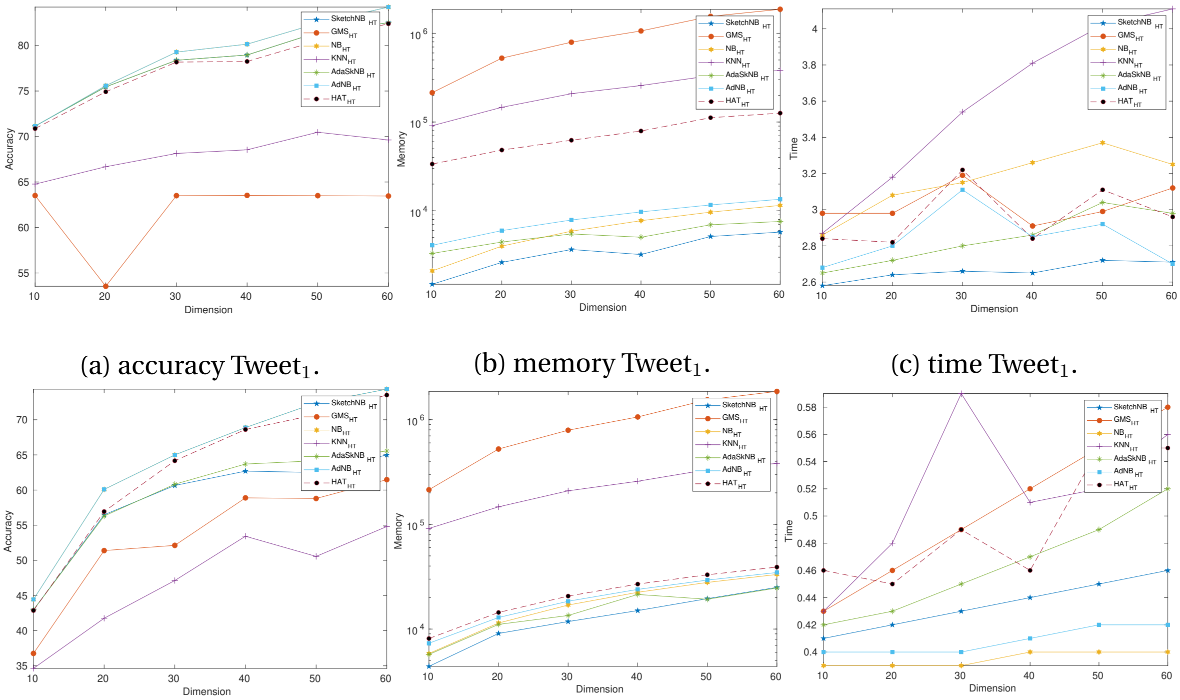
In order to simulate the proposed change detector classifier AdaSketchNB , we compare against the AdNB and HAT approaches. In Table 3.2, on overall average, we observe that AdaSketchNB achieves, practically, the same accuracy as AdNB whilst using a less amount

Table 3.3: Memory comparison of SNB, GMS, NB, kNN, ASNB, AdNB, and HAT. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive		
	SNB	GMS	NB	kNN	AdSNB	AdNB	HAT
SEA	**5,568**	49,312	**4,880**	58,280	7,832	**7,704**	2.48E6
RBF	18,824	226,072	**18,520**	100,920	**19,768**	20,688	3.61E6
LED	**23,568**	644,544	27,704	187,896	**22,792**	30,168	1.27E6
LED$_g$	**23,568**	644,544	27,704	187,896	**22,792**	30,528	623,856
AGR	19,360	201,000	**12,520**	92,760	**14,992**	15,344	3.85E6
HYP	20,624	225,952	**15,248**	100,824	**17,264**	17,568	925,600
Tweet$_1$	**569,880**	–	645,456	7.52E6	**582,144**	647,440	1.71E6
Tweet$_2$	**6.03E6**	–	6.55E6	7.5E7	**6.04E6**	6.39E6	1.23E7
Syn ∅	**839,598**	–	912,687	1.04E7	**840,600**	895,121	3.38E6
KDD99	**106,632**	1.32E6	123,496	303,200	108,456	61,088	**57,048**
Cover	**55,736**	1.71E6	60,488	377,832	61,440	46,344	**19,936**
Elec	14,776	178,464	**11,944**	88,712	14,560	**13,120**	42,432
Poker	**16,568**	223,952	19,800	98,760	18,608	**17,688**	278,528
Enron	**565,104**	–	688,752	7.13E6	534,264	690,400	**275,624**
IMDB	**1.41E6**	–	1.57E6	7.53E6	**1.37E6**	1.54E6	2.91E6
CNAE	**1.16E6**	–	1.58E6	1.58E6	1.23E6	1.58E6	**571,064**
Real ∅	**474,706**	–	578,263	2.44E6	**554,384**	563,117	593,306
O. ∅	**221,977**	–	270,343	6.69E6	**707,033**	740,186	2.08E6

Table 3.4: Time comparison of SNB, GMS, NB, kNN, ASNB, AdNB, and HAT. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive			b
	SNB	GMS	NB	kNN	AdSNB	AdNB	HAT	
SEA	1.75	2.08	**1.49**	10.91	2.87	**2.03**	7.75	675
RBF	6.62	9.65	**3.73**	31.2	9.6	**4.5**	17.67	7,191
LED	15.9	41.03	**5.67**	64.21	30.79	**7.28**	21.08	14,692
LED$_g$	15.93	43.45	**5.73**	68.83	29.23	**7.25**	23.09	14,692
AGR	4.09	5.94	**3.1**	26.05	5.47	**3.57**	9.12	3,678
HYP	4.51	6.42	**3.42**	27.79	4.11	**3.77**	9.8	5,448
Tweet$_1$	4.79	–	**4.17**	29.21	6.77	**2.89**	5.57	7,514,600
Tweet$_2$	73.44	–	**51.04**	385.23	91.43	**44.78**	65.93	6.48E8
Syn ∅	15.87	–	**9.79**	80.43	22.53	**9.51**	20	–
KDD99	40.02	90.75	**12.78**	51.66	64.42	**7.86**	18.11	15,074
Cover	18.92	50	**7.67**	64.42	32.74	**7.09**	19.55	59,675
Elec	0.33	0.42	**0.18**	1.06	0.35	**0.22**	0.67	4,555
Poker	6.87	12.3	**2.36**	22.03	11.67	11.67	**7.95**	4,859
Enron	0.98	–	**0.63**	6.07	1.29	**0.62**	0.97	15,340,000
IMDB	53.45	–	**27.07**	273.73	68.36	**32.89**	113.38	6,525,300
CNAE	1.38	–	**0.52**	0.67	2.26	**0.65**	1.71	7,231,800
Real ∅	17.41	–	**7.31**	59,95	25.8	**8.71**	23.2	–
O. ∅	16.54	–	**8.63**	70.87	24.06	**9.14**	21.49	–



(a) accuracy Tweet$_1$.

(b) memory Tweet$_1$.

(c) time Tweet$_1$.

(d) accuracy CNAE.

(e) memory CNAE.

(f) time CNAE.

Figure 3.4: Sorted plots of accuracy, memory and time over different output dimensions

of resources (see Table 3.3). In comparison with the HAT, the gain in memory exceeds the slight loss in accuracy. This is due to the use of, in addition to the sketch, the ADWIN change detector and estimator [21]. To assess the benefits in terms of resources usage, we observe the behavior of memory results is similar to the behavior of time results, i.e., when the memory usage increases, the running time increases accordingly. For some datasets, NB is more space-efficient than SketchNB (especially for datasets with a low number of attributes and classes, e.g., SEA and AGR) which is quite natural as simpler learners usually require less time for training and prediction. With large datasets (in terms of the number of attributes and classes), for instance Tweet$_1$ and Tweet$_2$, SketchNB and AdaSketchNB approaches consume fewer resources than the NB and AdNB respectively thanks to the CMS space-saving structure. In comparison with GMS, kNN, and HAT, the proposed algorithms are more space and time efficient.

Despite the gains exhibited with SketchNB and AdaSketchNB classifiers over different datasets, the results are still not entirely satisfying. Therefore, we proposed SketchNB$_{HT}$ (SNB$_{HT}$) and AdaSketchNB$_{HT}$ (ASNB$_{HT}$), a third contribution relying on pre-processing internally instances of SketchNB and AdaSketchNB , coupled with the hashing trick [20].

Table 3.5: Accuracy comparison of SNB$_{HT}$, GMS$_{HT}$, NB$_{HT}$, kNN$_{HT}$, ASNB$_{HT}$, AdNB$_{HT}$, and HAT$_{HT}$. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive		
	SNB$_{HT}$	GMS$_{HT}$	NB$_{HT}$	kNN_{HT}	ASNB$_{HT}$	AdNB$_{HT}$	HAT$_{HT}$
Tweet$_1$	77.99	61.84	**78.80**	68.04	77.99	**78.80**	77.53
Tweet$_2$	79.25	74.76	**79.66**	75.27	79.25	**79.66**	78.96
Enron	70.22	–	71.69	**89.54**	**83.14**	76.11	–
IMDB	68.40	–	69.98	**70.27**	69.73	69.55	**70.44**
CNAE	58.38	53.24	**64.20**	47.05	58.92	**64.20**	62.81
$O\varnothing$	70.85	–	**72.87**	70.03	**73.80**	73.67	–

Table 3.6: Memory comparison of SNB$_{HT}$, GMS$_{HT}$, NB$_{HT}$, kNN$_{HT}$, ASNB$_{HT}$, AdNB$_{HT}$, and HAT$_{HT}$. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive		
	SNB$_{HT}$	GMS$_{HT}$	NB$_{HT}$	kNN_{HT}	ASNB$_{HT}$	AdNB$_{HT}$	HAT$_{HT}$
Tweet$_1$	**3,648**	1,001,109	6,817	236,328	**5,472**	8,801	76,896
Tweet$_2$	**3,721**	1,001,109	6,817	236,328	**5,545**	8,801	62,404
Enron	**3,619**	–	7,760	265,373	**4,381**	9,341	–
IMDB	**4,528**	–	6,817	236,288	**5,580**	8,157	93,113
CNAE	**14,133**	1,001,165	19,623	236,848	**15,935**	21,103	23,689
$O\varnothing$	**5,930**	–	9,567	242,233	**7,382**	11,241	–

Table 3.7: Time comparison of SNB$_{HT}$, GMS$_{HT}$, NB$_{HT}$, kNN$_{HT}$, ASNB$_{HT}$, AdNB$_{HT}$, and HAT$_{HT}$. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive		
	SNB$_{HT}$	GMS$_{HT}$	NB$_{HT}$	kNN_{HT}	ASNB$_{HT}$	AdNB$_{HT}$	HAT$_{HT}$
Tweet$_1$	**2.66**	3.03	3.16	3.59	**2.84**	2.84	2.97
Tweet$_2$	**29.72**	43.06	43.48	37.93	**29.63**	33.28	34.11
Enron	**0.42**	–	0.47	0.61	0.44	**0.41**	–
IMDB	14.59	–	**13.54**	25.57	14.38	**13.58**	14.75
CNAE	0.44	0.51	**0.40**	0.52	0.46	**0.41**	0.49
$O\varnothing$	**9.55**	–	12.2	13.64	9.56	**9.51**	–

Figure 3.4 presents some detailed results of these experiments with two datasets, where each one is processed with 6 different output dimensions. We report in Tables 3.5, 3.6, and 3.7 the overall average accuracy, memory and running time over the different values of dimension for each dataset. For all datasets with different space dimension, the SketchNB$_{HT}$ approach's accuracy is similar to NB using a feasible amount of resources. For instance, Figure 3.4(a) depicts the ability of SketchNB$_{HT}$ and AdaSketchNB$_{HT}$ to achieve similar accuracy to the NB and AdNB on Tweet$_1$, and in Figure 3.4(b) and 3.4(c) we can see that they lead to large memory savings in lower time processing. The same behaviour is showed in Figures 3.4 (d), (e), and (f) using the CNAE dataset. Tables 3.5, 3.6, and 3.7) show also that SketchNB$_{HT}$ outperforms also GMS$_{HT}$ and kNN$_{HT}$ (excepts for accuracy with this latter). Nevertheless, the gain in memory and time is more interesting. Also with the adaptive classifiers, AdaSketchNB$_{HT}$ is more accurate than HAT$_{HT}$ for some datasets and uses much less memory.

3.5 Conclusion

In this chapter, we presented SketchNB, AdaSketchNB , SketchNB$_{HT}$, and AdaSketchNB$_{HT}$ approaches, new classifiers to handle evolving data streams and answer our main research questions (**Q1**, **Q2**, **Q3**) discussed in Chapter 1. The SketchNB algorithm extends the naive Bayes classifier using the CMS to reduce the memory needed. Then, we proposed AdaSketchNB , an adaptive version of SketchNB to handle concept drift. Finally, we coupled SketchNB and its adaptive version with the hashing trick technique for further gain with high-dimensional data to obtain SketchNB$_{HT}$ and AdaSketchNB$_{HT}$. We explained the learning process of these classifiers using sketches showing strong theoretical guarantees.

We compared these classifiers in an extensive evaluation with well-known classifiers showing that GMS, NB, HAT, and kNN methods are outperformed in memory by SketchNB and AdaSketchNB with large datasets. We showed also that using the HT and CMS techniques, the proposed SketchNB$_{HT}$ and AdaSketchNB$_{HT}$ obtain good results in terms of classification performance (accuracy, memory and time) when compared to other state-of-the-art classifiers.

In the following chapter, we aim to improve the performance of the kNN algorithm that proved to be costly in terms of resources with evolving data streams by using an efficient DR technique.

4

Compressed k-Nearest Neighbors Classification

Contents

This chapter contains results from a collaboration [108] with Rodrigo F. de Mello[1] and Nikolaos Tziortziotis[2] published at the European Conference on Artificial Intelligence (ECAI) 2020 under the title "*Compressed k-Nearest Neighbors Ensembles for Evolving Data Streams*".

[1] University of São Paulo, São Carlos, Brazil.
[2] Tradelab, Paris, France

4.1 Introduction

In the previous chapter, we noticed that the kNN is a resource-consuming algorithm, especially with this high-dimensional data streams, because it maintains a window of the most recent instances from the stream (Section 2.3.2). In practice, mining tasks reduce time and space requirements while only processing combinations of relevant features out of those redundant streams, what corresponds to data summaries generally obtained using a DR technique.

The dimensionality reduction process inherently arises when one deals with a large number of attributes, especially when data are sparse, in order to reduce the use of computational resources, more precisely the memory and processing time.

To address this issue, we apply *Compressed Sensing* (CS) which is a feature extraction strategy that provides theoretical lower and upper bounds on pairwise data transformation (Section 2.4.2). This data reduction is highly relevant in the context of data stream mining since it helps to reduce resource demands while ensuring the quality of learning (e.g., classification accuracy) and addressing the stream setting requirements. In this chapter, our main focus consists in increasing the time and memory efficiency of the kNN algorithm by compressing the input stream using an efficient DR technique (**Q2**), which allows us to ensure good theoretical bounds in between the original and the adapted stream, and therefore guaranteeing some close approximation to the accuracy that would be obtained using the original stream (**Q1**).

The plan of this work is organized as follows. We present the background of this work in Section 4.2. Afterwards, we provide the basics of the CS technique, followed by its application in conjunction with the kNN and the ensemble-based methods for evolving data streams in Section 4.3. Section 4.5 discusses the experimental results performed on both synthetic and real datasets that show the efficiency of our proposals. Section 4.6 concludes this chapter.

4.2 Preliminaries

Because of the potentially infinite nature of evolving data streams and the additional cost of high-dimensional data, classification algorithms, such as the kNN, suffer from resource issues. One way to cope with this issue is to couple the classifier with an efficient DR technique. The latter comprises the process of finding some transformation $A : \mathbb{R}^a \to \mathbb{R}^m$, where $m \ll a$, to be applied on each instance X_i from the stream S (see Equation(2.4)).

As we presented in Section 2.4.2, random projection matrices have been used in conjunction with CS [109]. This approach applies a random linear transformation on vectors, changing their original space and leading to significant results, outperforming PCA. For example, given data in a 10^4 dimensional space, two RPs will give a perfect recovery while two PCA projections will only recover data with a probability equals to $2/10^4$. In short, RP achieves good performance with few projections whereas the PCA performance increases linearly with the number of output dimension which makes it slower [109].

CS is a technique that has attracted a lot of attention and is based on the concept that a data compression method has to deal with redundancy while transforming and reconstructing data [23]. The basic idea is to use orthogonal features, i.e. complementary features, to provably and properly represent data as well as reconstruct them from small number of samples (measurements). CS relies on the principles that provide, with high probability, a good data reconstruction from a limited number of incoherent and possibly noisy measurements. Mathematically, the decompression of the data that obeys the linear relation in Equation (2.4) consists in approximating the error by ℓ_1-norm minimization that provides a convex relaxation and when data are sparse, the recovery via ℓ_1-minimization is provably exact [110]:

$$\arg\min_{x\in\mathbb{R}^d} \|x\|_1 \quad \text{s.t.} \quad y = Ax.$$

The goal is to find an efficient representation for each instance such that the sum of their reconstruction errors is minimized. The restricted isometry property (RIP) guarantees the proper computation of the above-mentioned recovery problem [111].

Hence, we aim to find the best tradeoff over three aspects (defined in Section 2.5): (i) the classifier accuracy; (ii) the memory usage; and (iii) the overall processing time.

All such aspects are strongly related: the drastic reduction of time and space complexities would make our approach much faster, but one should weigh in the classification performance – more precisely, accuracy – in the equation.

4.2.1 Construction of Sensing Matrices

Two related properties have been pointed out for the characterization of the sensing – or sampling – matrix: the sparsity and the RIP. The latter is both a necessary and a sufficient condition for an efficient data recovery. Randomization is a key ingredient in the construction of most of the RIP matrices used in the CS transformation process [112]. In what follows, we cite examples of matrices that have been used in the CS transformation process:

- *Fourier matrix* is obtained by applying Fourier transform on data and thereafter selecting uniformly at random p rows from a d dimensional Fourier matrix. However, this transformation requires to maintain the entire dataset;

- *Random Gaussian matrix* is generated randomly from a Gaussian distribution having independent and identically distributed (i.i.d) entries with zero mean and variance one: $A_{i,j} \sim \mathcal{N}(0, 1)$;

- *Random Bernoulli matrix* has entries which are randomly sampled from a Bernoulli distribution with equal probability: $A_{i,j} \in \{1/\sqrt{p}, -1/\sqrt{p}\}$.

For the data-independent random matrices, it has been proved that any matrix A satisfying the Johnson-Lindenstrauss Lemma (2.3) will also satisfy the RIP in CS with high probability if $m = \mathcal{O}(s \log(a))$ [85] which answers intuitively the question: "how many trials do we require to collect s different orthogonal and *most interesting* dimensions from a possibilities?". A comparison of the results obtained with these matrices and an explanation of the choice of the matrix used in this work are provided in the following.

4.3 Compressed Classification Using kNN Algorithm

The kNN is one of the most often used algorithms that has been adapted to the stream setting [113]. The prediction of the class label for an instance is made by taking the majority vote of its nearest neighbors inside a fixed size sliding window[3], using a defined distance metric. Since we are keeping instances in a window, a DR is imperative to avoid the curse of dimensionality, when dealing with high-dimensional data that could increase the use of computational resources during prediction.

The main idea to mitigate this drawback and improve kNN's performance is to use a simple strategy which has the desired properties – such as CS.

We focus on the analysis of an infinite stream of instances $X_i \in \mathbb{R}^a$ from which we wish to construct a low dimensional space $\mathbb{R}^m$, where $m \ll a$. We assume that instances are s-sparse in some basis, so we can use the CS with a RIP matrix and work in a lower dimension of $\mathcal{O}(s \log(a))$. It is important to perform such reduction because it is related to the number of dimensions and independent of the stream size, making it useful in applications for data streams where the size is unknown. This transformation can lead to information loss (by removing important attributes), except if the sensing matrix respects the RIP, then with high probability, the information loss is minimal and the original data can be recovered. Figure 4.1 presents the main flow of the proposed approach combining the simplicity of kNN and the strong properties of CS to obtain the compressed kNN classifier, called CS-kNN in the following.

Fundamentally, CS is composed of two phases: (i) the *compression* phase, where the data are projected onto a low-dimensional space; and (ii) the *decompression* phase, where the data are recovered. Nevertheless, the compressed nature of CS makes the paradigm a better fit to classification than the reconstruction. In this work, we are only concerned with the first stage, so the extracted features from the high-dimensional space are fed to kNN classifier which predicts target class labels. This does not, however, prevent the guarantees over the recovery to hold true.

[3] Streaming kNN is therefore adaptive and handles the question **Q3** presented in Chapter 1.

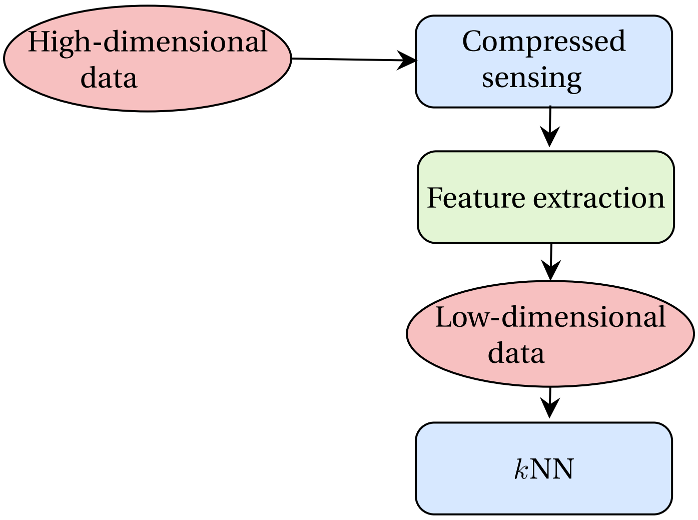


Figure 4.1: Compressed kNN Scheme.

