université PARIS-SACLAY

TELECOM ParisTech

NNT : 2019SACLT014

Thèse de doctorat

# Fast and Slow Machine Learning

Thèse de doctorat de l'Université Paris-Saclay
préparée à Télécom ParisTech

Ecole doctorale n°580 Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Paris, le 7 mars 2019, par

**JACOB MONTIEL LÓPEZ**

Composition du Jury :

| | |
|---|---|
| Ricard Gavaldà<br>Professor, Universitat Politècnica de Catalunya | Président |
| João Gama<br>Associate Professor, University of Porto | Rapporteur |
| Georges Hébrail<br>Senior Researcher, Électricité de France | Rapporteur |
| Themis Palpanas<br>Professor, Université Paris Descartes (LIPADE) | Examinateur |
| Jesse Read<br>Assistant Professor, École Polytechnique (LIX) | Examinateur |
| Albert Bifet<br>Professor, Télécom ParisTech (LTCI) | Directeur de recherche |
| Talel Abdessalem<br>Professor, Télécom ParisTech (LTCI) | Directeur de thèse |

# FAST AND SLOW MACHINE LEARNING

JACOB MONTIEL LÓPEZ



Thesis submitted for the Degree of
Doctor of Philosophy
at Université Paris-Saclay – Télécom Paristech

March 2019

Jacob Montiel López: *Fast and Slow Machine Learning*, © March 2019

SUPERVISORS:
Albert Bifet
Talel Abdessalem

LOCATION:
Paris, France

To *Dennis*, for your continuous teaching that life is here and now,
without you none of this would have been possible.

# ABSTRACT

The Big Data era has revolutionized the way in which data is created and processed. In this context, multiple challenges arise given the massive amount of data that needs to be efficiently handled and processed in order to extract knowledge. This thesis explores the symbiosis of batch and stream learning, which are traditionally considered in the literature as antagonists. We focus on the problem of classification from evolving data streams.

Batch learning is a well-established approach in machine learning based on a finite sequence: first data is collected, then predictive models are created, then the model is applied. On the other hand, stream learning considers data as infinite, rendering the learning problem as a continuous (never-ending) task. Furthermore, data streams can evolve over time, meaning that the relationship between features and the corresponding response (class in classification) can change.

We propose a systematic framework to predict over-indebtedness, a real-world problem with significant implications in modern society. The two versions of the early warning mechanism (batch and stream) outperform the baseline performance of the solution implemented by the Groupe BPCE, the second largest banking institution in France. Additionally, we introduce a scalable model-based imputation method for missing data in classification. This method casts the imputation problem as a set of classification/regression tasks which are solved incrementally.

We present a unified framework that serves as a common learning platform where batch and stream methods can positively interact. We show that batch methods can be efficiently trained on the stream setting under specific conditions. The proposed hybrid solution works under the positive interactions between batch and stream methods. We also propose an adaptation of the Extreme Gradient Boosting (XGBoost) algorithm for evolving data streams. The proposed adaptive method generates and updates the ensemble incrementally using mini-batches of data. Finally, we introduce scikit-multiflow, an open source framework in Python that fills the gap in Python for a development/research platform for learning from evolving data streams.

*What we want is a machine*
*that can learn from experience.*

**— Alan M. Turing**

# ACKNOWLEDGMENTS

I believe that humanity moves forward by the sum of actions from a multitude of individuals. Research, as a human activity, is no different. This thesis is possible thanks to the people and institutions that helped me during my PhD. Each and every one deserve my acknowledgment and have my gratitude.

I would like to thank my advisers, Prof. Albert Bifet and Prof. Talel Abdessalem for their continuous support, advice and mentoring. Thank you for your trust and encouragement to do my best and keep moving forward. I am certain that their human quality is one of the main factors in my positive experience working towards the completion of my PhD.

To my wife Dennis, who always believed in me and walked next to me through this adventure, always willing to hear about my research even when it was puzzling to her. To my parents, María Elena and José Alejandro, who layed the ground upon which I have been able to pursue all my dreams. To my family and friends whose continuous encouragement accompanies me on each step.

To the members of the *Data, Intelligence and Graphs* team: Jean-Louis, Mauro, Fabian, Pierre, Antoine, Camille, Mostafa, Heitor, Marie, Maroua, Dihia, Miyoung, Jonathan, Quentin, Pierre-Alexandre, Thomas [D], Thomas [F], Julien, Atef, Marc, Etienne, Nedeljko, Oana, Luis, Maximilien, Mikaël, Jean-Benoit and Katerina. As well as to the honorary members of the team: Nicoleta and Ziad. Our lunch-break discussions were a very welcomed break from the routine and are cherished memories from Télécom ParisTech. Special thanks to Prof. Jesse Read who taught me a lot about the practical side of machine learning, and to Prof. Rodrigo Fernandes de Mello whose passion for the theoretical elements of machine learning is an inspiration.

To the *Machine Learning* group at the University of Waikato for all their attentions during my visit. Special thanks to Prof. Bernhard Pfahringer, Prof. Eibe Frank and Rory Mitchell for the very interesting and enlightening discussions that led to our research collaboration.

Last but not least, I would like to thank the Government of Mexico for funding my doctoral studies via the National Council for Science and Technology (*Consejo Nacional de Ciencia y Tecnología*, CONACYT, scholarship 579465).