Algorithm 4 Compressed kNN algorithm. **Symbols:** $S = \{X_1, X_2, \ldots\}$: data stream; $C = \{c_1, c_2, \ldots\}$: set of class labels; W: sliding window; k: the number of neighbors; m: the target dimension; B: subset of W.

```
 1: function CS-$k$NN($S, w, k, m$)
 2:     Init $w \leftarrow \emptyset$
 3:     for all $X_i \in S$ do
 4:         $Y_i \leftarrow$ CS($X_i$)                    ▷ apply CS
 5:         for all $Y_j \in W$ do                        ▷ $\forall\, j \neq i$
 6:             compute $D_{Y_j}(Y_i)$                    ▷ Equation (4.2)
 7:         end for
 8:         $c \leftarrow \max_{c \in C} D_{B,k}(Y_i)$     ▷ Equation (4.3)
 9:         $w \leftarrow Y_i$                            ▷ maintain the compressed $X_i$ in $W$
10:     end for
11: end function
```

Algorithm 4 shows the pseudo-code of the CS-kNN. Given an infinite data stream S and the window size W, we apply CS on each instance X_i of the stream (lines 3-4), then we apply kNN by computing the distance of each Y_j in W with Y_i (line 6) and thus report the most frequent class label to Y_i (line 8). Finally, we feed the compressed version Y_i to W (line 9).

In this work, we opt to make kNN more efficient in terms of memory and speed taking into account the online aspect of evolving data streams. Our approach consists of the CS application on high-dimensional data obtained by compressing every new arrived instance via solving Equation (2.4). So, we need to use an effective sampling matrix that gives sufficiently good (or with minor loss in) accuracy and eventually leads to computational savings. In a recent work [114], authors reviewed different sampling matrices performance where the experiment results show that Gaussian random matrices perform nicely.

Table 4.1: Accuracy (%) comparison of compressed sensing with Bernoulli, Gaussian, Fourier matrices, and the entire dataset. Bold values indicate the best results.

Dataset	Bernoulli	Gaussian	Fourier	Whole data
Tweet$_1$	64.78	**77.90**	**77.90**	79.80
Tweet$_2$	64.59	77.17	**79.53**	79.20
Tweet$_3$	60.24	75.59	**77.13**	78.86
CNAE	27.04	**64.59**	58.49	73.33
Enron	95.88	**95.97**	95.91	96.18
Overall ∅	62.51	**78.24**	77.79	81.58

To motivate our choice in the following, we perform experiments to assess different sampling matrices. For this, we first generate synthetic random Bernoulli and Gaussian matrices, and we also construct the Fourier matrices on some datasets (see the description in Table 4.2). For each dataset, we build projections for 5 different settings of the target dimension $\{10, 20, 30, 40, 50\}$. Table 4.1 shows the results for kNN (with $k = 5$) along with the overall average over the different targeting dimensions for each matrix. We notice that with the random Bernoulli matrix, kNN performs worse on average, confirming previous studies [114, 115], compared to the Random Gaussian and Fourier matrices which are very close to the kNN, in terms of accuracy, using the whole data without projections. Nevertheless, Fourier transformation relies on data, i.e., it requires the presence of all instances which is unrealistic in the context of data streams. In [116], authors proposed a recursive scheme for Fourier matrix with data streams which constructs successive windows and uses the measurement in the previous window to obtain the next one. However, this approach is expensive in terms of memory since it keeps data on windows and it is still not as accurate as using a Gaussian matrix.

In this work, we want to use a data-independent matrix to ensure fast processing. The experiments above suggest that we should focus on the Gaussian matrix which not only provides good accuracy but also satisfies with high probability the RIP and therefore allows the recovery of instances [117].

We take the sampling matrix to be such that its elements are drawn independently from a Gaussian distribution, setting common to various CS problems [23]. The matrix A in Equation (2.4) satisfies the RIP, so X can be recovered with minimum error from Y, i.e., Y preserves the important information that X contains.

First, we need to bound the probability of error related to the estimated instance and its expected value using the Hoeffding inequality. Given $X_i, \forall i \in [1, N]$, where x_i^j are bounded by the interval $[p_j, b_j]$, then for any ϵ, the probability of error is upper bounded as follows:

$$P(|X_i - \hat{X}_i| \geq \epsilon) \leq 2 \exp\left(-\frac{2\epsilon^2}{\sum_{l=1}^{a}(b_l - p_l)^2}\right), \tag{4.1}$$

where $\hat{X}_i$ is the reconstructed instance and N can simply be the size of a "sliding window" over which the error guarantee is provided.

For applying the DR on instances from the original space, a good technique aims to preserves the structure of the projected data, i.e., keep close instances together and dissimilar instances far away as represented in the input space. The challenge is then to show that the neighborhood of a given instance is preserved after the CS projection to ensure a good classification performance of the kNN algorithm.

4.3.1 Theoretical Insights

To make the link between the kNN and the use of RIP matrices, we point out that the JL Lemma 2.4.2 [80, 81] asserts that a random projection preserves the distances between pairs of instances up to $1 \pm \epsilon$ guarantee with high probability. Similarly, the kNN algorithm is based on a function that measures the distances between instances to predict. Thus, we aim to provide theoretical guarantees on the connection between kNN and stream recovery by showing that the CS transformation using Gaussian matrix preserves the distance function and also approximately maintains the shape – in the neighborhood sense – as the original space, based on the concept of *persistent homology*.

Persistent homology [118] is one of the main tools used to extract information from topological features of a space at different scales for an effective shape description. Given a dataset in some metric space, computing the persistent homology naturally involves nearest neighbors since we are constructing the topological space by building open balls around instances. In this regards, it has been shown in [119] that the persistent homology of a distance such as in the JL Lemma (2.3) is $(1 \pm \epsilon)$-preserved under random projection into $m = \log N/\epsilon^2$ dimensions. The basic idea in [119] consists in preserving the radius of the minimum enclosing open ball of data up to a factor of $(1 \pm 4\epsilon)$.

In the following, we deal with the Euclidean distance function in both kNN and data reconstruction guarantees. Given a window W, the distance between instances X_i and X_j is defined as follows:

$$D_{X_j}(X_i) = \sqrt{\|X_i - X_j\|^2}. \tag{4.2}$$

Similarly, the k-nearest neighbors distance is defined as follows:

$$D_{W,k}(X_i) = \min_{\binom{W}{k}, X_j \in W} \sum_{j=1}^{k} D_{X_j}(X_i), \tag{4.3}$$

where $\binom{W}{k}$ denotes the subset of W of size k, i.e., the k-nearest neighbors to the instance x_i in W.

CS random matrices satisfy RIP, so we need to show that our matrix preserves the neighborhood for kNN without significant loss through the JL Lemma (2.3). This would allow us to conserve distances among instances and finally ensure distance-preservation among all neighbors. In [117], Baraniuk, Davenport, DeVore, and Wakin have indeed established a connection between the expressions in (2.3) and (2.5), and proved that the JL lemma implies the RIP for s-sparse data within an ϵ-multiplicative factor. A converse result has been proved in [120] wherein matrices having the RIP respect the JL lemma, i.e., preserve the distances in the transformation between any pairs of instances up to a $(1 \pm \epsilon)$-factor with target dimension in $\mathcal{O}(s \log(a))$, where a is the input dimensionality.

4.3.2 Application to Persistent Homology

Now we want to prove, based on the aforementioned result [120] derived from Equation (2.5), that CS preserves as well the distances between all instances up to $(1 \pm \epsilon)$-error and not only distances between pairs of instances. In other words, we prove that, given a RIP matrix, the resulting compressed instances preserve the kNN neighborhood of the data.

Theorem 4.1 *Given a set of instances in a sliding window $W = \{X_i\}$, $i \in [1, N]$ and $\epsilon \in [0, 1]$, if there exists a transformation matrix $A : \mathbb{R}^a \to \mathbb{R}^m$ having the RIP, such that $m = \mathcal{O}(s \log(a))$, where s is the sparsity of data, then $\forall X_i \in W$:*

$$(1 - \epsilon) D^2_{W,k}(X) \leq D^2_{W,k}(AX) \leq (1 + \epsilon) D^2_{W,k}(X). \tag{4.4}$$

Proof. Assume that $X_1, X_2, \cdots, X_k$ are the k-nearest neighbors to an instance $t \in W$. We have:

$$(1 - \epsilon)\|t - X_i\|^2 \leq \|At - AX_i\|^2 \leq (1 + \epsilon)\|t - X_i\|^2.$$

By summing these inequalities k times, we obtain:

$$(1 - \epsilon) \sum_{i=1}^{k} \|t - X_i\|^2 \leq \sum_{i=1}^{k} \|At - AX_i\|^2 \leq (1 + \epsilon) \sum_{i=1}^{k} \|t - X_i\|^2.$$

The distance of At to its k-nearest neighbors in W is minimal, so we have the lower bound as follows:

$$D^2_{W,k}(At) \leq \sum_{i=1}^{k} \|At - AX_i\|^2.$$

For the upper bound, we have:

$$D^2_{W,k}(At) \leq \sum_{i=1}^{k} \|At - AX_i\|^2 \leq (1 + \epsilon) \sum_{i=1}^{k} \|t - X_i\|^2,$$

$$D^2_{W,k}(At) \leq \sum_{i=1}^{k} \|At - AX_i\|^2 \leq (1 + \epsilon) D^2_{W,k}(t).$$

Assume that $Az_1, Az_2, \cdots, Az_k$ are the k-nearest neighbors to At, where $z_1, z_2, \cdots, z_k \in W$. So we have:

$$(1-\epsilon)\sum_{i=1}^{k}\|Y - z_i\| \leq \sum_{i=1}^{k}\|At - Az_i\|^2 = D^2_{W,k}(At).$$

Given the fact that $X_1, X_2, \cdots, X_k$ are the k-nearest neighbors to t, we found the lower bound as follows:

$$D^2_{W,k}(t) = \sum_{i=1}^{k}\|t - X_i\|^2 \leq \sum_{i=1}^{k}\|t - z_i\|^2.$$

This completes the proof. □

We demonstrated that the CS-kNN has desirable geometrical properties: by achieving homology preservation while being scale-invariant in terms of distances, it captures the neighborhood up to some (ϵ)-divergence between the original and the compressed instances.

4.4 Compressed kNN Ensembles

We propose another application in this framework that consists in using the CS technique with an ensemble-based method which applies CS-kNN as a base learner under the Leveraging Bagging (LB) [24], denoted CS-kNN^{LB}. To increase the diversity inside the LB ensemble method, in addition to sampling with the Poisson distribution (λ), with $\lambda \geq 1$, we can use several random matrices by generating a different CS matrix for each ensemble member (CS-kNN) instead of using only one random matrix for all the learners (the case of CS-kNN^{LB}). We refer to the aforementioned approach in the following as Compressed Sensing Bagging Ensemble (CSB), (CSB-kNN). The properties assessing the neighborhood preservation proved for CS-kNN, hold also for the ensemble-based methods that uses CS-kNN as a base learner.

4.5 Experimental Evaluation

In this section, we assess the impact of feature transformations on kNN and the ensemble methods for data streams. In order to thoroughly evaluate our proposals, we conduct extensive experiments using several datasets.

4.5.1 Datasets

We use 4 synthetic and 5 real-world datasets from a variety of domains. Table 4.2 presents a short description of each dataset, and further details are provided in what follows.
Tweets. Tweets was created using the text data generator provided by MOA [97]. It simulates sentiment analysis on tweets, where messages can be classified into two categories

Table 4.2: Overview of the datasets.

Dataset	#Instances	#Attributes	#Classes	Type
Tweets$_1$	1,000,000	500	2	Synthetic
Tweets$_2$	1,000,000	1,000	2	Synthetic
Tweets$_3$	1,000,000	1,500	2	Synthetic
RBF	1,000,000	200	10	Synthetic
CNAE	1,080	856	9	Real
Enron	1,702	1,000	2	Real
IMDB	120,919	1,001	2	Real
Spam	9,324	39,916	2	Real
Covt	581,012	54	7	Real

depending on whether they convey positive or negative feelings. Tweets$_1$, Tweets$_2$, and Tweets$_3$ produce instances of 500, $1,000$, and $1,500$ attributes, respectively.

RBF. The Radial Basis Function generator creates centroids at random positions, and each one has a standard deviation, a weight and a class label.

CNAE. CNAE is the national classification of economic activities dataset, initially used in [105]. Instances represent descriptions of Brazilian companies categorized into 9 classes. The original texts were pre-processed to obtain the current highly sparse dataset.

Enron. The Enron corpus is a cleaned version of a large set of emails that was made public during the legal investigation concerning the Enron corporation [103].

IMDB. IMDB[4] movie reviews dataset was first proposed for sentiment analysis [104], where reviews have been pre-processed, and each review is encoded as a sequence of word indexes (integers).

Spam. The spam corpus is the result of a text mining on an online news dissemination system which intends on creating an incremental filtering of e-mails classifying them as spam or not [121]. Each attribute represents the presence of a word in the instance (an e-mail).

Covt. The forest covertype dataset obtained from US Forest Service Region 2 Resource Information System (RIS) data.

4.5.2 Results and Discussions

The experiments were implemented and evaluated in Java by extending the MOA framework [37, 97]. We used the online evaluation setting for Test-Then-Train method [91], where each instance is used first for testing and then for training.

Table 4.3 presents the results for distinct sizes of W and shows that; for shorter windows ($W = 100$), the accuracy degrades, while for bigger windows the accuracy slightly increases.

[4] http://waikato.github.io/meka/datasets/.

Table 4.3: Performance of kNN with different window sizes.

Accuracy (%)

	W=100	W=1000	W=5000
Overall ∅	75.48	80.33	82.44

Time (sec)

	W=100	W=1000	W=5000
Overall ∅	537.76	6592.72	20028.01

Memory (MB)

	W=100	W=1000	W=5000
Overall ∅	46.85	269.65	2000

On the other hand, the processing time and memory usage increase as well. Therefore this parameter selection implies an accuracy-time-memory tradeoff. The following experiments are performed with $W = 1000$ for kNN, because using a greater window size yields indeed to a better accuracy but the resource consumption is more significant.

Non-Ensemble Methods

For fair comparison of the the performance of our proposed classifier, CS-kNN, we use commonly-used techniques in the literature coupled with kNN as well; self-adjusting memory kNN[5] with the CS technique (CS-sam$k$NN), kNN using the hashing trick (HT-kNN), principal component analysis (PCA-kNN), and the standard kNN without projection as well (using the entire data). The streaming kNN has two principal parameters: the number of neighbors k and the window size W.

Tables 4.4, 4.5, and 4.6 report the final accuracies, memory consumption, and speed of the classification task in a 40-dimensional space after the projections, based on two setups of $k = 5, 11$. We choose 40 dimensions because we noticed that, starting from this size of space, improvements are statistically insignificant as showed in Figure 4.2. The latter illustrates a detailed comparison with five different values of output dimension $(10, 20, \cdots, 50)$ on Tweet$_2$ and Enron datasets.

We notice that our proposed CS-kNN approach has more accurate results (Table 4.4) than the HT-kNN for all datasets and it is slightly outperformed by the CS-sam$k$NN, the standard kNN (without projection) and PCA-kNN; this quite a natural result since kNN processes the whole data stream and PCA-kNN formally tries to find a lower-dimensional space under which the sum of square distances – representing the error, between the original data and its projection – is minimized. The CS-kNN is moderately less accurate than CS-

[5] Best paper award at ICDM 2016.

Table 4.4: Accuracy comparison of CS-kNN, CS-sam$k$NN, HT-kNN, PCA-kNN, and kNN over the whole dataset.

Dataset	CS-kNN		CS-sam$k$NN		HT-kNN		PCA-kNN		kNN	
	$k=5$	$k=11$	$k=5$	$k=11$	$k=5$	$k=11$	$k=5$	$k=11$	$k=5$	$k=11$
Tweet$_1$	78.82	78.88	76.02	74.31	73.77	73.14	80.43	79.43	79.80	78.17
Tweet$_2$	78.13	78.36	75.74	74.13	73.02	72.61	80.06	78.89	79.20	77.74
Tweet$_3$	76.75	76.16	73.03	72.56	72.40	72.36	81.93	82.38	78.86	77.73
RBF	98.90	97.31	99.87	99.78	19.20	19.20	99.00	97.86	98.89	97.33
CNAE	70.00	68.70	73.77	72.19	65.00	65.28	75.83	72.08	73.33	71.48
Enron	96.02	95.65	96.23	96.06	95.76	95.48	94.59	93.18	96.18	96.00
IMDB	69.86	72.32	74.29	74.53	69.65	72.03	70.57	72.81	70.94	72.51
Spam	85.39	81.01	91.34	90.48	83.82	80.63	96.00	94.66	81.17	77.32
Covt	91.36	89.92	90.47	87.71	77.18	76.59	91.55	90.16	91.67	90.30
Overall $\varnothing$	82.80	82.04	83.42	82.42	69.98	69.70	85.55	84.61	83.34	82.06

Table 4.5: Time comparison of CS-kNN, CS-sam$k$NN, HT-kNN, PCA-kNN, and kNN over the whole dataset.

Dataset	CS-kNN		CS-sam$k$NN		HT-kNN		PCA-kNN		kNN	
	$k=5$	$k=11$	$k=5$	$k=11$	$k=5$	$k=11$	$k=5$	$k=11$	$k=5$	$k=11$
Tweet$_1$	62.55	91.06	41.81	59.20	93.24	99.78	622.65	629.60	1198	1432
Tweet$_2$	107.48	112.97	74.92	99.77	120.83	127.95	705.71	712.84	2029	2502
Tweet$_3$	126.73	142.95	83.01	101.43	154.22	165.11	988.25	995.93	2864	3643
RBF	59.47	80.52	60.08	77.00	168.31	169.88	243.26	258.12	284.34	439.23
CNAE	0.87	0.92	0.56	0.63	0.95	1.02	3.97	4.14	32.19	35.04
Enron	1.58	1.63	1.31	1.57	1.81	1.90	7.21	7.28	86.08	91.99
IMDB	95.62	120.66	80.82	103.51	125.62	129.27	1686	1692	7892	8217
Spam	159.92	183.19	197.22	208.94	194.07	216.37	11329	14820	34231	35031
Covt	30.94	51.08	39.25	45.55	88.17	90.85	161.00	164.16	252.69	268.28
Overall $\varnothing$	71.68	87.22	64.33	75.29	105.25	111.42	1749	2142	5430	5740

sam$k$NN for some datasets containing drifts, because the latter deals with different types of concept drift which makes it stronger facing changes in data distributions.

To assess the benefits in terms of computational resources–where small values are desirable– Tables 4.5 and 4.6 point out the improvements of CS-kNN in terms of memory and time against CS-sam$k$NN, PCA-kNN, and kNN which are significant enough to justify relatively minor losses in accuracy. In fact, the CS-sam$k$NN algorithm maintains models for current and past concepts which makes it memory inefficient. The PCA-kNN performs worse, in terms of resource usage, than RP since it incrementally stores and updates the eigenvectors and eigenvalues, confirming previous studies [109]. Our proposed approach is also faster than the HT-kNN, although they have similar memory behavior, because both are based on RP and do not rely on data. For some datasets such as Spam, the CS-kNN outperforms kNN (using the whole data) simply because finding relevant combinations of

Table 4.6: Memory comparison of CS-kNN, CS-sam$k$NN, HT-kNN, PCA-kNN, and kNN over the whole dataset.

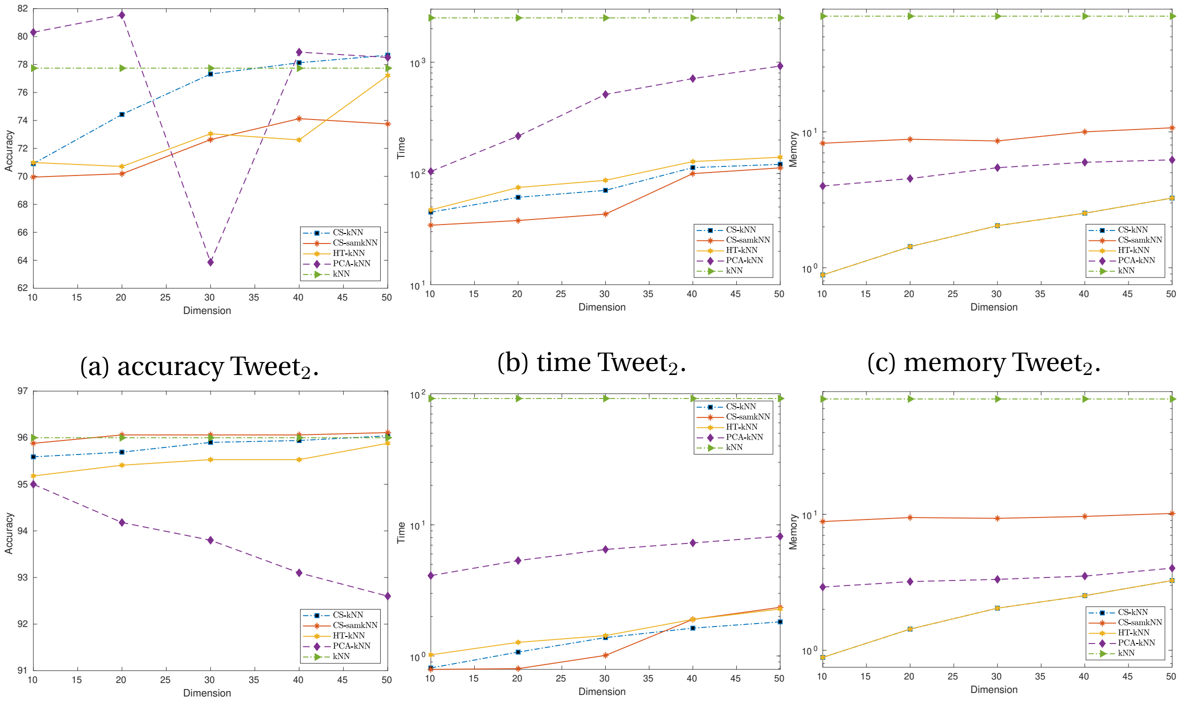
Dataset	CS-kNN	CS-sam$k$NN	HT-kNN	PCA-kNN	kNN
Tweet$_1$	2.52	8.86	2.52	3.03	34.64
Tweet$_2$	2.52	10.48	2.52	5.97	70.97
Tweet$_3$	2.52	10.52	2.52	8.84	103.19
RBF	2.52	10.31	2.52	8.86	13.18
CNAE	2.52	10.22	2.52	3.09	61.37
Enron	2.52	9.84	2.52	3.51	70.60
IMDB	2.52	10.28	2.52	8.81	70.65
Spam	2.52	10.57	2.52	245.22	1476.11
Covt	2.52	9.96	2.52	3.02	3.47
Overall ∅	2.52	10,12	2.52	32.26	211.57

existing features and presenting them in a different space can help supervised models to improve accuracy. Even if data are not sparse, CS surprisingly performs transformations on suitable bases.

Figures 4.2(a) and 4.2(d) depict the typical tradeoff for accuracy: a small feature space cannot properly represent data, therefore it can significantly degrade the accuracy; whereas a higher dimensional space (e.g., 50) increases the accuracy and makes it closer to the results with kNN. We also notice the stability of our CS-kNN, i.e., the accuracy is linearly boosted with the target space size and converges to the accuracy of kNN. On the other hand, CS-sam$k$NN, HT-kNN and PCA-kNN have different behaviors, clearly illustrated in Figure 4.2(a); this results deduce that, in practice, it may be hard to fix a proper space size. We also show that kNN, PCA-kNN and HT-kNN are outperformed in terms of processing time (Figures 4.2(b) and 4.2(e)) and that CS-kNN requires also less memory compared to these baselines. For instance, with Tweet2 and Enron in Figures 4.2(c) and 4.2(f) respectively, we observe large gains compared to kNN, PCA-kNN, and CS-sam$k$NN algorithms, albeit our proposal has the same memory usage as the HT-kNN because both do not rely on data. We also observe that the behavior of memory usage is correlated to the running time trends, i.e., when the memory usage increases, the processing time also increases accordingly.

Ensemble Methods

We compare the proposed LB with CS-kNN as a base learner (CS-kNN^{LB}) and the CSB-kNN with a different CS matrix for each learner, both using 10 learners (the size of ensemble) and $k = 5$, against popular ensemble methods such as the adaptive random forest (ARF) [25] and leveraging bagging using Hoeffding tree [41] as base learner (HTree^{LB}), with 30 and 10 ensemble members, respectively. Tables 4.7, 4.8 and 4.9 display the performance of the ensembles. In this evaluation, each of the ensemble member uses the same CS matrix to



(a) accuracy Tweet$_2$.

(b) time Tweet$_2$.

(c) memory Tweet$_2$.

(d) accuracy Enron.

(e) time Enron.

(f) memory Enron.

Figure 4.2: Sorted plots of accuracy, time and memory over different output dimensions.

Table 4.7: Accuracy (%) comparison of CS-kNN^{LB} , CSB-kNN, CS-HTree^{LB}, and CS-ARF.

Dataset	CS-kNN^{LB}	CSB-kNN	CS-HTree^{LB}	CS-ARF
Tweets$_1$	78.94	81.80	81.35	81.53
Tweets$_2$	78.24	81.28	80.39	80.75
Tweets$_3$	76.06	80.40	78.59	79.54
RBF	98.90	99.68	99.24	99.25
CNAE	71.64	81.48	65.70	62.55
Enron	95.94	96.00	96.17	95.88
IMDB	70.02	74.27	74.80	74.88
Spam	86.08	90.28	90.02	89.04
Covt	91.09	91.76	88.48	88.01
Overall ∅	82.99	86.33	83.86	83.49

Table 4.8: Time (sec) comparison of CS-kNN^{LB}, CSB-kNN, CS-HTree^{LB}, and CS-ARF.

Dataset	CS-kNN^{LB}	CSB-kNN	CS-HTree^{LB}	CS-ARF
Tweets$_1$	1130.44	1251.77	82.18	170.52
Tweets$_2$	1449.44	1526.30	105.87	212.69
Tweets$_3$	1668.26	1825.41	127.19	239.97
RBF	735.21	772.62	90.22	223.08
CNAE	8.99	11.02	1.80	4.66
Enron	20.07	21.92	2.11	3.78
IMDB	1552.81	1649.94	90.17	174.54
Spam	359.07	2194.93	218.16	270.15
Covt	612.62	694.02	41.69	115.3
Overall ∅	837.43	1105.33	84.37	108.70

Table 4.9: Memory (MB) comparison of CS-kNN^{LB}, CSB-kNN, CS-HTree^{LB}, and CS-ARF.

Dataset	CS-kNN^{LB}	CSB-kNN	CS-HTree^{LB}	CS-ARF
Tweets$_1$	6.16	27.13	60.71	175.71
Tweets$_2$	6.16	28.97	66.75	177.32
Tweets$_3$	6.16	30.80	73.89	176.92
RBF	6.16	25.96	9.91	25.90
CNAE	6.15	28.11	0.48	1.31
Enron	6.15	28.59	1.59	4.10
IMDB	6.16	28.60	5.60	18.63
Spam	5.38	151.91	5.15	10.44
Covt	6.16	24.10	4.44	11.66
Overall ∅	6.07	41.57	25.39	66.89

perform the reduction into 40 dimensions, except CSB-kNN which sets up a different matrix for each member in attempt to assess the ensemble diversity impact.

Tables 4.7, 4.8 and 4.9 show, using only 10 learners, CSB-kNN performs better than the reputed CS-ARF [25] using 30 learners (trees) on most of the datasets. We noticed that with CSB-kNN, when the features set is large (e.g. Spam), the memory usage is relatively high. On the other hand, for large datasets (e.g. Tweets), the CS-ARF and CS-HTree^{LB} require more memory whereas our approaches use less, what makes them useful for the stream setting. The CS-kNN^{LB} ensemble method is the most memory efficient and even proved competitive with CS-HTree^{LB} and CS-ARF. However, this is at the price of being slower. Also, computational resources of CSB-kNN with different CS matrices increase considerably for the sake of accuracy and diversity (in order for the ensemble to generalize well).

In conclusion, our CSB-kNN ensemble method has good overall performance compared to competitors. We showed that our proposal can be used to classify accurately data streams with a large number of attributes using a relatively small number of base learners, in contrast with CS-ARF where more–or less– base trees can considerably affect the classification performance.

4.6 Conclusion

In this chapter, we presented a scheme to enable the sliding window kNN algorithm (**Q3**) to be efficient with evolving high-dimensional data streams, in terms of classification performance and computational resources (memory and time) (**Q1** and **Q2** handled), after space transformations provided by CS given its ability to ensure theoretical lower and upper bounds on pairwise data transformations. Our first contribution in this chapter is the mix of two main ingredients: CS and kNN, thus resulting in the CS-kNN algorithm designed to work on evolving data streams while operating on a reduced feature space. We proposed also an ensemble method, CSB-kNN, that uses CS-kNN as base learner under the LB, where each ensemble member has a different CS matrix to help increasing the overall accuracy. We showed theoretically that for the CS-kNN using Gaussian matrices, the neighborhood distance is preserved up to some $1 \pm \epsilon$-factor. The key idea is to show that squared kNN distances, in the original data, are too within the same factor. Consequently, our CS-kNN algorithm also conserves such distances.

We evaluated the proposed algorithms via extensive experiments using synthetic and real-world datasets with different parameters. Results show the potential of the CS-kNN and CSB-kNN algorithms to obtain close approximations to what would be obtained using the input instances from data streams.

The following chapter will explore the ARF ensemble method and enhance its performance using the CS technique.

5

Compressed Adaptive Random Forest Ensemble

Contents

This chapter contains results from a collaboration [122] with Heitor Murilo Gomes[1] published at the International Joint Conference on Neural Networks (IJCNN) 2020 under the title "*CS-ARF: Compressed Adaptive Random Forests for Evolving Data Stream Classification*".

5.1 Introduction

Streaming ensemble-based methods have become very popular thanks to their high predictive performance and the fact that they can be used – or tested – with new learners [48]. Nevertheless, other than their sensitivity to the learning algorithm used as a base learner, most of the existing stream ensemble methods are often expensive and time-consuming

[1] University of Waikato, Hamilton, New Zealand.

when dealing with sparse and high-dimensional data streams.

Despite their good classification performance, the major drawback of ensembles is the high computational cost exacerbating as the dimensionality of data increases.

In a recent work [25], the adaptive random forest method (ARF) was proposed to deal with evolving data streams by extending the random forest algorithm (RF) using a concept drift mechanism[2] to deal with changes in the distribution over time (Section 2.6). However, based on the results from the previous chapter, it appears that ARF is effective (in terms of accuracy) but inefficient (in terms of resource usage) with high-dimensional data streams.

In attempt to improve the performance of ARF, we propose the compressed adaptive random forest; an ensemble-based method that extends the ARF [25] to handle high-dimensional and sparse data streams. To do so, we follow the strategy used in the previous chapter by incorporating a DR technique, CS [23], to project the data into a lower-dimensional space by finding useful combinations of existing attributes (**Q1** and **Q2** handled). Therefore, instead of building trees using high-dimensional instances, we will use a smaller representation of these instances that will boost the efficiency of the ARF method.

The rest of the chapter is organized as follows. Section 5.2 discusses the details of research issues that motivated our study. In Section 5.3, we introduce the strategy adopted with our proposed method. Section 5.4 outlines and discusses the experimental evaluation. We finally draw concluding remarks in Section 5.5.

5.2 Motivation

One notable issue related to the ensemble-based methods with evolving data streams is the massive computational demand (in terms of memory and running time). Ensembles require more resources than single classifiers which become significantly worse with high-dimensional data streams, as mentioned in the previous chapter. To cope with this problem without importantly affecting the predictive performance of the ARF method, we need to incorporate an efficient DR technique that can internally and incrementally transform high-dimensional data into a lower space before using them for the learning task.

The feature extraction task plays a critical role when dealing with high-dimensional data and is often used in data mining and machine learning. This task consists on extracting a subset of relevant attributes (in low-dimensional space) from a set of input attributes in high-dimensional space [52]. This pre-processing step provides potential benefits to stream mining algorithms, such as reducing the storage usage, decreasing the processing time, and enhancing – or not losing much in – the prediction performance.

[2] The ARF method naturally handles concept drifts and embraces the question **Q3**.

In this context, we aim to use the CS technique [23] that deals with redundancy while transforming and reconstructing data. The basic idea is to use orthogonal attributes or samples, i.e. complementary attributes, to provably and properly represent data as well as reconstruct them from a small number of samples. More details about the basic notions of this technique are available in Section 2.4.2.

5.3 Compressed Adaptive Random Forest

The random forest [123] is a well known ensemble-based method that is widely used in the batch learning classification. It grows several trees while randomly selecting attributes at each split node from an entire set of input attributes. Nonetheless, this is inapplicable on evolving data streams because the random forest algorithm performs multiple passes to establish bootstraps which is inappropriate in the streaming framework. For this to happen, an adaptive random forest method [25] has been proposed to adapt random forest to work under the streaming setting. This adaptation includes the use of: (i) an online bootstrap process to approximate the original data explained in [25]; and (ii) a random subset of attributes to limit the size of input set during each leaf split. To cope with concept drifts, ARF method is coupled with a warning and drift detection operators to adapt to changes in the data distribution over time which will lead to a superior classification performance. As mentioned previously, the major drawback of the ensemble-based methods, and particularly the ARF method, is the important amount of computational resources needed to deal with high-dimensional data streams. To cope with this issue, we use an efficient technique with relevant properties, such as CS [23, 109].

In this vein, we propose our novel approach Compressed Adaptive Random Forest, denoted CS-ARF in the following, that combines the simplicity of the CS and the high learning performance of the reputed ARF method for evolving data streams. Given an infinite stream of high-dimensional instances $X \in \mathbb{R}^a$, we wish to construct a low-dimensional representation $Y \in \mathbb{R}^m$, where $m \ll a$ and Y is the dense representation of X after the application of the reduction using the CS projection.

We assume that all the instances X in the stream S are s-sparse to adhere to the CS requirements and use a RIP matrix in order to transform data into lower dimensional space of $\mathcal{O}(s \log(a))$ [23]. This compression space size is easy to obtain, since it depends on the size of the input attributes, which makes it convenient for applications in the streaming context where the total number of instances is unknown. CS is also different from RP which satisfies the JL Lemma 2.4.2 [81] asserting that N instances from a Euclidean space can be projected into a lower dimensional space of $\mathcal{O}(\log N/\epsilon^2)$ dimensions.

In this work, we are only concerned by the compression phase that will alleviate the need of resources in the ARF classification task while dealing with high-dimensional streams. So, for each tree inside our ensemble approach, CS-ARF, we apply a pre-processing step consisting in the CS transformation on every incoming instance via solving Equation (2.4). Therefore, the low-dimensional representation of the current instance will be fed to the underlying ARF ensemble member for prediction and then used to update the corresponding model.

For the purpose of obtaining sufficiently good – or with minor loss in – accuracy and reducing the use computational resources, we need perform projection using an effective sensing matrix A that respects the RIP for the CS application. In this regard, recent studies [114, 115] and Chapter 4 (Table 4.1) assessed the performance of different sensing matrices that satisfy the restricted isometry property with high probability and showed that the CS, using Gaussian random matrices, achieves good results in comparison with other sensing matrices.

In the light of this, we focus on using Gaussian random matrices because of their simplicity and data-independent nature, which is suitable to the evolving data streams nature.

Actually, we do not need the instances from the stream to achieve the projection of high-dimensional data. Instead, we build the sensing matrix A such that its elements are independently generated from a Gaussian distribution $A_{i,j} \sim \mathcal{N}(0,1)$.

Algorithm 5 shows the pseudo-code of the proposed CS-ARF approach. As explained previously, for each ensemble member t, we apply the CS transformation by generating a Gaussian random matrix gm (different from the ones generated for the rest of the ensemble members) and therefore represent the current instance using e low-dimensional representations to fed them to each of the e trees (lines $6-8$), instead of feeding the high-dimensional instance X. Then, we predict the class label for the current compressed dense instance $Y \in \mathbb{R}^m$ (line 9) before using it to train the trees (line 10). For more details about the tree training task and how trees are updated, we redirect readers to the work of Gomes, Bifet, Read, Barddal, Enembreck, Pfharinger, Holmes, and Abdessalem. To handle drifts in the stream, the ARF method includes a warning and drift detection mechanisms, where once a warning is detected for an ensemble member, a background tree is created (lines $11-13$). This tree will be replaced by its corresponding background tree if this warning signal becomes a drift (lines $14-15$).

The main novelty of our approach is in how we internally couple the CS technique with the ARF method to deal with evolving data streams. In fact, we use several CS matrices by generating a different Gaussian matrix for each tree in order to promote diversity inside the ensemble and lose as little as possible in terms predictive performance.

Each ensemble member in our CS-ARF approach will be preceded by a dimensionality reduction step that uses a different sensing matrix.

Therefore, models – or trees – are going to be different inside the CS-ARF ensemble because of: (i) the randomization due to the generation of different Gaussian random

Algorithm 5 CS-ARF algorithm. **Symbols:** $S \in \mathbb{R}^a$: data stream; m: output dimension; e: ensemble size; f: maximum features evaluated per split; C: change detector; B: set of background trees; δ_w: warning threshold; δ_d: drift threshold.

```
function CS-ARF($m, e, f, \delta_w, \delta_d$)
    $T \leftarrow CreateTrees(e)$
    $G \leftarrow GaussianMatrix(e, a, m)$                ▷ generate $e$ random matrices
    $B \leftarrow \emptyset$
    for all $X \in S$ do
        $(x, c) \leftarrow X$
        for all $t \in T$ and $gm \in G$ do
            $y_i \leftarrow CS(x, m, gm)$                 ▷ project $x$ into $m$-dimensions using CS
            $\hat{c} \leftarrow predict(t, y)$
            $TreeTrain(f, t, Y)$                          ▷ train $t$ on the compressed $Y \leftarrow (y, c)$
            if $C(\delta_w, t, Y)$ then                   ▷ if a warning is detected
                $b \leftarrow CreateTree()$               ▷ create a background tree
                $B(t) \leftarrow b$
            end if
            if $C(\delta_d, t, Y)$ then                   ▷ if a drift is detected
                $t \leftarrow B(t)$                       ▷ Replace $t$ by $b$
            end if
        end for
        for all $b \in B$ do
            $TreeTrain(f, b, Y)$
        end for
    end for
end function
```

Table 5.1: Overview of the datasets.

Dataset	#Instances	#Attributes	#Classes	Type
Tweets$_1$	1,000,000	500	2	Synthetic
Tweets$_2$	1,000,000	1,000	2	Synthetic
Tweets$_3$	1,000,000	1,500	2	Synthetic
RBF	1,000,000	200	10	Synthetic
Enron	1,702	1,000	2	Real
IMDB	120,919	1,001	2	Real
Nomao	34,465	119	2	Real
Har	10,299	561	6	Real
ADS	3,279	1,558	2	Real

matrices; and (ii) the construction of the trees by using random subsets of attributes for node splits.

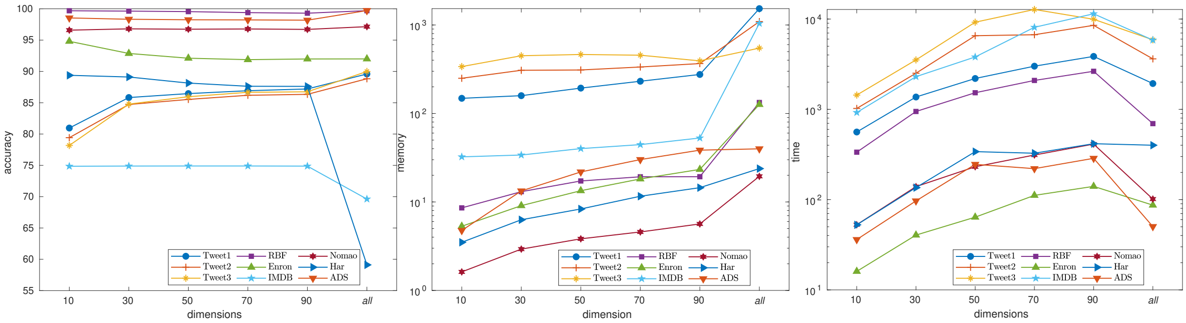
5.4 Experimental Evaluation

In this section, we present with detail all results provided by the proposed CS-ARF method. Then, we analyze them in order to explain the main advantages of using the CS technique.

5.4.1 Datasets

We use 4 synthetic and 5 real datasets that have been thoroughly used in the literature to evaluate the performance of stream classifiers. Table 5.1 presents a short description of each dataset, further details are provided in what follows.
Tweets. Tweets was created using the tweets text data generator provided by MOA [97] that simulates sentiment analysis on tweets, where messages can be classified into two categories depending on whether they convey positive or negative feelings. Tweets$_1$, Tweets$_2$, and Tweets$_3$ produce $1,000,000$ instances of 500, $1,000$, and $1,500$ attributes, respectively.
RBF. The Radial Basis Function (RBF) generator provided also by MOA. It creates centroids at random positions, and each one has a standard deviation, a weight and a class label. This dataset simulates drift by moving the centroids with constant speed.
Enron. The Enron corpus dataset is a large set of email messages that was made public during the legal investigation concerning the Enron corporation [103]. This cleaned version of Enron consists of $1,702$ instances and $1,000$ attributes.
IMDB. IMDB[3] movie reviews dataset was first proposed for sentiment analysis [104], where reviews have been pre-processed, and each review is encoded as a sequence of word indexes (integers).

[3] http://waikato.github.io/meka/datasets/.



(a) accuracy. (b) memory. (c) time

Figure 5.1: CS-ARF and ARF comparison: the CS-ARF while projecting into different dimensions $(10, 30, 50, 70, 90)$; the ARF with the entire datasets (*all* on x-axis).

Nomao. Nomao [124] is a large dataset that has been provided by Nomao Labs. It contains data coming from several sources on the web about places (name, website, address, localization, fax, etc$\cdots$).
Har. Human Activity Recognition dataset [125] built from several subjects performing daily living activities, such as walking upstairs/downstairs, sitting, standing and laying, while wearing a waist-mounted smartphone equipped with sensors. The sensor signals were pre-processed using noise filters and attributes were normalized and bounded within $[-1, 1]$.
ADS. Advertisements dataset[4] is a set of possible advertisements on internet pages, where each row represents one image tagged as ad or nonad (which are the class labels).

5.4.2 Results and Discussions

The experiments were implemented and evaluated in Java by extending the MOA framework [37, 97] using the datasets described above and the online learning setting for Interleaved Test-Then-Train method [91] for evaluation. For a fair comparison, we evaluate the CS-ARF approach against state-of-the-art classifiers coupled with CS as a filter, where we use one CS matrix for DR with all the ensemble members. For the state-of-the-art classification comparison, we use Leveraging Bagging [24] (LB^{cs}), Streaming Random Patches [50] (SRP^{cs}), Hoeffding Adaptive Trees [44] (HAT^{cs}), Self-Adjusting Memory kNN [40] (SAMkNN^{cs}), and Naive Bayes [38] (NB^{cs}) algorithms. We include single classifiers (HAT, SAMkNN, NB) in our comparison, because they are often used as baselines in the stream classification. It has been proved in [25, 50] that the ensemble-based methods, LB and SRP, are the best outperforming other ensemble classifiers using a similar set of datasets to the one used in this work.

Parameterization: we fix $k = 11$ for number of neighbors in the SAMkNN algorithm. We use a similar configuration for the HAT algorithm and Hoeffding tree (HT) – the base

[4] https://www.kaggle.com/uciml/internet-advertisements-data-set.

learner for all the ensemble methods – with the grace period, the split confidence, and subspace size set to $g = 50$, $c = 0.01$, and $m = 80\%$, respectively. To cope with drifts, the ensemble methods are coupled with the change detector and estimator ADWIN [21] using the default parameters for: (i) the warning threshold $\delta_w = 0.00001$; and (ii) the drift threshold $\delta_d = 0.0001$ [24, 25, 50]. We fix the ensemble size to $e = 30$ learners for all the ensemble-based methods.

Figure 5.1 presents the results of the CS-ARF approach, while applying a CS transformation over all the datasets into different space sizes $(10, 30, 50, 70, 90)$, and the vanilla ARF method, whilst using all the input attributes of the data stream without any projection (*all* on the X-axis). We notice that for almost all the datasets, the accuracy of our CS-ARF approach is moderately affected while varying the output dimension p (Figure 5.1(a)). It slightly improves when we increases the p, because we are using random subspaces from a dense set of attributes and not sparse ones (with many zeros). On the other hand, the ARF method using the original data (presented by *all* in the X-axis) somewhat outperforms the CS-ARF approach for almost all datasets. This behaviour is explained by the fact that when we use a dimensionality reduction technique we are removing attributes that may impact the accuracy of any classifier. In contrast, Figure 5.1(b) illustrates the behavior of the memory usage which is different in the sense that vanilla ARF, using the entire data without projection (*all*), is more memory consuming than the CS-ARF approach. Figure 5.1(c) depicts the CS-ARF processing time that increases with p and becomes slower than the ARF method. This is due to the fact that with the CS-ARF approach, we have the additional processing of the CS computation that increases when the CS matrix becomes larger. We highlight that this is an accuracy-resource usage tradeoff, because for a low value of p, our approach is able to be as accurate as the ARF method while using much smaller computational resources. Moreover, the accuracy increases slightly when we increase the number of dimensions to reach the accuracy of the ARF method.

Figure 5.2 shows an accuracy comparison of the CS-ARF approach against reputed state-of-the-art algorithms, coupled with a compressed sensing filter, on the Tweet$_1$ dataset. We notice that our approach achieves consistently better accuracy than its competitors for different output dimensions. Single classifiers (HAT^{cs}, SAMkNN^{cs}, NB^{cs}) are less accurate than the ensemble-based methods because the latter combine the predictions of several single "weak" classifiers and are all coupled with drift detection techniques.

Due to the stochastic nature of the CS technique and therefore our CS-ARF approach, all the results reported in this work are an average of several runs (with different random Gaussian matrices). Figure 5.3 depicts the standard deviation based on the accuracies obtained over several runs for different output dimensions using Tweet$_3$ and Har datasets (Figure 5.3(a) and 5.3(b), respectively). For both datasets, our approach has a small standard deviation (too close to zero), i.e. for all the runs, the accuracies obtained are close to the mean reported in this chapter. On the other hand, a larger standard deviation is obtained with

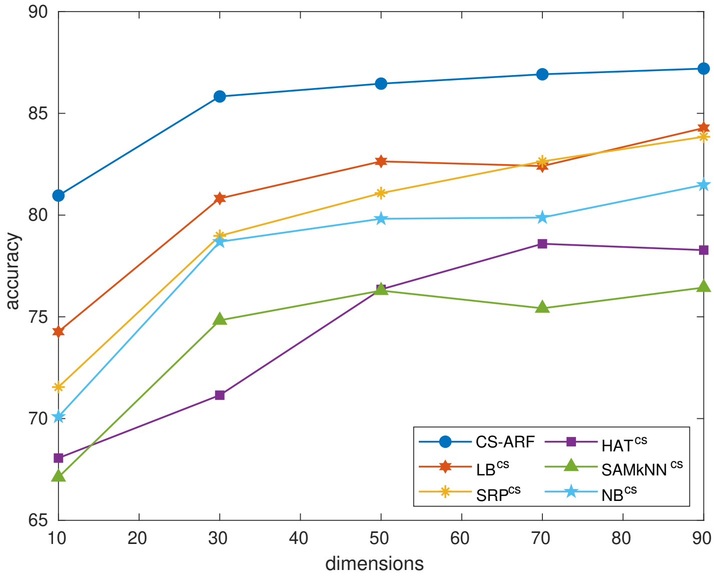
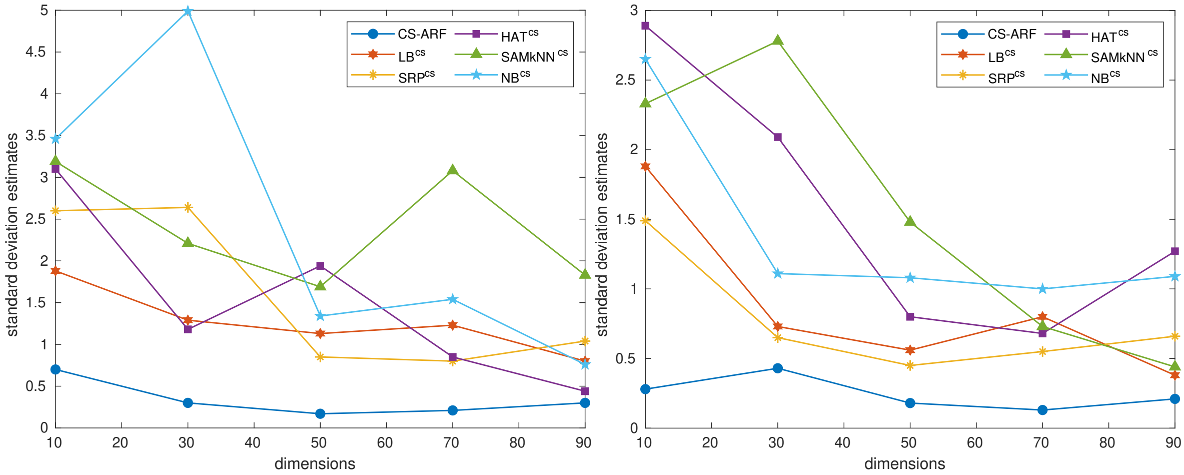


Figure 5.2: Accuracy comparison over different output dimensions on Tweet1 dataset.

the other algorithms showing that the classification accuracies obtained for the different runs are farther away from the mean. This difference is explained by the fact that the competitors use one CS matrix as an internal filter while our approach uses a different Gaussian matrix for each ensemble member. This strategy somewhat increases the diversity inside the ensemble and thus a better predictive performance is obtained, guaranteeing some close approximation (with a CS perturbation ϵ) to the accuracy that would be obtained using the original stream. Based on these results, we use $p = 50$ in the following, because the standard deviation is minimal for most of the algorithms.

The results presented in Table 5.2 show the classification performance of the CS-ARF approach against other algorithms for all datasets projected in a space of 50-dimensions using the compressed sensing technique. We note that the CS-ARF performs the best on most of the datasets and highlight the difference that is statistically insignificant when outperformed by other algorithms, as reported in [50].

To assess the benefits in terms of resources – where small values are desirable – Figure 5.4 shows the memory behavior for the ensemble-based methods. This figure depicts the large gains on almost all datasets of our approach, CS-ARF, which outperforms the LB^{cs} and the SRP^{cs} methods, confirming previous studies [25, 50] that reveal the high consumption of the LB. We also note that with small datasets, such as Enron and ADS, the CS-ARF does not achieve a prominent gain. Indeed, with large datasets our proposed approach is efficient which makes it highly convenient for high-dimensional data streams where the stream size is potentially infinite, which is not the case of the Enron and ADS datasets.



(a) Tweet$_3$ dataset.

(b) Har dataset.

Figure 5.3: The standard deviation of the methods while projecting into different dimensions

Table 5.2: Accuracy (%) comparison of CS-ARF, LB^{cs}, SRP^{cs}, SAMkNN^{cs}, and NB^{cs}.

Dataset	CS-ARF	LB^{cs}	SRP^{cs}	HAT^{cs}	SkNN^{cs}	NB^{cs}
Tweets$_1$	**86.46**	82.64	81.08	76.35	76.29	79.82
Tweets$_2$	**85.53**	81.88	80.93	76.69	74.06	79.48
Tweets$_3$	**86.96**	79.65	78.58	71.30	72.61	78.24
RBF	99.55	99.50	99.74	96.20	**99.77**	96.41
Enron	92.11	96.18	**96.35**	94.59	96.17	91.37
IMDB	**74.90**	74.86	74.87	74.04	74.55	74.27
Nomao	**96.74**	96.70	96.68	95.02	96.63	86.25
Har	88.14	88.61	**88.65**	80.22	82.07	81.72
ADS	98.25	99.74	**99.81**	98.71	98.52	89.48
Overall ∅	**89.65**	88.91	88.52	84.79	85.63	84.11

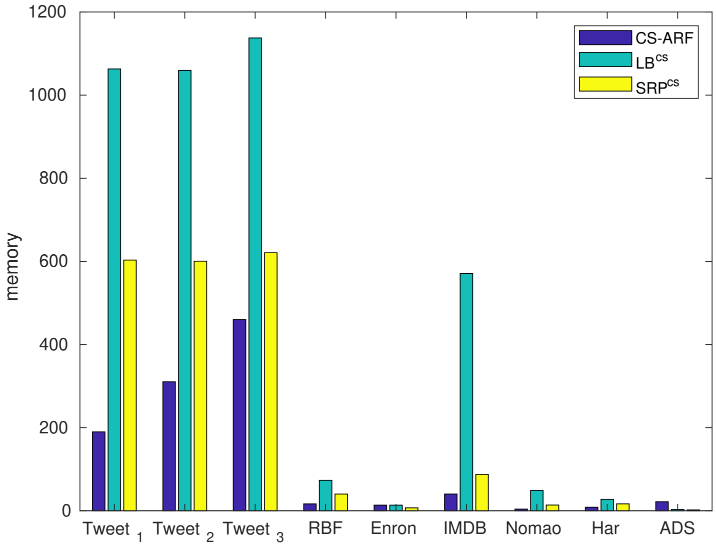


Figure 5.4: Memory (MB) comparison of the ensemble-based methods on all the datasets.

5.5 Conclusion

In this work, we presented the compressed adaptive random forest approach (handles the **Q3**) to enable the ARF to be both efficient (in terms of resource usage) and effective (in terms of classification accuracy) with high-dimensional data streams (**Q1** and **Q2**). The CS-ARF approach combines the CS technique, given its ability to preserve pairwise distances within $1 \pm \epsilon$-factor, in conjunction with the strength of the reputed ARF method, that achieves high predictive performance. Our proposed approach transforms high-dimensional data streams, using the CS technique as an internal online pre-processing step, afterwards it uses the corresponding obtained low-dimensional representation for the learning task using the ARF method.

We evaluated and discussed the proposed method via extensive experiments using a diverse set of datasets. Results showed the ability of our approach to achieve good performance, close to what would be obtained using the original datasets without projections, and outperform well-known state-of-the-art algorithms. We also showed that, despite its stochastic nature, the CS-ARF approach achieves good stable accuracy, by extracting relevant attributes from sparse data in different low-dimensional spaces, while using feasible amount of resources.

6

Batch-Incremental Classification Using UMAP

Contents

This chapter concerns a collaboration [126] with Bernhard Pfahringer[1] published at the Symposium on Intelligent Data Analysis (IDA) 2020 under the title "*Efficient Batch-Incremental Classification Using UMAP for Evolving Data Streams*".

6.1 Introduction

Data stream learning – or incremental learning – approaches can generally be divided into two main branches [113]:

[1] University of Waikato, Hamilton, New Zealand.

- **Instance-incremental** approaches that update the model with each instance as soon as it arrives, e.g. (i) Naive Bayes [127] which uses the Bayes formula to compute posterior probabilities; (ii) Self-Adjusting Memory kNN (sam$k$NN) [40] which is a streaming version of kNN [22]; and (iii) Hoeffding Adaptive Tree (HAT) [44] which is an extension of the Hoeffding decision trees [41] to deal with concept drifts. See Section 2.3 for more approaches.

- **Batch-incremental** approaches (multiple instances) which make no change/increment to their model until a batch is completed. The main parameter to be fixed is then the size of batches. E.g. (i) logistic regression [128] that uses a logistic function to model a dependent variable; (ii) support vector machines [129] which builds a model that assigns new instances into the space based on their class labels; and (iii) batch-incremental ensemble of decision trees [130] which divides the stream into single batches then, after learning from each batch, the ensemble combines the underlying models to one global model.

In this chapter, we propose another attempt to improve the performance of the kNN that consists in incorporating a batch-incremental feature transformation strategy to tackle potentially high-dimensional and possibly infinite batches of data streams while ensuring effectiveness and quality of learning (e.g., accuracy), which addresses the questions **Q1** and **Q2**. This is achieved via a new DR technique that has attracted a lot of attention recently: Uniform Manifold Approximation and Projection (UMAP) [26](Section 2.4.3), built upon rigorous mathematical foundation through the Riemannian geometry. As far as we know, no incremental – streaming or online – version of UMAP exists which makes it not applicable on very large dynamic datasets. In order for that to happen, the approach outlined in this chapter proposes a batch-incremental strategy where we use UMAP as an internal pre-processing step to the kNN algorithm on evolving data streams.

This chapter is organized as follows. Section 6.2 reviews the prominent related work. Section 6.3 gives a brief background of UMAP, followed by the description of our approach. In Section 6.4, we outline and discuss the results of experiments on diverse datasets. Finally we draw our conclusions.

6.2 Related Work

Beyond any doubt, DR is a powerful tool in data science to look for hidden structure in data and reduce the resources usage of learning algorithms. DR techniques facilitate the classification task, by removing redundancies and extracting the most relevant features in the data, and permits a better data visualization. A simple sub-taxonomy (to the one introduced in Figure 2.7) divides these techniques into two major groups as follows: *matrix factorization* and *graph/neighborhood-based* techniques.

- **Matrix factorization** techniques require matrix computation tools, such the well-known PCA [131]. It uses singular value decomposition and aims to find a lower-dimensional basis by converting the data into features called principal components by computing the eigenvalues and eigenvectors of a covariance matrix. Different incremental versions of PCA have been developed to handle streams of data [63, 59, 58], *inter alia*, a batch-incremental PCA have been proposed to incrementally learn a subspace representation using successive batches [59] (see Section 2.4.1 for details).
- **Graph/Neighborhood-based** techniques are leveraged in the context of DR and data visualization by using insight that similar instances in the high-dimensional space should be represented by close instances in the corresponding low-dimensional space, whereas dissimilar instances should be well separated. As introduced in Section 2.4.3, tSNE [90] converts pairwise high-dimensional Euclidean distances between input instances into conditional probabilities P which represent matrix of pairwise similarities. Likewise a probability distribution Q is computed describing the similarity in the lower dimensional space after a first random projection. The objective behind t-SNE is to find a representation in a low-dimensional space where Q faithfully represents P. To do so, an optimization scheme is used to minimize the difference between P and Q over all instances. In addition to the fact that it is computationally expensive, t-SNE has some limitations while projecting in a two–or three-dimensions. In fact, it does not preserve distances between all instances nor density, it only preserves nearest-neighbors and can affect any density- or distance-based algorithm and hence preserves more of the local structure than the global structure [90].

6.3 Batch-Incremental Classification

In this section, we present a batch-incremental adaptation of the UMAP technique [26] for the stream kNN algorithm.

6.3.1 Prior Work

Unlike tSNE [90], the UMAP [26] serves not only for visualization but also as a general DR technique. It uses the concept of k-nearest neighbors by constructing open balls over all instances and building simplicial complexes (Section 2.4.3).

UMAP offers better visualization quality than tSNE by preserving more of the global structure in a shorter running time.

To the best of our knowledge, tSNE and UMAP do not have any incremental version, and, ultimately, both techniques are essentially transductive[2] and do not learn a mapping function from the input space. Hence, they need to process all the instances for each new unseen observation, which prevents them from being applicable within the stream framework.

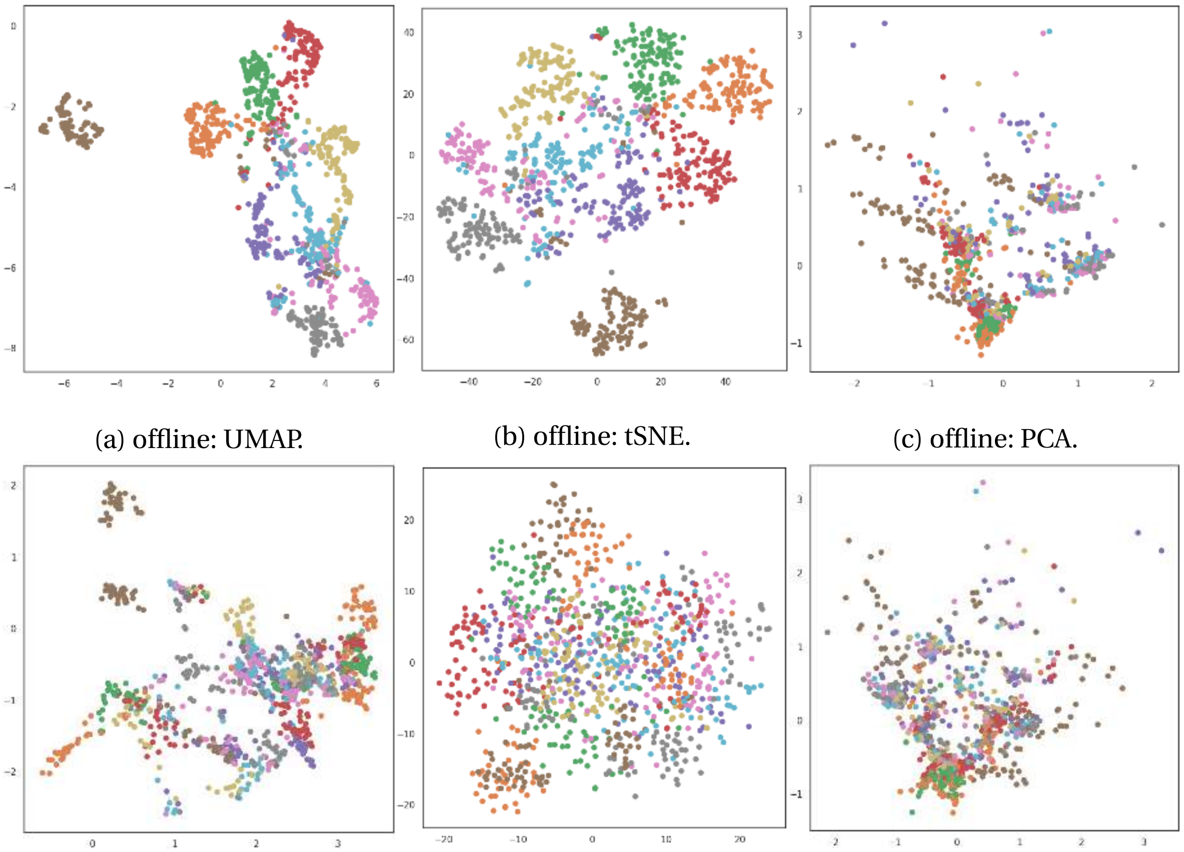
Figure 6.1 shows the projection of CNAE dataset [105] (see Table 6.1) into 2-dimensions with UMAP, t-SNE, and PCA in an offline/online fashions where each color represents a label. In Figure 6.1(a), we notice that UMAP offers the most interesting visualization and has far better local structure showing its embedding effectiveness on separating classes clearly (9 classes), by putting similar data together, beating t-SNE and PCA (Figures 6.1(b) and 6.1(c) respectively). The overlap in the new space, for instance with tSNE in Figure 6.1(b), can potentially affect later classification task, notably distance-based algorithms because properties like global distances and density may be lost in favor of preserving local structure. On the other hand, linear transformation, such as PCA, cannot discriminate between instances which prevents them from being represented in the form of clusters as other algorithms (Figure 6.1(c)).

To motivate our choice in the following, we project the same dataset with these approaches using our batch-incremental strategy (more details in Section 6.3.2). Figure 6.1(d) illustrates the change from the offline UMAP representation which is not as drastic as the ones engendered by tSNE and PCA (Figures 6.1(e) and 6.1(f), respectively) showing their limits on capturing information from data that arrive in a batch-incremental manner.

6.3.2 Algorithm Description

A very efficient and simple scheme in supervised learning is *lazy learning* [132, 133]. Since lazy learning approaches are based on distances between every pair of instances, they unfortunately exhibit low performance in terms of execution time. The kNN algorithm is a well-known lazy algorithm that does not require any work during training, so it uses the entire dataset to predict labels for test instances. However, it is impossible to store an evolving data stream which is potentially infinite – nor to scan it multiple times – due to its tremendous volume. To tackle this challenge, a basic incremental version of kNN has been proposed which uses a fixed-length window that slides through the stream and merges new arriving instances with the closest ones already in the window (Section 2.3.2). To predict the class label for an incoming instance, we take the majority class labels of its nearest neighbors inside the window using a defined distance metric (Equation 4.2). Since we keep the recent arrived instances inside the sliding window for prediction, the search for the nearest neighbors is still costly in terms of memory and time [39] and high-dimensional

[2] Transductive learning consists on learning on a dataset but predicting on a known set of unlabeled instances from the same dataset.



(a) offline: UMAP.

(b) offline: tSNE.

(c) offline: PCA.

(d) batch-incremental: UMAP.

(e) batch-incremental: tSNE.

(f) batch-incremental: PCA.

Figure 6.1: Projection of CNAE dataset in 2-dimensional space.

data stream impose further resources.

The main idea to cope with this drawback and improve the performance of kNN consists of using an efficient strategy which has consistent results, such as UMAP.

Since UMAP is a transductive technique, an instance-incremental learning approach that includes UMAP does not work because the entire stream needs to be processed whenever a new instance arrives. By doing it this way, the process will be costly and will not respond to the streaming requirements. To alleviate the processing cost considering the framework within which several challenges shall be respected, including the memory constraint and the incremental behavior of data, we adopt a batch-incremental strategy. In the following, we introduce the procedure of our novel approach, batch-incremental UMAP-kNN.

Step 1: ***Partition of the stream.***
During this step, we must address the major weakness of UMAP, which is its uneffectiveness with instance-incremental manner. The basic idea is to employ a batch-incremental strategy consisting on processing subsets of the stream. To do so, we assume that data arrive in batches – or chunks – by dividing the stream into disjoint partitions $S_1, S_2, \cdots$ of size s.

Figure 6.2 shows a stream of instances divided into batches of equal size. So, instead of having observations available one by one i.e., one at a time, they will arrive as a group of instances simultaneously, $S_1, S_2, \ldots S_q$, where S_q is the qth chunk. A simple example of data stream is a video sequence where at each instant we have a succession of images.

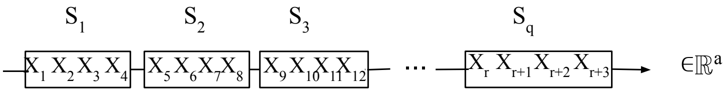


Figure 6.2: Stream of mini-batches.

Step 2: ***Data pre-processing.***
We focus on the analysis of an infinite stream of high-dimensional instances $X_i \in \mathbb{R}^a$ from which we wish to construct a low-dimensional representation $Y_i \in \mathbb{R}^m$, where $m \ll a$.

As mentioned before, UMAP is unable to compress data incrementally and needs to transform more than one instance at a time because it builds a neighborhood-graph on a set of instances and then lays it out in a lower dimensional space [26]. Thus, our proposed approach operates on batches of the stream where a single batch S_i of data is processed at a time T_i. The two first steps in Figure 6.3 show the application of UMAP on disjoint batches. Once a batch is complete, throughout the second step, we apply UMAP on it independently from the batches that have been already processed, so each $S_i \in \mathbb{R}^a$ will be transformed and represented by $S_i \in \mathbb{R}^m$. This new representation is very likely devoid of redundancies, irrelevant attributes, and is obtained by finding potentially useful non-linear combinations

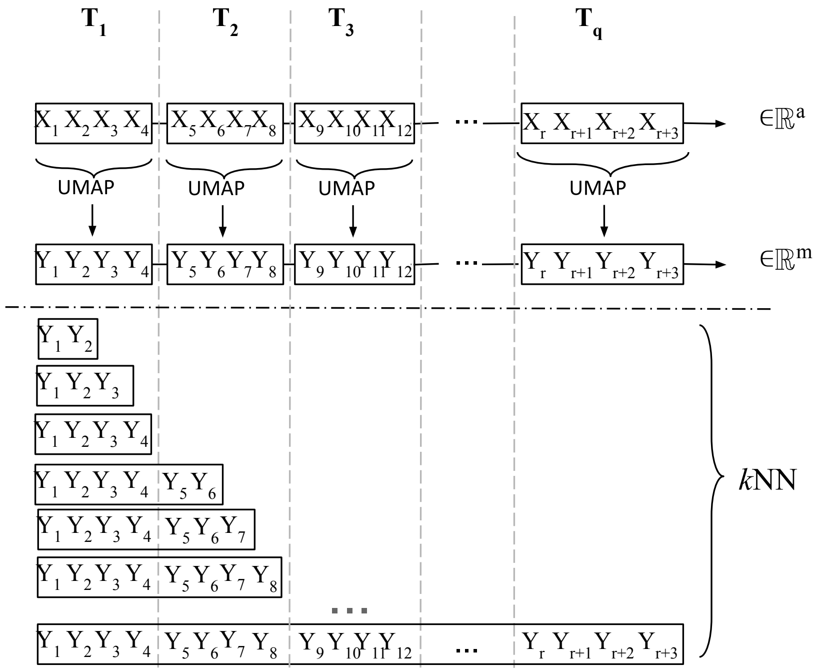


Figure 6.3: Batch-incremental UMAP-kNN scheme.

of existing attributes, i.e. by repacking relevant information of the larger feature space and encoding it more compactly.

For UMAP to learn when moving from a batch to another, we seed each chunk's embedding with the outcome of the previous one, i.e., match the prior initial coordinates for instances in the current embedding to the final coordinates in the preceding one. This will help to avoid losing the topological information of the stream and to keep stability in successive embeddings as we transition from one batch to its successor. Afterwards, we use the compressed representation of the high-dimensional chunk for the next step that consists in supporting the incremental kNN classification algorithm internally.

Step 3: *kNN classification.*

The UMAP-kNN approach aims to decrease the computational costs of kNN on high-dimensional data stream by reducing the input space size using the famous dimension reduction UMAP in a batch-incremental way. Besides the prediction phase of the kNN algorithm that is based on the neighborhood[3], UMAP operates on a k-nearest graph (topological representation) as well and optimizes the low-dimensional representation of the data using gradient descent. One nice takeaway is that UMAP, because of its solid theoretical backing as a manifold technique, keeps properties such as density and pairwise

[3] The distances between the new incoming instance and the instances already available inside the adaptive window (**Q3**) are computed in order to assign it to a particular class.

distances. Thus, it is not going to bias the performance of he neighborhood-based kNN algorithm.

This step consists of classifying the evolving data stream, where the learning task occurs on consecutive batches, i.e. we train incrementally kNN with instances becoming successively available in chunk buffers after pre-processing. Figure 6.3 shows the underlying batch-incremental learning scheme used which is built upon the divide-and-conquer strategy. Since UMAP is independently applied to batches, so once a chunk is complete and has been transformed in $\mathbb{R}^m$, we feed the half of the batch to the sliding window and we predict incrementally the class label for the second half (the rest of instances).

Given that kNN is adaptive, the main novelty of UMAP-kNN is in how it merges the current batch to previous ones. This is done by adding it to the instances from previous chunks inside the kNN window. Even if past chunks have been discarded, only some of them have been stored and maintained while the adaptive window scrolls. Thereafter, instances kept temporarily inside the window are going to be used to define the neighborhood and predict the class labels for later incoming instances. As presented in Figure 6.3, the intuitive idea to combine results from different batches is to use the half of each batch for training and the second half for prediction. In general, due to the possibility of having often very different successive embeddings, one would expect that this may affect the global performance of our approach. Thus, we adopt this scheme to maintain a stability over an adaptive batch-incremental manifold classification approach.

Algorithm 6 UMAP-kNN algorithm. **Symbols:** $S = \{X_1, X_2, \ldots\} \in \mathbb{R}^a$: data stream; $S = \{Y_1, Y_2, \ldots\} \in \mathbb{R}^m$: transformed data stream; s: batch size; $C = \{c_1, c_2, \ldots\}$: set of class labels; W: sliding window; k: the number of neighbors.

```
function UMAP-$k$NN($S, s, w, k, m$)
    $W \leftarrow \emptyset$, $j \leftarrow 0$
    for all $S_i \in S$ do
        $S_i \leftarrow$ UMAP($S_i$,$m$)                          ▷ apply UMAP
        $W \leftarrow S_{i[1\cdots\frac{s}{2}]}$
        $j \leftarrow \frac{s}{2} + 1$
        for all $Y_j \in S_{i[\frac{s}{2}\cdots s]}$ do
            for all $Y_k \in W$ do                             ▷ $\forall\, k \neq j$
                compute $D_{Y_k}(Y_j)$                         ▷ Equation (4.2)
            end for
            $c \leftarrow \max_{c \in C} D_{S,k}(Y_j)$          ▷ Equation (4.3)
            $W \leftarrow Y_j$                 ▷ maintain the projected $X_j$, $Y_j$, in $W$
        end for
    end for
end function
```

Algorithm 6 shows the pseudo-code of the UMAP-kNN. We start by applying UMAP on each new incoming chunk S_i. Then, we provide a dense representation of it, instead of sparse high-dimensional one. Later, we feed the half of the transformed instances, S_i, to the window W (line 5). Afterwards, we apply kNN on the second half of the chunk by computing the distance between Y_j and the accumulated instances inside W (lines $7-10$). Finally, we report the most frequent class label to Y_j by taking the majority vote over its nearest neighbors and add it to the window (lines $10-11$).

6.4 Experimental Evaluation

In this section, we present a series of experiments carried out on synthetic and real datasets to assess the classification performance of our proposal.

6.4.1 Datasets

In order to show whether the UMAP-kNN approach is capable of working in different scenarios, we use 3 synthetic and 6 real-world high-dimensional datasets from different scenarios that have been thoroughly used in the literature to evaluate the classification performance of data streams classifiers. Table 6.1 presents a short description of each dataset, and further details are provided in what follows.

Tweets. The dataset was created using the tweets text data generator provided by MOA [97] that simulates sentiment analysis on tweets, where messages can be classified depending on whether they convey positive or negative feelings. Tweets$_{1,2,3}$ produce instances of 500, $1,000$ and $1,500$ attributes respectively.

Har. Human Activity Recognition dataset [125] built from several subjects performing daily living activities, such as walking, sitting, standing and laying, while wearing a waist-mounted smartphone equipped with sensors. The sensor signals were pre-processed using noise filters and attributes were normalized.

CNAE. CNAE is the national classification of economic activities dataset [105]. Instances represent descriptions of Brazilian companies categorized into 9 classes. The original texts were pre-processed to obtain the current highly sparse data.

Enron. The Enron corpus dataset is a large set of email messages that was made public during the legal investigation concerning the Enron corporation [103]. This cleaned version of Enron consists of $1,702$ instances and $1,000$ attributes.

IMDB. IMDB movie reviews dataset was proposed for sentiment analysis [104], where each review is encoded as a sequence of word indexes (integers).

Nomao. Nomao dataset [124] was provided by Nomao Labs where data come from several sources on the web about places (name, address, localization, etc).

Covt. The forest covertype dataset obtained from US forest service resource information system data where each class label presents a different cover type.

Table 6.1: Overview of the datasets.

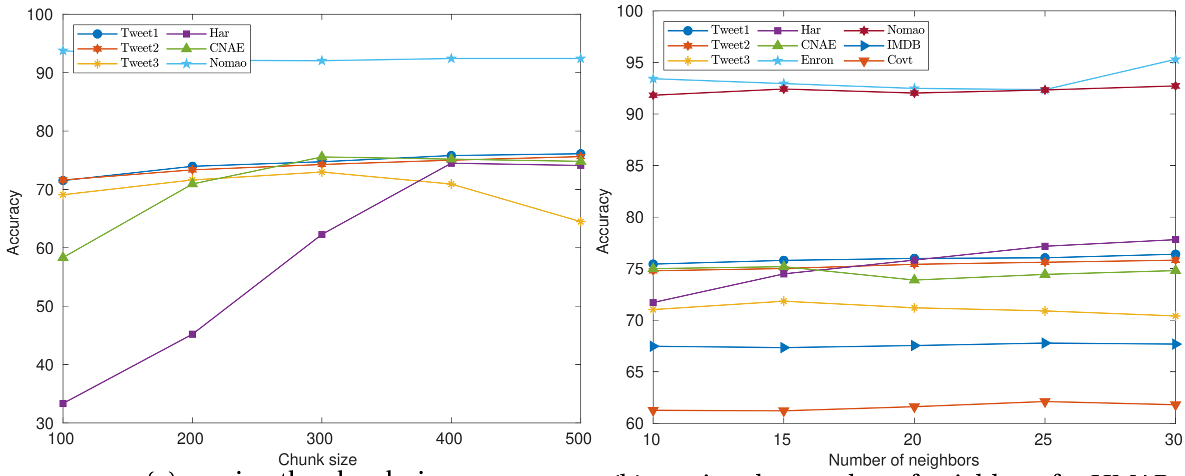
Dataset	#Instances	#Attributes	#Classes	Type
Tweets_1	1,000,000	500	2	Synthetic
Tweets_2	1,000,000	1,000	2	Synthetic
Tweets_3	1,000,000	1,500	2	Synthetic
Har	10,299	561	6	Real
CNAE	1,080	856	9	Real
Enron	1,702	1,000	2	Real
IMDB	120,919	1,001	2	Real
Nomao	34,465	119	2	Real
Covt	581,012	54	7	Real

6.4.2 Results and Discussions

We compare our proposed classifier, UMAP-kNN, to various commonly-used baseline methods in dimensionality reduction and machine learning areas. PCA [59] provided in scikit-learn [134], which is an incremental-batch approach that updates the eigenbasis for each batch buffer, tSNE (fixing the perplexity to 30, which is the best value as reported in [90]), SAM-kNN (SkNN) [40]. We use HAT, a classifier with a different structure based on trees [44], to assess its performance with the neighborhood-based UMAP. For fair comparison, we compare UMAP-kNN against PCA and t-SNE using the same strategy proposed in this chapter both with kNN. We compare the performance of UMAP-kNN approach with competitor classifiers using UMAP as well and the same batch-incremental manner. Actually, incremental kNN has two crucial parameters: (i) the number of neighbors k fixed to 5; and (ii) the window size W, that maintains the low-dimensional data, fixed to 1000. According to previous studies such as [39], a bigger window will increase the resources usage and smaller size will impact the accuracy. All experiments were implemented and evaluated in Python by extending the Scikit-multiflow framework[4] [135] and scikit-learn [134] using the datasets described in Section 6.1.

Figure 6.4(a) depicts the influence of the chunk size on the accuracy using UMAP-kNN with some datasets. Generally, fixing the chunk size imposes the following dilemma: choosing a small size so that we obtain an accurate reflection of the current data or choosing a large size that may increase the accuracy since more data are available. The ideal would be to use a batch with the maximum of instances to represent as possible the whole stream. In practice, the chunk size needs to be small enough to fit in the main memory otherwise the running time of the approach will increase. UMAP is a slow technique, so we choose small chunk sizes to overcome this issue with UMAP-kNN. Based on the obtained results in Figure 6.4(a), we fix the chunk size to 400 for an efficient tradeoff accuracy-memory.

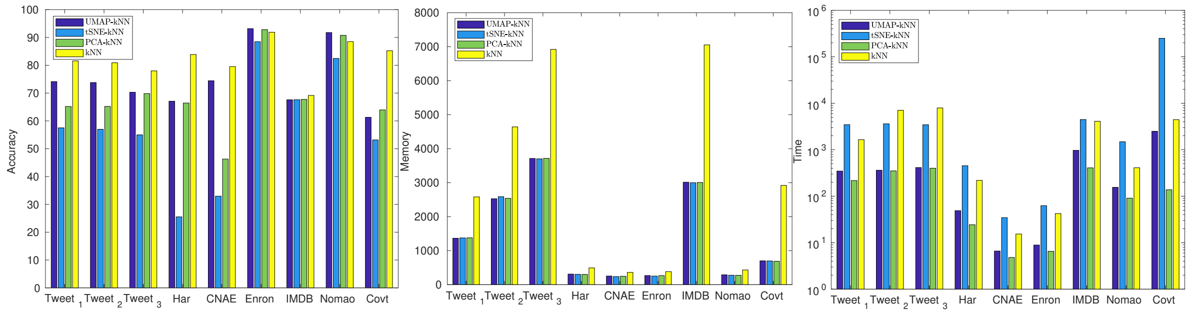
[4] https://scikit-multiflow.github.io/.



(a) varying the chunk size.

(b) varying the number of neighbors for UMAP.

Figure 6.4: Accuracy while varying the chunk size and the number of neighbors.



(a) accuracy.

(b) memory.

(c) time.

Figure 6.5: Comparison of UMAP-kNN, tSNE-kNN, PCA-kNN, and kNN (with the entire datasets) while projecting into 3 dimensions.

We investigate the behavior of a crucial parameter that controls UMAP, number of neighbors, via the classification performance of our approach. Based on the size of the neighborhood, UMAP constructs the manifold and focuses on preserving local and global structures. Figure 6.4(b) shows the accuracy when the number of neighbors is varied on diverse datasets. We notice that for all datasets, the accuracy is consistently the same with no large differences, e.g. Har. Thanks to the batch-incremental strategy that permits for each chunk to maintain its structure in the sliding window to support the kNN classification, we therefore offer a stability for successive embeddings by merging them. Since large neighborhood leads to a slower learning process, in the following we fix the number of neighbors for UMAP to 15.

tSNE is a visualization technique, so we are limited to project high-dimensional data into 2 or 3 dimensions. In order to evaluate the performance of our proposal in a fair comparison against each of tSNE-kNN and PCA-kNN, we project data into 3-dimensional space. We illustrate in Figure 6.5(a) that UMAP-kNN makes significantly more accurate predictions beating consistently the best performing baselines (tSNE-kNN and PCA-kNN) notably with the CNAE and tweets datasets. Figure 6.5(b) depicts the quantity of memory needed by the three algorithms which is practically the same for some datasets. Compared to kNN that uses the whole data without projection, we notice that UMAP-kNN consumes much less memory whilst sacrificing a bit in accuracy because we are removing many attributes. Figure 6.5(c) shows that our approach is consistently faster than tSNE-kNN because tSNE computes the distances between every pair of instances to project. On the other hand, PCA-kNN is a bit faster thanks to the simplicity of PCA in comparison with the manifold learning technique UMAP. But with this tradeoff our approach performs good on almost all datasets.

In addition to its good classification performance in comparison with competitors, the batch-incremental UMAP-kNN did a better job of preserving density by capturing both of global and local structures, as shown in Figure 6.1(d). The fact that UMAP and kNN are both neighborhood-based methods arises as a key element in achieving a good accuracy. UMAP has not only the power of visualization but also the ability to reduce the dimensionality of data efficiently which makes it useful as pre-processing technique for machine learning.

Table 6.2 reports the comparison of UMAP-kNN against state-of-the-art classifiers. We highlight that our approach performs better on almost all datasets. It achieves similar accuracies to UMAP-SkNN on several datasets but in terms of resources, the latter is slower because of its drift detection mechanism. UMAP-kNN has a better performance than PCA-kNN, e.g., the Tweets datasets at the cost of being slower. We also observe the UMAP-HAT failed to overcome our approach (in terms of accuracy, memory, and time) due to the integration of a neighborhood-based technique (UMAP) to a tree structure (HAT).

Figure 6.6 reports detailed results for Tweet$_1$ dataset with five output dimensions. Figure 6.6(a) exhibits the accuracy of our approach which is consistently above competitors whilst ensuring stability for different manifolds. Figures 6.6(b) and 6.6(c) show that kNN-based classifiers use much less resources than the tree-based UMAP-HAT. We see that UMAP-kNN requires less time than UMAP-HAT and UMAP-SkNN to execute the stream but PCA-kNN is fastest thanks to its simplicity. Still, the gain in accuracy with UMAP-kNN is more significant.

6.5 Conclusion

Motivated by the high performance of UMAP proposed recently by McInnes, Healy, and Melville, this chapter addresses the problem of high-dimensionality using UMAP applied to the stream classification (**Q2**). We presented a novel batch-incremental approach for mining data streams using the adaptive kNN algorithm (**Q3**). UMAP-kNN combines the simplicity of kNN and the high performance of UMAP which is used as an internal incremental pre-

Table 6.2: Comparison of UMAP-kNN , PCA-kNN, UMAP-SkNN, and UMAP-HAT.

Accuracy (%)

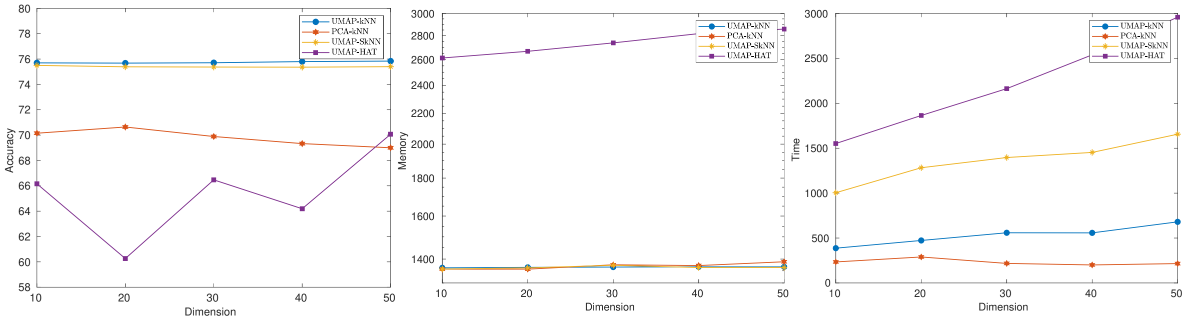
Dataset	UMAP-kNN	PCA-kNN	UMAP-SkNN	UMAP-HAT
Tweets$_1$	75.71	69.89	75.37	66.47
Tweets$_2$	75.16	69.21	74.40	61.27
Tweets$_3$	71.01	70.81	70.47	66.98
Har	75.30	70.50	64.09	84.89
CNAE	76.67	67.41	75.18	40.18
Enron	92.24	93.41	91.89	91.77
IMDB	67.38	67.28	67.43	64.52
Nomao	91.92	91.13	91.63	83.75
Covt	61.29	66.73	53.08	55.43

Memory (MB)

Dataset	UMAP-kNN	PCA-kNN	UMAP-SkNN	UMAP-HAT
Tweets$_1$	1366.71	1354.24	1373.15	2738.32
Tweets$_2$	2530.30	2518.76	2532.95	4891.23
Tweets$_3$	3706.99	3706.55	3722.68	7144.77
Har	311.58	310.48	312.84	381.49
CNAE	254.17	246.94	260.29	262.52
Enron	269.00	267.31	271.56	288.74
IMDB	3012.85	3013.28	3018.04	7471.64
Nomao	289.81	285.50	290.60	508.50
Covt	700.69	689.97	704.46	3788.54

Time (Sec)

Dataset	UMAP-kNN	PCA-kNN	UMAP-SkNN	UMAP-HAT
Tweets$_1$	558.56	217.44	1396.32	2163.14
Tweets$_2$	616.50	350.63	908.59	3453.21
Tweets$_3$	667.43	400.62	1066.98	6273.19
Har	75.20	24.37	77.99	82.47
CNAE	8.89	4.81	13.17	19.78
Enron	12.80	9.52	17.26	32.84
IMDB	715.68	407.60	1038.77	4691.07
Nomao	248.79	20.46	327.36	228.00
Covt	2311.21	137.62	3756.41	2297.01



(a) accuracy.

(b) memory.

(c) time.

Figure 6.6: Comparison of UMAP-kNN, PCA-kNN, UMAP-SkNN, and UMAP-HAT over different output dimensions using Tweet$_1$.

processing step to reduce the feature space of data streams (**Q1**). We showed that UMAP is capable of embedding efficiently data streams within a batch-incremental strategy.

We assessed the performance of the proposed approach in an extensive evaluation with well-known state-of-the-art algorithms considering a diverse set of datasets. We further demonstrated that the batch-incremental approach is just as effective as the offline approach in visualization and its classification performance significantly outperforms reputed baselines while using reasonable resources usage.

We would like to pursue our promising approach further to enhance its run-time performance by applying a fast dimension reduction before using of UMAP. Another area for future work could be the use of a different mechanism, such as the application of UMAP for each incoming data inside a sliding window. We believe that this may be slow but will be suited for instance-incremental learning.

Part III

Concluding Remarks

7 Conclusions and Future Work

Contents

The amount of data streams generated daily by active devices and sensors is considerably increasing. Useful knowledge can be extracted from these evolving data for later decision-making. These data streams pose however several challenges for learning algorithms, including mainly, but not limited to, restricted resources (in terms of memory usage and running time), high-dimensionality, and concept drift constraints.

In this thesis, we investigated the problem of evolving data stream classification while addressing the aforementioned challenges. This chapter concludes the thesis with a discussion on the completed work and indicates possible lines of further investigation.

7.1 Conclusions

We thoroughly analyze and present the two main problems addressed in this thesis, which are the data stream classification and summarization, with a focus on sketching and dimensionality reduction.

In the first step of the thesis, we defined the basic characteristics of data streams and discussed different approaches in the research, designed to handle the challenges of such evolving environments. During our discussion, we focused on addressing the stream challenges in the context of classification. In this context, we reviewed in detail the state-of-the-art algorithms of classification and dimensionality reduction. The aim of this thesis settled on improving current stream classifiers by developing new approaches that use DR techniques. During the introductory part in Chapter 1, we divided our objectives in three research questions (**Q1**, **Q2**, **Q3**) that were our main focus throughout our contributions.

7.1.1 Naive Bayes Classification

In Chapter 3, we studied the well-known summarization technique with promising theoretical guarantees, CMS [19]. We adopted a strategy that allows us to compactly store instances from the stream in a sketch table with a fixed size, during the learning phase, in a way that the data could be easily retrieved from it for prediction using the Naive Bayes algorithm (**Q1** handled). Theoretical guarantees characterizing the size of the sketch are provided to build the sketch table, capable to store synopsis of data without much loss in information.

Concept drift is defined as the changes which occur in the learned model due to time-evolving stream distributions. These changes mainly involve some fluctuations of the underlying distributions. The occurrence of concept drifts sometimes leads to a significant drop in the predictive performance for some classifiers. Moreover, models and partitions need to be updated with new information, that is why the new proposed classification algorithms dedicated to data streams are generally coupled with a drift detection mechanism. To cope with this phenomena and address **Q3**, we incorporated a drift detection strategy using ADWIN [21] to the SketchNB.

To cope with high-dimensional **Q2**, we added an incremental pre-processing step to the above-mentioned SketchNB, and its adaptive version, to reduce the dimensionality of data before storing them in the sketch table and therefore minimize the resources that would be used for high-dimensional data. For this to happen, we used the hashing trick [20], a fast and simple DR technique, that employs a hash function for projection.

7.1.2 Lazy learning

Valuable results have been obtained during the research of the first main contribution, where we performed several experiments on a diverse set of datasets (including high-dimensional data) by evaluating well-known learners – among others the kNN algorithm. In our results, we noticed that the latter is very costly, i.e., it uses huge amounts of resources (in terms of memory and time). These results motivated us to investigate ways to enhance the kNN performance using successful DR techniques. In the following, we sum up the principal contributions to the lazy learning under the streaming framework:

- In Chapter 4, we proposed an efficient kNN approach that reduces high-dimensional data streams (**Q2**) using a data-independent DR technique, CS [23], before feeding their low-dimensional representation to the sliding window of the kNN algorithm (**Q3**). This approach is indeed memory-efficient and faster than the traditional incremental kNN because it processes low-dimensional data instead of high-dimensional data (**Q1**). Linked to some geometrical properties, we theoretically proved that the neighborhood before and after projection (using the CS technique) is preserved up some ϵ-distortion. This means that our CS-kNN approach achieves an accuracy close to the one that would be obtained using the input high-dimensional stream (without projection) with the classical kNN.

- We also proposed another variation of kNN that handles high-dimensional data streams using the UMAP [26] to reduce the input space dimension (**Q2**). UMAP has no incremental version that is able to process data streams. This is because of its transductive nature, demanding the presence of the entire data for each new incoming observation. In this contribution (presented in Chapter 6), we proposed a batch-incremental UMAP-kNN that pre-processes the data in a batch-incremental manner and fed them to the kNN for prediction in an instance-incremental way. This approach provided promising results by reducing the computational cost of the kNN while obtaining good accuracy and being efficient in visualization (**Q1**).

The results of the aforementioned proposed approaches exhibited a parameterized resource-accuracy tradeoff. In fact, small kNN sliding window – or output dimension after projection – can significantly degrade the accuracy, whereas a large window – or space dimension – will lead to a better predictive performance but also amplify the use of computational resources.

7.1.3 Ensembles

Several theoretical and empirical studies have shown that combining multiple individual learners (ensembles) leads to better accuracy. There is no free lunch, ensemble-based methods are very costly (in terms of resources) in comparison to single classifiers – this motivated us to improve the performance of two reputed ensembles, namely the LB and ARF methods, with high-dimensional data streams. In this context, we summarize our ensemble-based contributions as follows:

- We proposed two LB versions that use the CS-kNN, that addresses the **Q1**, **Q2**, and **Q3**, in Chapter 4. The first one, CS-kNN^{LB}, operates using the CS-kNN as a base learner to the LB ensemble with only one CS random matrix (as a filter) for all the ensemble members. Thereafter, we proposed the CSB-kNN method that improves the predictive results of the later by increasing the diversity inside the ensemble using different CS random matrices, one for each ensemble member.

- In chapter 5, we studied a new ensemble method, the ARF, that provides good accuracy in comparison with state-of-the-art ensembles but consumes a lot of computational resource. Similarly to the strategy used in the CSB-kNN, we proposed the CS-ARF method that decreases the computational resource usage (**Q1**) of the adaptive random forest (**Q3**) with high-dimensional data streams (**Q2**). We used different CS matrices, to reduce the input dimensionality, within the different trees used in the ensemble.

7.2 Open Issues and Future Directions

The contribution results presented in this thesis open new perspectives in the stream domain that we explored. We outline with a list of some potentially interesting research directions that directly follow from our work.

SketchNB. In this work, we pre-processed the datasets and transformed numerical attributes into discrete attributes using WEKA [107] because CMS works principally on categorical data. A promising direction to fully handle numerical attributes incrementally is to use a discretization algorithm, such as the *Partition Incremental Discretization (PiD)* [106] as filter. One way to test its feasibility is to develop it as a *Java* implementation under the MOA framework [97].

CS-kNN. Instead of keeping synopsis of data, the kNN algorithm maintains a window that contains a part of the stream making the kNN slow. Thus, the ensemble-based CSB-kNN method, that combines several CS-kNN, is also slow. We suggest that further research should investigate how to optimize the processing time of the CSB-kNN (which already obtains sufficiently good accuracy) and provide guarantees along the number of output dimensions. Once the latter is fixed, we could efficiently project the data streams knowing the input dimensions.

UMAP-kNN. We believe that our promising batch-incremental UMAP-kNN approach could be pursued further to enhance its run-time performance by applying a fast DR before using UMAP. Another area for future work could be to use of a different mechanism, such as the application of UMAP on each incoming data inside a sliding window. We believe that this may be slow but will be suited for instance-incremental learning.

Towards AutoML. There is no doubt, the proposed algorithms mentioned in this thesis are suitable for data streams. In fact, before starting the incremental processing of the stream, we need to fix in advance the parameter(s) for the algorithm being used (e.g., the number of neighbors k for the kNN algorithm, the number of learners for the ensemble-based algorithms, the output dimension m for DR techniques). However, these algorithms are not fully automated because the generated models might change over time and decrease the performance of the corresponding classifier. Consequently, the parameterization fixed at the beginning might not hold for the entire stream. So how could this issue be fixed?

Auto Machine Learning (autoML) is a new topic that addresses the problem of variability by automatic monitoring models. Some systems[1] have been developed recently that allow

[1] https://www.automl.org/automl/.

hyper-parameter tuning, such as AutoWeka[2] and AutoSklearn[3] (i.e., autoML combined with Weka and Scikit-learn, respectivety). On the other hand, very few contributions on AutoML for data stream monitoring are found in the literature. In this context, Veloso, Gama, and Malheiro proposed a Self Parameter Tuning (SPT) technique that ensures a full automation of stream modelling algorithms by continuously searching for the optimal hyper-parameters while processing the stream. This hyper-parameterization could change over time, depending on the current distribution of the stream.

We believe that hyper-parameters tuning and algorithms configuration have the potential to be revolutionary for the data stream mining tasks. Finding the optimal hyper-parameters (e.g., using SPT) for a classification algorithm is tedious, especially that the latter is going to be used in different contexts on different streams. Thus, adjusting the hyper-parameters of stream algorithms automatically and dynamically is a very promising avenue.

[2] https://www.automl.org/automl/autoweka/.

[3] https://www.automl.org/automl/auto-sklearn/.

A

Open Source Contributions

Contents

A.1 Introduction

In this appendix, we present existing tools used in this thesis to apply machine learning to data streams for both research and practical applications. These open-source frameworks are designed to facilitate collaboration among research groups and allow users to implement their own – or extend existing – algorithms, and to compare against state-of-the-art algorithms already implemented. In the following, we present the two main data stream mining frameworks[1] used to implement and evaluate our contributions. We describe the installation process and some implementation details used in order to test our new classification approaches proposed throughout this thesis.

A.2 Massive Online Analysis

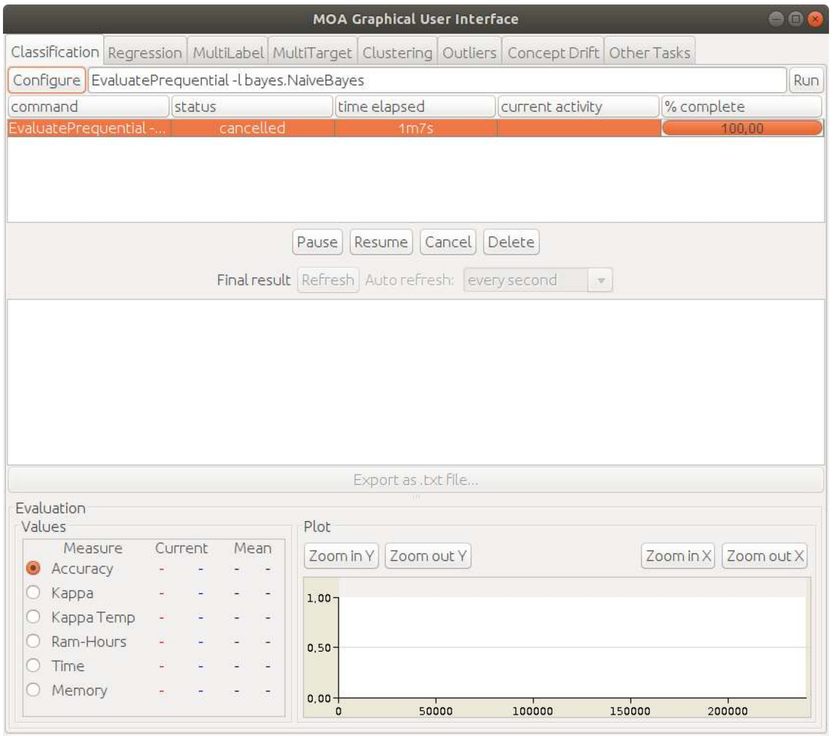
Massive Online Analysis (MOA)[2] [97] is the most popular open source framework for data stream analysis, implemented in Java and developed on top of WEKA [107], with a very active

[1] Requirements of such frameworks are introduced in Part I of the thesis.

[2] `https://moa.cms.waikato.ac.nz/`.

community. It provides data generators (e.g, waveform, random tree, and SEA generators), evaluation methods (e.g., prequential and interleaved test-then-train evaluations), statistics (e.g., running time, memory), and algorithms for data mining tasks (e.g., classification, regression, clustering) for evolving data streams. To create a new classifier, users only need to extend an abstract class, called `moa.classifiers.AbstractClassifier`, and implement the desired algorithm.

MOA can be run on Windows, Mac, and Unix/Linux systems and used through a user interface (See Figure A.1) or a command line. A recent book [37] discusses the MOA software, how to use it, and also includes exercises and lab sessions. In the following, we discuss the components added concerning our contributions to the MOA framework.



EvaluatePrequential -l bayes.NaiveBayes

command	status	time elapsed	current activity	% complete
EvaluatePrequential -...	cancelled	1m7s		100,00

	Measure	Current		Mean	
●	Accuracy	-	-	-	-
○	Kappa	-	-	-	-
○	Kappa Temp	-	-	-	-
○	Ram-Hours	-	-	-	-
○	Time	-	-	-	-
○	Memory	-	-	-	-

Figure A.1: MOA main window.

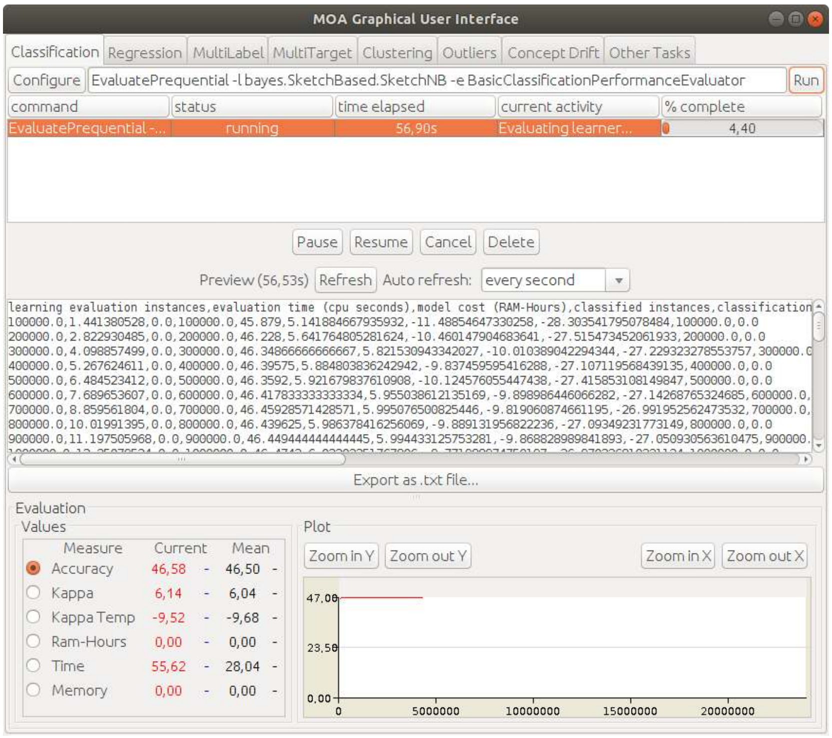
Sketch Naive Bayes

We used the MOA framework to implement our sketch-based NB approaches that can be run using the codes and instructions in the following GitHub repository `https://github.com/marouababahri/SketchNB`. The basic SketchNB classifier is implemented, by following the pseudo-code listed in Algorithms 1 and 2 (Chapter 3), in the file *SketchNB.java* placed in the `moa.classifiers` package of the MOA framework. SketchNB is available from the learner selection dialog in the MOA graphical interface (Figure A.2).

The basic parameters that can be set in the SketchNB classifier are the following:

- -d: delta to fix the depth of the sketch table.

- -e: epsilon to fix the width of the sketch table.
- -C: constant to adjust the size of the sketch.



EvaluatePrequential -l bayes.SketchBased.SketchNB -e BasicClassificationPerformanceEvaluator

command	status	time elapsed	current activity	% complete
EvaluatePrequential -...	running	56,90s	Evaluating learner...	4,40

Preview (56,53s) Auto refresh: every second

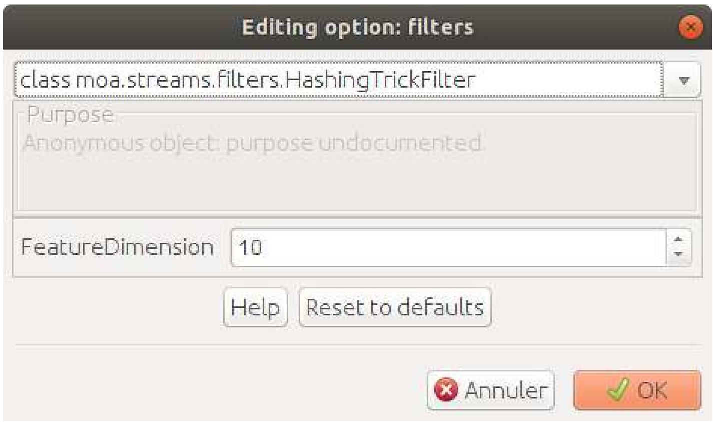
```
learning evaluation instances,evaluation time (cpu seconds),model cost (RAM-Hours),classified instances,classification
100000.0,1.441380528,0.0,100000.0,45.879,5.141884667935932,-11.48854647330258,-28.303541795078484,100000.0,0.0
200000.0,2.822930485,0.0,200000.0,46.228,5.641764805281624,-10.460147904683641,-27.515473452061933,200000.0,0.0
300000.0,4.098857499,0.0,300000.0,46.34866666666667,5.821530943342027,-10.010389042294344,-27.229323278553757,300000.0
400000.0,5.267624611,0.0,400000.0,46.39575,5.884803836242942,-9.837459595416288,-27.107119568439135,400000.0,0.0
500000.0,6.484523412,0.0,500000.0,46.3592,5.921679837610908,-10.124576055447438,-27.415853108149847,500000.0,0.0
600000.0,7.689653607,0.0,600000.0,46.417833333333334,5.955038612135169,-9.898986446066282,-27.14268765324685,600000.0,
700000.0,8.859561804,0.0,700000.0,46.45928571428571,5.995076500825446,-9.819060874661195,-26.991952562473532,700000.0,
800000.0,10.01991395,0.0,800000.0,46.439625,5.986378416256069,-9.889131956822236,-27.09349231773149,800000.0,0.0
900000.0,11.197505968,0.0,900000.0,46.449444444444445,5.994433125753281,-9.868828989841893,-27.050930563610475,900000.
```

	Measure	Current		Mean	
●	Accuracy	46,58	-	46,50	-
○	Kappa	6,14	-	6,04	-
○	Kappa Temp	-9,52	-	-9,68	-
○	Ram-Hours	0,00	-	0,00	-
○	Time	55,62	-	28,04	-
○	Memory	0,00	-	0,00	-

Figure A.2: SketchNB classifier.

The SketchNB$_{HT}$ uses the binary hashing trick filter implemented in the class *HashingTrickFilter.java* that minimizes the space size before the learning phase with the SketchNB. This classifier, in addition to the aforementioned parameters, needs to fix the output dimensionality after projection (see also Figure A.3):

- -d: the output dimensionality.



class moa.streams.filters.HashingTrickFilter

Purpose
Anonymous object: purpose undocumented.

FeatureDimension 10

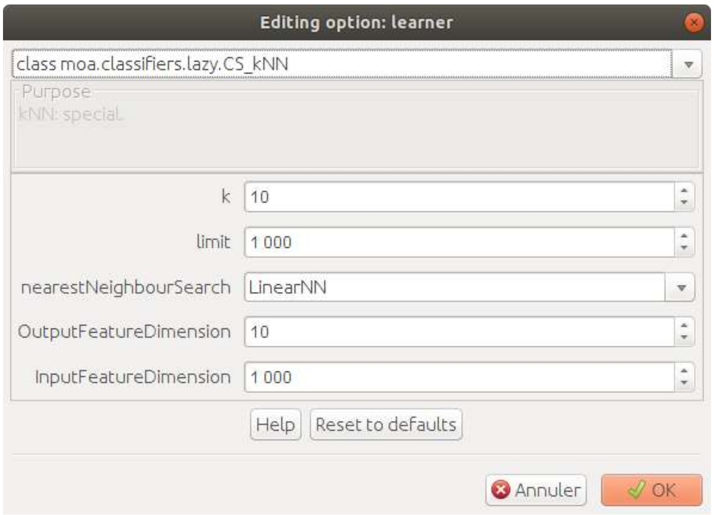
Figure A.3: Hashing trick filter.

Compression-Based Algorithms

We implemented, in the file *CS-Filter.java* placed in the `moa.streams.filters` package, the compressed sensing technique (based on Gaussian random projection) under MOA (Figure A.4). This technique can be used in conjunction with any data mining algorithm as a filter.

We extended the AbstractClassifier class to build our proposed CS-kNN algorithm that uses the CS internally, i.e., there is no need to couple it with the CS-filter. All source codes and datasets employed in our analysis are available at `https://github.com/marouabahri/CS-kNN`. The main parameters that must be defined are the following:

- -k: the number of neighbors.
- -w: the maximum number of instances to store inside the sliding window.
- -d: the output dimensionality.



class moa.classifiers.lazy.CS_kNN

kNN: special.

k 10

limit 1 000

nearestNeighbourSearch LinearNN

OutputFeatureDimension 10

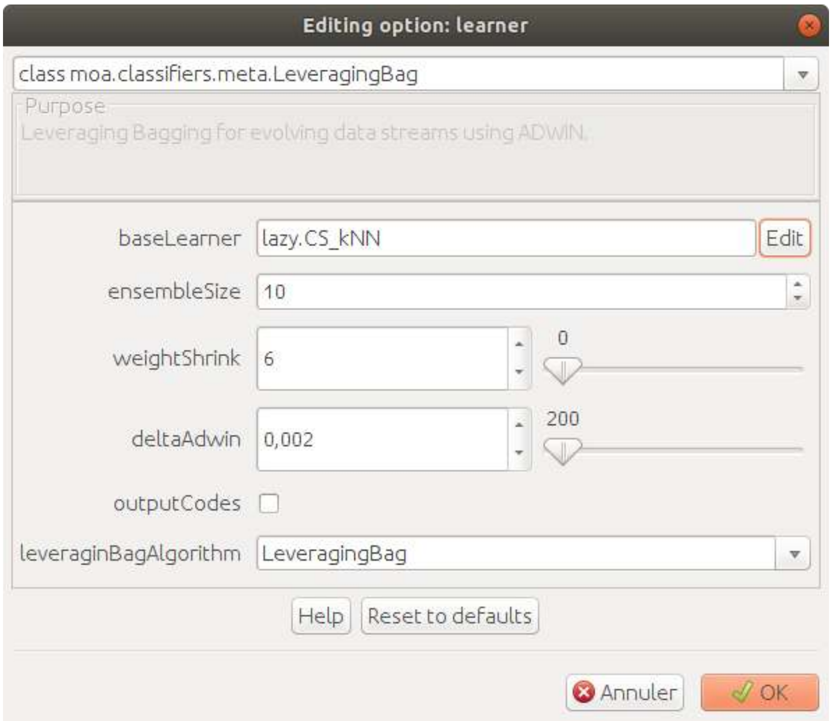
InputFeatureDimension 1 000

Figure A.4: CS-kNN algorithm.

To use the ensemble-based method, CSB-kNN, we use the *LeveragingBag.java* class and select the CS-kNN as a base learner (Figure A.5). The crucial parameter that needs to be fixed in the CSB-kNN is:

- -s: the ensemble size, i.e., the number of CS-kNN inside the ensemble.

A similar strategy has been proposed based on the adaptive random forest, CS-ARF, where we updated the tree-based model used with ARF and coupled it with the CS technique. The codes and datasets used to evaluate the CS-ARF are available at `https://github.com/marouabahri/CS-ARF`.



Editing option: learner

class moa.classifiers.meta.LeveragingBag

Purpose

Leveraging Bagging for evolving data streams using ADWIN.

baseLearner: lazy.CS_kNN

ensembleSize: 10

weightShrink: 6

deltaAdwin: 0,002

outputCodes: □

leveraginBagAlgorithm: LeveragingBag

Figure A.5: CSB-kNN algorithm.

A.3 Scikit-Multiflow

Scikit-multiflow[3] [135] is a new framework inspired by the popular frameworks *scikit-learn*[4] [134] and MOA [97] that fills the void in *Python* for stream learning tasks. It contains, *inter alia*, stream generators, learning and evaluation methods as in MOA.

UMAP-kNN

We implemented our UMAP-kNN using the scikit-multiflow because of two reasons; (i) the only available implementation of UMAP is in Python; and (ii) to promote this new promising open source framework. The UMAP-kNN is implemented in the file *batchIncrementalUMAPkNN.py* placed in the skmultiflow.lazy package of scikit-multiflow framework. The materiel used in our analysis is available in the GitHub repository https://github.com/marouabahri/UMAP-kNN.

The parameters that need to be fixed are the following:

- -k: the number of neighbors.
- -w: the maximum number of instances to store inside the sliding window.
- -d: the output dimensionality.

[3] https://scikit-multiflow.github.io/.
[4] https://scikit-learn.org/stable/.

- -batch: the batch size.

Other parameters related to the UMAP technique could be changed. In fact, UMAP is a graph-based technique, so an important parameter is the number of neighbors (*n-neighbors*) that controls how UMAP neighborhood structure in the data will be (local versus global structure) when attempting to project the data. Thus a low values of n-neighbors will lead to a local structure preservation, while with large values, larger neighborhoods preservation will be assured using UMAP.

A.4 Conclusion

In this appendix, we presented the two open-source software libraries for data stream mining that we used for the implementation, evaluation, and comparison of our contributions in this thesis. We also presented our work and specified the parameters that need to be fixed for each algorithm. The source code and datasets used throughout the thesis are provided at the following address `https://github.com/marouabahri`.

B

Résumé en Français

Contents

B.1 Contexte et Motivation

Ces dernières décennies ont connu une révolution technologique omniprésente qui envahit notre vie à toutes les échelles. Cette montée technologique vertigineuse inclut, de plus en plus, de systèmes et d'applications qui génèrent continuellement de grandes quantités de données connues sous le nom de *flux de données*. Un exemple d'application est l'Internet des Objets (IdO) qui est défini comme un vaste réseau de dispositifs (objets) physiques et de capteurs qui connectent, interagissent et échangent des données. L'IdO est un élément-clé de l'automatisation du quotidien, par exemple les voitures, les drones, les avions et la domotique. Ces dispositifs créent et continueront de créer de manière exponentielle une quantité massive de données à cause des flux générés en temps réel. D'ici la fin de 2020, 31 milliards de tels dispositifs seront connectés dans le monde entier, et vers 2025 ce nombre devrait augmenter à environ 75 milliards, selon Statista[1]. Dans ce contexte, plusieurs méthodes et applications doivent être explorées pour faire face à ces données volumineuses qui sont caractérisées par les 3V: volume, vélocité et variabilité.

Le succès de l'IdO est lié à sa capacité à extraire des connaissances utiles en acquérant automatiquement les informations cachées dans le vaste volume de données générées au fil

[1] www.statista.com/statistics/976079/number-of-IdO-connected-objects-worldwide-by-type/.

du temps. Les tâches typiques d'exploration de données qui ont été étudiées en profondeur au cours des dernières années comprennent la classification, la régression et le clustering. En effet, les approches traditionnelles proposées pour les données statiques ont certaines limites lorsqu'elles sont appliquées aux flux de données dynamiques. Par conséquent, de nouvelles approches et techniques d'exploration de données sont nécessaires pour traiter les flux de données. Dans ce contexte de l'IdO, les approches d'exploration de données devraient être capables de gérer la vitesse et l'infinité des flux de données en utilisant des ressources limitées – en termes de temps d'exécution et de mémoire. Plus de détails sur ces contraintes sont présentés dans la section B.2. Pour extraire des connaissances utiles de ces données, nous utilisons généralement des algorithmes d'*apprentissage automatique* adaptés au cadre incrémental des flux de données [3, 4]. Dans ce contexte, la tâche de l'exploration de ces données est devenue indispensable dans de nombreuses applications du monde réel. Cette catégorie d'applications génère souvent des données à partir de flux en constante évolution et exige un traitement en temps réel.

Dans le domaine d'analyse de données, la ***classification*** est l'une des tâches les plus utilisées. Elle tente de prédire les catégories – ou les classes – des observations non-libellées en utilisant un modèle construit des données déjà traitées. La classification de flux est considérée comme une application de recherche active dans le domaine de l'exploration de flux de données, où l'accent est mis sur le développement de nouveaux algorithmes – ou d'améliorer les algorithmes existants [5]. Il existe un certain nombre de *classificateurs* qui sont largement utilisés dans l'exploration de données et sont appliqués dans plusieurs applications réelles, telles que les arbres de décision, les réseaux de neurones, les k plus proches voisins, les réseaux bayésiens, etc. Le Chapitre 2 couvre, *entre autres*, une étude de l'état de l'art sur les algorithmes de classification pour les flux de données les plus connus et récents.

B.2 Défis

Comme mentionné ci-dessus, la classification des flux de données vise à prédire les classes de nouvelles instances non-libellées qui arrivent constamment. Après la prédiction, les modèles existants vont être mis à jour continuellement au fur et à mesure de l'évolution du flux pour suivre la distribution actuelle des données. La nature massive et potentiellement infinie des flux, qui soulève des problèmes critiques et fait échouer les algorithmes traditionnels d'exploration de données, impose des contraintes pour gérer convenablement le comportement dynamique et incrémental des flux.

Bien que les contraintes suivantes soient communes entre les différentes applications d'exploration de flux de données, nous abordons ces exigences dans le contexte de la classification [6, 7, 8].

- **Nature évolutive des flux de données.** Tout algorithme de classification doit tenir compte de l'évolution considérable des données et s'adapter à leur nature incrémentale à grande vitesse. Ainsi, les algorithmes doivent classer de manière séquentielle et incrémentale les instances récentes.

- **Temps d'exécution.** Un algorithme devrait traiter rapidement les instances entrantes provenant du flux, car plus l'algorithme est lent, moins il sera efficace pour les applications qui nécessitent un traitement en temps réel.

- **Mémoire illimitée.** En raison des quantités énormes des flux qui exigent une mémoire illimitée pour leur traitement, tout classificateur devrait avoir la capacité de fonctionner avec une mémoire limitée en gardant des synopsis de données sur les instances traitées et les modèles actuels.

- **Flux de données en grande dimension.** Les flux de données peuvent être de grande dimension, tels que les documents texte. Pour ce type de données, les distances entre les instances augmentent de façon exponentielle, ce qui peut potentiellement avoir un impact sur les performances de n'importe quel classificateur.

- **Dérive conceptuelle.** La distribution des données peut changer à tout moment, ce qui induit un phénomène connu sous le nom de *dérive conceptuelle* ou *concept drifts* en anglais. La dérive conceptuelle peut impacter les résultats du classificateur au fil du temps, notamment sa qualité prédictive. Cependant, pour faire face aux nouvelles directions des données qui doivent être détectées en même temps que leur apparition, un mécanisme de détection de dérive est généralement associé aux algorithmes de classification des flux de données. Nous dirigeons le lecteur vers [9] pour une étude détaillée sur ce phénomène.

Dans le cadre de cette thèse, une question cruciale se pose sur la manière avec laquelle on traite des données potentiellement infinies tout en addressant leurs défis à *moindres coûts*.

Ces défis susmentionnés sont d'une grande importance dans la tâche de classification des flux. Nous remarquons que les techniques d'exploration de flux doivent être différentes des techniques traditionnelles pour les bases de données statiques. Pour relever ces défis, les algorithmes de classification doivent incorporer une stratégie incrémentale qui permet d'assurer le bon déroulement du traitement des flux de données sous les contraintes présentées dans la section 2.2. Le tableau B.1 présente une comparaison des environnements pour les données statiques et les flux (données dynamiques) [10].

En plus du volume énorme de données, leur *dimension* augmente considérablement et pose un défi notable dans de nombreux domaines, tels que la biologie (données omiques [2]) [11, 12] et le filtrage des e-mails [13] (classer un e-mail comme spam ou non en fonction

[2] Les données omiques se réfèrent aux données biologiques se terminant par -omique, par exemple, génomique, métabolomique.

Table B.1: Comparaison entre les données statiques et les flux.

	Données statiques	**Flux de données**
Accès	aléatoire	séquentiel
Nombre de passes	plusieurs passes	passe unique
Temps d'exécution	illimité	limité
Mémoire	illimitée	limitée
Type de résultat	précis	approximatif
Environnement	statique	dynamique

du contenu de celui-ci). Ces données de grande dimension contiennent généralement de nombreux attributs redondants ou non-pertinents qui peuvent être réduits à un ensemble plus petit de combinaisons pertinentes extraites de l'ensemble d'attributs d'entrée sans perte d'informations significatives.

Afin de traiter ce type de données de manière optimale à moindre coût, une étape de prétraitement est impérative pour filtrer les attributs pertinents et donc permettre des économies en termes de ressources avec des algorithmes d'exploration de flux de données. Pour ce faire, des synopsis peuvent être construits à partir d'instances de flux à l'aide des techniques de ***réduction*** (par exemple, *résumés minimalistes* en conservant les fréquences des données), en sélectionnant une partie des données entrantes sans réduire la dimension (l'échantillonnage), ou en appliquant une technique de Réduction de Dimensionnalité (RD) pour réduire le nombre d'attributs. Naturellement, le choix de la technique appropriée dépend du problème à résoudre [14].

Notre objectif dans cette thèse est motivé par les critères décrits ci-dessus pour l'exploration de flux de données. Nous concentrons principalement sur la tâche de classification et visons à développer de nouvelles approches de classification pour améliorer les performances des algorithmes existants en utilisant des techniques de réduction de données.

La réduction de dimensionnalité est définie comme la projection de données de haute dimension dans un espace de basse dimension en réduisant les attributs d'entrée aux plus pertinents. En effet, la RD est un processus crucial pour éviter la *malédiction de la dimension* – qui peut augmenter l'utilisation des ressources de calcul et affecter négativement les performances prédictives de tout algorithme d'exploration de données. Pour minimiser ces impacts, plusieurs techniques de réduction ont été proposées et largement étudiées dans le cadre statique [15, 16] pour gérer des données de grande dimension. Cependant, ces techniques ne respectent pas les exigences en termes de ressources de calcul des flux de données [17, 18]. Plus de détails sont fournis dans le Chapitre 2.

B.3 Contributions

La préoccupation principale dans cette thèse aborde les problèmes susmentionnés qui concernent les performances des algorithmes d'exploration des flux de données. Cette thèse contribue à ce domaine en introduisant et en explorant de nouvelles approches qui réduisent les ressources utilisées par les algorithmes existants tout en sacrifiant un minimum de précision.

Au cours de cette introduction, nous divisons l'objectif de la thèse en trois questions de recherche principales:

- **Q1**: *Comment pouvons-nous améliorer les performances des classificateurs existants en termes de ressources tout en conservant une bonne précision ?*
- **Q2**: *Pouvons-nous faire mieux en prétraitant dynamiquement les flux de données de grande dimension ?*
- **Q3**: *Comment pouvons-nous traiter les dérives conceptuelles où les modèles actuels ne sont plus représentatifs ?*

Dans ce qui suit, nous résumons brièvement nos contributions:

- Dans le **Chapitre 3**, nous visons à améliorer les performances du classificateur bayesien naïf en développant trois nouvelles approches pour le rendre efficace et efficient avec des données de haute dimension.
  - Nous étudions une structure de données efficace, appelée *Count-Min Sketch (CMS)* [19], pour maintenir des synopsis (fréquences) de données en mémoire.
  - Nous proposons un nouveau bayesien naïf, basé sur les résumés minimalises, qui utilise CMS pour stocker des informations provenant du flux de données dans une mémoire de taille fixe.
  - Des preuves théoriques sur la taille de la table CMS sont fournies en adaptant les garanties de la technique CMS au classificateur bayesien naïf.
  - Pour gérer les données de haute dimensionnalité, nous ajoutons une étape de prétraitement incrémentale au cours de laquelle les données seront compressées à l'aide d'une technique de RD rapide, telle que le *hachage* [20].
  - Nous incorporons dans la phase d'apprentissage une stratégie adaptative qui utilise ADaptive WINdowing (ADWIN) [21], un détecteur de dérive conceptuelle, afin de s'adapter aux changements dans la distribution.
- Dans le **Chapitre 4**, nous étudions l'algorithme des k plus proches voisins (kNN) [22]. Ainsi, nous proposons deux approches qui visent à améliorer les coûts de calcul de kNN

avec les flux de données de grande dimension en explorant la technique *Compressed Sensing (CS)* [23] pour réduire la taille de l'espace d'entrée.

  - Nous proposons un nouvel algorithme pour la classification des flux de données, CS-kNN. Notre objectif principal consiste à améliorer l'utilisation des ressources de kNN en compressant les flux d'entrée à l'aide de CS avant d'appliquer la tâche de classification. Cela entraîne une réduction considèrable en resources (mémoire et temps d'exécution) sollicitées par kNN.
  - Nous fournissons des preuves théoriques sur la préservation du voisinage avant et après la projection en utilisant la technique CS. Par conséquent, nous nous assurons que la performance prédictive (précision) de CS-kNN est presque la même que celle qui aurait pu être obtenue avec le kNN standard (en utilisant les données d'entrée de haute dimension, sans CS).
  - Nous proposons également une méthode d'ensemble basée sur Leveraging Bagging [24] où nous combinons les résultats de plusieurs CS-kNN pour améliorer la précision d'un classificateur unique.

- Dans le **Chapitre 5**, nous visons à améliorer les performances de la nouvelle méthode d'ensemble performante, Adaptive Random Forest (ARF) [25] avec les données de haute dimension.
  - Nous proposons un nouvel ensemble qui vise à minimiser l'utilisation des ressources de la méthode ARF en réduisant la dimension des données d'entrée. Pour cela, nous utilisons la technique CS en interne qui diminue la taille des données ensuite les transmet aux membres de l'ensemble.

- Dans le **Chapitre 6**, nous explorons une nouvelle technique de RD qui a récemment attiré beaucoup d'attention grâce à ses hautes performances: UMAP (Uniform Manifold Approximation and Projection) [26]. Nous utilisons cette technique, qui conserve le voisinage, pour prétraiter les données afin d'améliorer les résultats de l'algorithme kNN, basé également sur le voisinage.
  - Nous proposons une technique d'apprentissage incrémental par lots (batches): une adaptation de UMAP pour les flux de données évolutifs. Au lieu d'appliquer UMAP sur un ensemble de données statiques, nous l'adaptons en utilisant des mini-batches du flux de manière incrémentale.
  - Nous proposons également un nouvel algorithme incrémental par batches, UMAP-kNN, pour la classification de flux en utilisant UMAP. L'idée principale est d'appliquer kNN sur des mini-batches de données de petite dimension obtenues de l'étape de prétraitement.

References

[1] Jacques Bughin, Eric Hazan, Sree Ramaswamy, Michael Chui, Tera Allas, Peter Dahlstrom, Nicolaus Henke, and Monica Trench. *Artificial intelligence: the next digital frontier?* McKinsey Global Institute Report, 2017.

[2] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. “Mining data streams: a review”. In: *SIGMOD* 34.2 (2005), pp. 18–26.

[4] Charu C Aggarwal. *Data streams: models and algorithms*. Vol. 31. Springer, 2007.

[5] David J Hand, Heikki Mannila, and Padhraic Smyth. *Principles of data mining*. MIT press, 2001.

[6] Charu C Aggarwal and S Yu Philip. “A survey of synopsis construction in data streams”. In: *Data Streams*. Springer, 2007, pp. 169–207.

[7] João Gama and Mohamed Medhat Gaber. *Learning from data streams: processing techniques in sensor networks*. Springer, 2007.

[8] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[9] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. “A survey on concept drift adaptation”. In: *Computing Surveys (CSUR)* 46.4 (2014), p. 44.

[10] Mohamed Medhat Gaber, João Gama, Shonali Krishnaswamy, João Bártolo Gomes, and Frederic Stahl. “Data stream mining in ubiquitous environments: state-of-the-art and current directions”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4.2 (2014), pp. 116–138.

[11] Luca Albergante, Jonathan Bac, and Andrei Zinovyev. “Estimating the effective dimension of large biological datasets using Fisher separability analysis”. In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[12] Chunming Xu and Scott A Jackson. *Machine learning and complex biological data*. 2019.

[13] Tiago A Almeida, Jurandy Almeida, and Akebo Yamakami. “Spam filtering: how the dimensionality reduction affects the accuracy of Naive Bayes classifiers”. In: *Journal of Internet Services and Applications* 1.3 (2011), pp. 183–200.

[14] John P Cunningham and Zoubin Ghahramani. “Linear dimensionality reduction: Survey, insights, and generalizations”. In: *Journal of Machine Learning Research (JMLR)* 16.1 (2015), pp. 2859–2900.

[15] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. “Dimensionality reduction: a comparative”. In: *Journal of Machine Learning Research (JMLR)* 10.66-71 (2009), p. 13.

[16] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. “A survey of dimensionality reduction techniques”. In: *arXiv preprint arXiv:1403.2877* (2014).

[17] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. “Issues in evaluation of stream learning algorithms”. In: *SIGKDD*. ACM. 2009, pp. 329–338.

[18] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. “Machine learning for streaming data: state of the art, challenges, and opportunities”. In: *ACM SIGKDD Explorations Newsletter* 21.2 (2019), pp. 6–22.

[19] Graham Cormode and Shan Muthukrishnan. “An improved data stream summary: the count-min sketch and its applications”. In: *Journal of Algorithms* 55.1 (2005), pp. 58–75.

[20] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. “Feature hashing for large scale multitask learning”. In: *International Conference on Machine Learning (ICML)*. ACM. 2009, pp. 1113–1120.

[21] Albert Bifet and Ricard Gavalda. “Learning from time-changing data with adaptive windowing”. In: *International Conference on Data Mining (ICDM)*. SIAM. 2007, pp. 443–448.

[22] Naomi S Altman. “An introduction to kernel and nearest-neighbor nonparametric regression”. In: *The American Statistician* 46.3 (1992), pp. 175–185.

[23] David L Donoho. “Compressed sensing”. In: *IEEE Transactions on Information Theory* 52.4 (2006), pp. 1289–1306.

[24] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. “Leveraging bagging for evolving data streams”. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2010, pp. 135–150.

[25] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrıcio Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. “Adaptive random forests for evolving data stream classification”. In: *Machine Learning* (2017), pp. 1–27.

[26] Leland McInnes, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426* (2018).

[27] Maroua Bahri, Albert Bifet, João Gama, Silviu Maniu, and Heitor Murilo Gomes. *Data stream analysis: foundations, progress in classification and tools.* submitted.

[28] Maroua Bahri, Albert Bifet, Silviu Maniu, and Heitor Murilo Gomes. "Survey on feature transformation techniques for data streams". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2020.

[29] Elena Ikonomovska, Suzana Loskovska, and Dejan Gjorgjevik. "A survey of stream data mining". In: *National Conference with International participation ETAI*. 2007, pp. 19–21.

[30] Willie Ng and Manoranjan Dash. "Discovery of frequent patterns in transactional data streams". In: *Transactions on large-scale data-and knowledge-centered systems II*. Springer, 2010, pp. 1–30.

[31] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. "Models and issues in data stream systems". In: *ACM SIGMOD*. ACM. 2002, pp. 1–16.

[32] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. "Querying and mining data streams: you only get one look a tutorial". In: *ACM SIGMOD*. 2002, pp. 635–635.

[33] Gurmeet Singh Manku and Rajeev Motwani. "Approximate frequency counts over data streams". In: *Very Large Data Bases (VLDB)*. Elsevier. 2002, pp. 346–357.

[34] Burton H Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[35] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. "New ensemble methods for evolving data streams". In: *SIGKDD*. ACM. 2009, pp. 139–148.

[36] Albert Bifet and Richard Kirkby. "Data stream mining a practical approach". In: (2009).

[37] Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer. *Machine learning for data streams: with practical examples in MOA*. MIT Press, 2018.

[38] Nir Friedman, Dan Geiger, and Moises Goldszmidt. "Bayesian network classifiers". In: *Machine learning* 29.2-3 (1997), pp. 131–163.

[39] Albert Bifet, Bernhard Pfahringer, Jesse Read, and Geoff Holmes. "Efficient data stream classification via probabilistic adaptive windows". In: *Symposium On Applied Computing (SIGAPP)*. ACM. 2013, pp. 801–806.

[40] Viktor Losing, Barbara Hammer, and Heiko Wersing. "KNN classifier with self adjusting memory for heterogeneous concept drift". In: *International Conference on Data Mining (ICDM)*. IEEE. 2016, pp. 291–300.

[41] Pedro Domingos and Geoff Hulten. "Mining high-speed data streams". In: *SIGKDD*. ACM. 2000, pp. 71–80.

[42] João Gama, Ricardo Rocha, and Pedro Medas. "Accurate decision trees for mining high-speed data streams". In: *SIGKDD*. ACM. 2003, pp. 523–528.

[43] João Gama, Ricardo Fernandes, and Ricardo Rocha. "Decision trees for mining data streams". In: *Intelligent Data Analysis (IDA)* 10.1 (2006), pp. 23–45.

[44] Albert Bifet and Ricard Gavaldà. "Adaptive learning from evolving data streams". In: *Intelligent Data Analysis (IDA)*. Springer. 2009, pp. 249–260.

[45] Robi Polikar. "Ensemble based systems in decision making". In: *IEEE Circuits and Systems Magazine* 6.3 (2006), pp. 21–45.

[46] Leo Breiman. "Bagging predictors". In: *ML* 24.2 (1996), pp. 123–140.

[47] Thomas G Dietterich. "Ensemble methods in machine learning". In: *International Workshop on Multiple Classifier Systems*. Springer. 2000, pp. 1–15.

[48] Heitor Murilo Gomes, Jean Paul Barddal, Fabrıcio Enembreck, and Albert Bifet. "A survey on ensemble learning for data stream classification". In: *Computing Surveys (CSUR)* 50.2 (2017), p. 23.

[49] Nikunj C Oza. "Online bagging and boosting". In: *IEEE International Conference on Systems, Man and Cybernetics*. Vol. 3. Ieee. 2005, pp. 2340–2345.

[50] Heitor Murilo Gomes, Jesse Read, and Albert Bifet. "Streaming random patches for evolving data stream classification". In: *International Conference on Data Mining (ICDM)*. IEEE. 2019.

[51] Huan Liu and Hiroshi Motoda. *Computational methods of feature selection*. CRC Press, 2007.

[52] Huan Liu and Hiroshi Motoda. *Feature extraction, construction and selection: A data mining perspective*. Vol. 453. Springer Science & Business Media, 1998.

[53] Jean Paul Barddal, Heitor Murilo Gomes, Fabrıcio Enembreck, and Bernhard Pfahringer. "A survey on feature drift adaptation: Definition, benchmark, challenges and future directions". In: *Journal of Systems and Software* 127 (2017), pp. 278–294.

[54] Sergio Ramırez-Gallego, Bartosz Krawczyk, Salvador Garcıa, Michał Woźniak, and Francisco Herrera. "A survey on data preprocessing for data stream mining: Current status and future directions". In: *Neurocomputing* 239 (2017), pp. 39–57.

[55] Xuegang Hu, Peng Zhou, Peipei Li, Jing Wang, and Xindong Wu. "A survey on online feature selection with streaming features". In: *Frontiers of Computer Science* 12.3 (2018), pp. 479–493.

[56] Noura AlNuaimi, Mohammad Mehedy Masud, Mohamed Adel Serhani, and Nazar Zaki. "Streaming feature selection algorithms for big data: A survey". In: *Applied Computing and Informatics* (2019).

[57] Matej Artac, Matjaz Jogan, and Ales Leonardis. "Incremental PCA for on-line visual learning and recognition". In: *Object Recognition Supported by User interaction for Service Robots*. Vol. 3. IEEE. 2002, pp. 781–784.

[58] Juyang Weng, Yilu Zhang, and Wey-Shiuan Hwang. "Candid covariance-free incremental principal component analysis". In: *Transactions on Pattern Analysis and Machine Intelligence* 25.8 (2003), pp. 1034–1040.

[59] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. "Incremental learning for robust visual tracking". In: *International Journal of Computer Vision (IJCV)* 77.1-3 (2008), pp. 125–141.

[60] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. "Memory limited, streaming PCA". In: *Neural Information Processing Systems (NIPS)*. 2013, pp. 2886–2894.

[61] Wenjian Yu, Yu Gu, Jian Li, Shenghua Liu, and Yaohang Li. "Single-pass PCA of large high-dimensional data". In: *International Joint Conference on Artificial Intelligence (IJCAI)* (2017).

[62] Wu Feng, Zhong Yan, Li Ai-ping, and Wu Quan-yuan. "Online classification algorithm for data streams based on fast iterative Kernel principal component analysis". In: *International Conference on Natural Computation (ICNC)*. Vol. 1. IEEE. 2009, pp. 232–236.

[63] Simon Günter, Nicol N Schraudolph, and SVN Vishwanathan. "Fast iterative kernel principal component analysis". In: *Journal of Machine Learning Research (JMLR)* 8.8 (2007), pp. 1893–1918.

[64] Hervé Cardot and David Degras. "Online principal component analysis in high dimension: Which algorithm to choose?" In: *International Statistical Review* 86.1 (2018), pp. 29–50.

[65] Florian Wickelmaier. "An introduction to MDS". In: *Sound Quality Research Unit* 46.5 (2003), pp. 1–26.

[66] Arvind Agarwal, Jeff M Phillips, Hal Daumé III, and Suresh Venkatasubramanian. "Incremental multidimensional scaling". In: *The Learning Workshop*. Citeseer. 2010, pp. 131, 173, 227.

[67] Xi Zhang, Hao Huang, Klaus Mueller, and Shinjae Yoo. “Streaming classical multidimensional scaling”. In: *New York Scientific Data Summit (NYSDS)*. IEEE. 2018, pp. 1–2.

[68] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. “Greedy layer-wise training of deep networks”. In: *Neural Information Processing Systems (NIPS)*. 2007, pp. 153–160.

[69] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. “Extracting and composing robust features with denoising autoencoders”. In: *International Conference on Machine Learning (ICML)*. ACM. 2008, pp. 1096–1103.

[70] Guanyu Zhou, Kihyuk Sohn, and Honglak Lee. “Online incremental feature learning with denoising autoencoders”. In: *Artificial Intelligence and Statistics*. 2012, pp. 1453–1461.

[71] Yue Dong and Nathalie Japkowicz. “Threaded ensembles of supervised and unsupervised neural networks for stream learning”. In: *Canadian AI*. Springer. 2016, pp. 304–315.

[72] Geoffrey J McLachlan. *Discriminant analysis and statistical pattern recognition*. Vol. 544. Wiley, 2004.

[73] Ahsanul Haque, Latifur Khan, Michael Baron, Bhavani Thuraisingham, and Charu Aggarwal. “Efficient handling of concept drift and concept evolution over stream data”. In: *International Conference on Data Engineering (ICDE)*. IEEE. 2016, pp. 481–492.

[74] Shaoning Pang, Seiichi Ozawa, and Nikola Kasabov. “Incremental linear discriminant analysis for classification of data streams”. In: *Transactions on Systems, Man, and Cybernetics* 35.5 (2005), pp. 905–914.

[75] Jieping Ye, Qi Li, Hui Xiong, Haesun Park, Ravi Janardan, and Vipin Kumar. “IDR/QR: An incremental dimension reduction algorithm via QR decomposition”. In: *Transactions on Knowledge and Data Engineering (TKDE)* 17.9 (2005), pp. 1208–1222.

[76] Tae-Kyun Kim, Björn Stenger, Josef Kittler, and Roberto Cipolla. “Incremental linear discriminant analysis using sufficient spanning sets and its applications”. In: *International Journal of Computer Vision (IJCV)* 91.2 (2011), pp. 216–232.

[77] Haifeng Li, Tao Jiang, and Keshu Zhang. “Efficient and robust feature extraction by maximum margin criterion”. In: *Neural Information Processing Systems (NIPS)*. 2004, pp. 97–104.

[78] Jun Yan, Benyu Zhang, Shuicheng Yan, Qiang Yang, Hua Li, Zheng Chen, Wensi Xi, Weiguo Fan, Wei-Ying Ma, and Qiansheng Cheng. “IMMC: incremental maximum margin criterion”. In: *SIGKDD*. ACM. 2004, pp. 725–730.

[79] Santosh S Vempala. *The random projection method.* Vol. 65. American Mathematical Soc, 2005.

[80] Sanjoy Dasgupta and Anupam Gupta. “An elementary proof of the Johnson-Lindenstrauss lemma”. In: *International Computer Science Institute, Technical Report* 22.1 (1999), pp. 1–5.

[81] William B Johnson, Joram Lindenstrauss, and Gideon Schechtman. “Extensions of Lipschitz maps into Banach spaces”. In: *Israel Journal of Mathematics* 54.2 (1986), pp. 129–138.

[82] Chenlu Qiu, Wei Lu, and Namrata Vaswani. “Real-time dynamic MR image reconstruction using Kalman filtered compressed sensing”. In: *International Conference on Acoustics, Speech and Signal Processing.* IEEE. 2009, pp. 393–396.

[83] Hanxi Li, Chunhua Shen, and Qinfeng Shi. “Real-time visual tracking using compressive sensing”. In: *Computer Vision and Pattern Recognition (CVPR).* IEEE. 2011, pp. 1305–1312.

[84] M Uttarakumari, Ashray V Achary, Sujata D Badiger, DS Avinash, Anisha Mukherjee, and Nancy Kothari. “Vehicle classification using compressive sensing”. In: *International Conference on Recent Trends in Electronics, Information and Communication Technology (RTEICT).* IEEE. 2017, pp. 692–696.

[85] Dimitris Achlioptas. “Database-friendly random projections: Johnson-Lindenstrauss with binary coins”. In: *Journal of computer and System Sciences* 66.4 (2003), pp. 671–687.

[86] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. “Locality-sensitive hashing scheme based on p-stable distributions”. In: *Symposium on Computational Geometry (SOCG).* ACM. 2004, pp. 253–262.

[87] Joshua B Tenenbaum, Vin De Silva, and John C Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *science* 290.5500 (2000), pp. 2319–2323.

[88] Martin HC Law, Nan Zhang, and Anil K Jain. “Nonlinear manifold learning for data stream”. In: *International Conference on Data Mining (ICDM).* SIAM. 2004, pp. 33–44.

[89] Frank Schoeneman, Suchismit Mahapatra, Varun Chandola, Nils Napp, and Jaroslaw Zola. “Error metrics for learning reliable manifolds from streaming data”. In: *International Conference on Data Mining (ICDM).* SIAM. 2017, pp. 750–758.

[90] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of Machine Learning Research (JMLR)* 9 (2008), pp. 2579–2605.

[91] A Philip Dawid. “Present position and potential developments: Some personal views statistical theory the prequential approach”. In: *Journal of the Royal Statistical Society: Series A (General)* 147.2 (1984), pp. 278–290.

[92] Zhexue Huang. “Extensions to the k-means algorithm for clustering large data sets with categorical values”. In: *Data Mining and Knowledge Discovery* 2.3 (1998), pp. 283–304.

[93] Maroua Bahri, Silviu Maniu, and Albert Bifet. “Sketch-based naive Bayes algorithms for evolving data streams”. In: *International Conference on Big Data*. IEEE. 2018, pp. 604–613.

[94] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. “Synopses for massive data: Samples, histograms, wavelets, sketches”. In: *Foundations and Trends in Databases* 4.1–3 (2012), pp. 1–294.

[95] Graham Cormode. “Data Sketching”. In: *Queue* 15.2 (2017), p. 60.

[96] Branislav Kveton, Hung Bui, Mohammad Ghavamzadeh, Georgios Theocharous, S Muthukrishnan, and Siqi Sun. “Graphical model sketch”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2016, pp. 81–97.

[97] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. “Moa: Massive online analysis”. In: *Journal of Machine Learning Research (JMLR)* 11.May (2010), pp. 1601–1604.

[98] W Nick Street and YongSeog Kim. “A streaming ensemble algorithm (SEA) for large-scale classification”. In: *SIGKDD*. ACM. 2001, pp. 377–382.

[99] Wei-Yin Loh. “Classification and regression trees”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.1 (2011), pp. 14–23.

[100] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. “Database mining: A performance perspective”. In: *Transactions on Knowledge and Data Engineering (TKDE)* 5.6 (1993), pp. 914–925.

[101] Geoff Hulten, Laurie Spencer, and Pedro Domingos. “Mining time-changing data streams”. In: *SIGKDD*. ACM. 2001, pp. 97–106.

[102] Michael Harries and New South Wales. “Splice-2 comparative evaluation: electricity pricing”. In: (1999).

[103] Bryan Klimt and Yiming Yang. “The enron corpus: A new dataset for email classification research”. In: *European Conference on Machine Learning (ECML)*. Springer. 2004, pp. 217–226.

[104] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. “Learning word vectors for sentiment analysis”. In: *ACL-HLT*. Association for Computational Linguistics. 2011, pp. 142–150.

[105] Patrick Marques Ciarelli and Elias Oliveira. “Agglomeration and elimination of terms for dimensionality reduction”. In: *International Conference on Intelligent Systems Design and Applications*. IEEE. 2009, pp. 547–552.

[106] João Gama and Carlos Pinto. “Discretization from data streams: applications to histograms and data mining”. In: *ACM symposium on Applied computing*. 2006, pp. 662–667.

[107] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. “Weka: A machine learning workbench”. In: *Proceedings of the second Australia and New Zealand Conference on Intelligent Information Systems,* IEEE. 1994, pp. 357–361.

[108] Maroua Bahri, Albert Bifet, Silviu Maniu, Rodrigo de Mello, and Nikolaos Tziortziotis. “Compressed k-nearest neighbors ensembles for evolving data streams”. In: *European Conference on Artificial Intelligence (ECAI)*. IEEE. 2020.

[109] Yair Weiss, Hyun Sung Chang, and William T Freeman. “Learning compressed sensing”. In: *Snowbird Learning Workshop*. Citeseer. 2007.

[110] Jean-Luc Starck, Fionn Murtagh, and Jalal Fadili. *Sparse image and signal processing: Wavelets and related geometric multiscale analysis*. Cambridge university press, 2015.

[111] Alfred M Bruckstein, David L Donoho, and Michael Elad. “From sparse solutions of systems of equations to sparse modeling of signals and images”. In: *SIAM Review* 51.1 (2009), pp. 34–81.

[112] Avishy Y Carmi, Lyudmila Mihaylova, and Simon J Godsill. *Compressed sensing and sparse filtering*. Springer, 2014.

[113] Jesse Read, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. “Batch-incremental versus instance-incremental learning in dynamic and evolving data”. In: *Intelligent Data Analysis (IDA)*. 2012, pp. 313–323.

[114] Youness Arjoune, Naima Kaabouch, Hassan El Ghazi, and Ahmed Tamtaoui. “A performance comparison of measurement matrices in compressive sensing”. In: *International Journal of Communication Systems* 31.10 (2018), e3576.

[115] Thu LN Nguyen and Yoan Shin. “Deterministic sensing matrices in compressive sensing: a survey”. In: *The Scientific World Journal* (2013).

[116] Nikolaos M Freris, Orhan Oçal, and Martin Vetterli. “Compressed sensing of streaming data”. In: *51st Annual Allerton Conference on Communication, Control, and Computing*. IEEE. 2013, pp. 1242–1249.

[117] Richard Baraniuk, Mark Davenport, Ronald DeVore, and Michael Wakin. “A simple proof of the restricted isometry property for random matrices”. In: *Constructive Approximation* 28.3 (2008), pp. 253–263.

[118] Herbert Edelsbrunner and John Harer. “Persistent homology-a survey”. In: *Contemporary Mathematics* 453 (2008), pp. 257–282.

[119] Donald R Sheehy. “The persistent homology of distance functions under random projection”. In: *Symposium on Computational Geometry (SOCG)*. ACM. 2014, p. 328.

[120] Felix Krahmer and Rachel Ward. “New and improved Johnson–Lindenstrauss embeddings via the restricted isometry property”. In: *Mathematical Analysis* 43.3 (2011), pp. 1269–1281.

[121] Ioannis Katakis, Grigorios Tsoumakas, Evangelos Banos, Nick Bassiliades, and Ioannis Vlahavas. “An adaptive personalized news dissemination system”. In: *Journal of Intelligent Information Systems* 32.2 (2009), pp. 191–212.

[122] Maroua Bahri, Heitor Murilo Gomes, Albert Bifet, and Silviu Maniu. “Compressed adaptive random forests for evolving data streams”. In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020.

[123] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[124] Laurent Candillier and Vincent Lemaire. “Design and analysis of the Nomao challenge active learning in the real-world”. In: *ALRA, Workshop ECML-PKDD*. sn. 2012.

[125] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L Reyes-Ortiz. “Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine”. In: *IWAAL*. Springer. 2012, pp. 216–223.

[126] Maroua Bahri, Bernhard Pfahringer, Albert Bifet, and Silviu Maniu. “Efficient batch-incremental classification for evolving data streams”. In: *Intelligent Data Analysis (IDA)*. Springer. 2020.

[127] Frank Klawonn and Plamen Angelov. “Evolving extended naive Bayes classifiers”. In: *International Conference on Data Mining Workshops*. IEEE. 2006, pp. 643–647.

[128] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. Vol. 398. Wiley, 2013.

[129] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *ML* 20.3 (1995), pp. 273–297.

[130] Geoffrey Holmes, Richard Brendon Kirkby, and David Bainbridge. “Batch-incremental learning for mining data streams”. In: (2004).

[131] Harold Hotelling. “Analysis of a complex of statistical variables into principal components.” In: *Journal of Educational Psychology* 24.6 (1933), p. 417.

[132] David W Aha. *Lazy learning*. Springer, 2013.

[133] Charu C Aggarwal. *Data mining: the textbook*. Springer, 2015.

[134] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research (JMLR)* 12.Oct (2011), pp. 2825–2830.

[135] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. "Scikit-multiflow: a multi-output streaming framework". In: *Journal of Machine Learning Research (JMLR)* 19.1 (2018), pp. 2915–2914.

[136] Bruno Veloso, João Gama, and Benedita Malheiro. "Self hyper-parameter tuning for data streams". In: *Data Streams*. 2018.

Titre : Amélioration de l'Analyse des Flux de Données IdO à l'Aide de Techniques de Réduction de Données.

Mots clés : Internet des Objets ; Flux de Données ; Apprentissage Supervisé ; Classification ; Réduction de Dimension ; Résumés Minimalistes.

Résumé : Face à cette évolution technologique vertigineuse, l'utilisation des dispositifs de l'Internet des Objets (IdO), les capteurs, et les réseaux sociaux, d'énormes flux de données IdO sont générées quotidiennement de différentes applications pourront être transformées en connaissances à travers l'apprentissage automatique. En pratique, de multiples problèmes se posent afin d'extraire des connaissances utiles de ces flux qui doivent être gérés et traités efficacement. Dans ce contexte, cette thèse vise à améliorer les performances (en termes de mémoire et de temps) des algorithmes de l'apprentissage supervisé, principalement la classification à partir de flux de données en évolution. En plus de leur nature infinie, la dimensionnalité élevée et croissante de ces flux données dans certains domaines rendent la tâche de classification plus difficile.
La première partie de la thèse étudie l'état de l'art des techniques de classification et de réduction de dimension pour les flux de données, tout en présentant les travaux les plus récents dans ce cadre.
La deuxième partie de la thèse détaille nos contributions en classification pour les flux de données. Il s'agit de nouvelles approches basées sur les techniques de réduction de données visant à réduire les ressources de calcul des classificateurs actuels, presque sans perte en précision. Pour traiter les flux de données de haute dimension efficacement, nous incorporons une étape de prétraitement qui consiste à réduire la dimension de chaque donnée (dès son arrivée) de manière incrémentale avant de passer à l'apprentissage. Dans ce contexte, nous présentons plusieurs approches basées sur : Bayesien naïf amélioré par les résumés minimalistes et hashing trick, k-NN qui utilise compressed sensing et UMAP, et l'utilisation d'ensembles d'apprentissage également.

Title : Improving IoT Data Stream Analytics Using Summarization Techniques.

Keywords : Internet of Things ; Data Stream ; Supervised Learning ; Classification ; Dimensionality Reduction ; Sketching.

Abstract : With the evolution of technology, the use of smart Internet-of-Things (IoT) devices, sensors, and social networks result in an overwhelming volume of IoT data streams, generated daily from several applications, that can be transformed into valuable information through machine learning tasks. In practice, multiple critical issues arise in order to extract useful knowledge from these evolving data streams, mainly that the stream needs to be efficiently handled and processed. In this context, this thesis aims to improve the performance (in terms of memory and time) of existing data mining algorithms on streams. We focus on the classification task in the streaming framework. The task is challenging on streams, principally due to the high – and increasing – data dimensionality, in addition to the potentially infinite amount of data. The two aspects make the classification task harder.
The first part of the thesis surveys the current state-of-the-art of the classification and dimensionality reduction techniques as applied to the stream setting, by providing an updated view of the most recent works in this vibrant area.
In the second part, we detail our contributions to the field of classification in streams, by developing novel approaches based on summarization techniques aiming to reduce the computational resource of existing classifiers with no – or minor – loss of classification accuracy. To address high-dimensional data streams and make classifiers efficient, we incorporate an internal preprocessing step that consists in reducing the dimensionality of input data incrementally before feeding them to the learning stage. We present several approaches applied to several classifications tasks : Naive Bayes which is enhanced with sketches and hashing trick, k-NN by using compressed sensing and UMAP, and also integrate them in ensemble methods.

Institut Polytechnique de Paris
91120 Palaiseau, France