# CONTENTS


I INTRODUCTION

1 INTRODUCTION 3
- 1.1 Motivation . . . 3
- 1.2 Challenges and Opportunities . . . 5
- 1.3 Open Data Science . . . 6
- 1.4 Contributions . . . 7
- 1.5 Publications . . . 9
- 1.6 Outline . . . 10

2 PRELIMINARIES AND RELATED WORK 11
- 2.1 Streaming Supervised Learning . . . 11
  - 2.1.1 Performance Evaluation . . . 13
  - 2.1.2 Concept Drift . . . 13
  - 2.1.3 Ensemble Learning . . . 15
  - 2.1.4 Incremental Learning . . . 17
- 2.2 Over-Indebtedness Prediction . . . 18
- 2.3 Missing Data Imputation . . . 19

II CORE CONTENT

3 OVER-INDEBTEDNESS PREDICTION 25
- 3.1 A Multifaceted Problem . . . 26
  - 3.1.1 Feature Selection . . . 27
  - 3.1.2 Data Balancing . . . 28
  - 3.1.3 Supervised Learning . . . 28
  - 3.1.4 Stream Learning . . . 30
- 3.2 A Data-driven Warnining Mechanism . . . 30
  - 3.2.1 Generalization . . . 33
- 3.3 Experimental Evaluation . . . 33
- 3.4 Results . . . 36
  - 3.4.1 Feature Selection . . . 36
  - 3.4.2 Data Balancing . . . 39
  - 3.4.3 Batch vs Stream Learning . . . 40

4 MISSING DATA IMPUTATION AT SCALE 43
- 4.1 A Model-based Imputation Method . . . 43
  - 4.1.1 Cascade Imputation . . . 44
  - 4.1.2 Multi-label Cascade Imputation . . . 46
- 4.2 Experimental Evaluation . . . 49
  - 4.2.1 Impact on Classification . . . 50
  - 4.2.2 Scalability . . . 52
  - 4.2.3 Imputed vs Incomplete Data . . . 52
  - 4.2.4 Multi-label Imputation . . . 52
  - 4.2.5 Measuring Performance . . . 53

4.3 Results . . . . . 55
5 LEARNING FAST AND SLOW 61
5.1 From Psychology to Machine Learning . . . . . 62
5.2 Fast and Slow Learning Framework . . . . . 63
5.2.1 Fast and Slow Classifier . . . . . 63
5.3 Experimental Evaluation . . . . . 68
5.4 Results . . . . . 70
6 ADAPTIVE XGBOOST 79
6.1 Adapting XGBoost to Stream Learning . . . . . 80
6.1.1 Preliminaries . . . . . 80
6.1.2 Adaptive eXtreme Gradient Boosting . . . . . 81
6.1.3 Ensemble Update . . . . . 82
6.1.4 Handling Concept Drift . . . . . 83
6.2 Experimental Evaluation . . . . . 85
6.3 Results . . . . . 87
6.3.1 Predictive Performance . . . . . 87
6.3.2 Hyper-parameter Relevance . . . . . 91
6.3.3 Memory and Model Complexity . . . . . 94
6.3.4 Training Time . . . . . 96
7 SCIKIT-MULTIFLOW 99
7.1 Architecture . . . . . 101
7.2 In a Nutshell . . . . . 102
7.2.1 Data Streams . . . . . 102
7.2.2 Stream Learning Experiments . . . . . 103
7.2.3 Concept Drift Detection . . . . . 105
7.3 Development . . . . . 106

III CONCLUSIONS

8 FUTURE WORK AND CONCLUSIONS 109
8.1 Future Directions . . . . . 109
8.1.1 Over-indebtedness Prediction . . . . . 109
8.1.2 Missing Data Imputation . . . . . 109
8.1.3 Fast and Slow Learning . . . . . 110
8.1.4 Adaptive XGBoost . . . . . 110
8.2 Final Remarks . . . . . 110

IV APPENDICES

A CASCADE IMPUTATION 115
B RÉSUMÉ EN FRANÇAIS 123
B.1 Défis et opportunités . . . . . 123
B.2 Open Data Science . . . . . 124
B.3 Contributions . . . . . 125

BIBLIOGRAPHY 129

# Part I

# INTRODUCTION

# 1 INTRODUCTION

In recent years we have witnessed a digital revolution that has dramatically changed the way in which we generate and consume data. In 2016, 90% of the world's data was created just on the last 2 years and is expected that by 2020 the digital universe will reach 44 zettabytes (44 trillion gigabytes[1]). This new paradigm of ubiquitous data has impacted different sectors of society including government, healthcare, banking and entertainment, to name a few. Due to the extent of its potential data has been dubbed "the oil of the digital era"[2].

The term *Big Data* is used to refer to this groundbreaking phenomenon. A continuous effort exists to delimit this term; a popular approach are the so called "*4 Vs*" of big data: *Volume, Velocity, Variety* and *Veracity*. Volume is related to the scale of the data which varies depending on the application. Velocity considers the rate at which data is generated/collected. Variety represents the different types of data, including traditional structured data and emerging unstructured data. Finally, Veracity corresponds to the reliability that can be attributed to the data.

Another significant factor is the dramatic growth of the *Internet of Things* (IoT), which is the ecosystem where devices (things) connect, interact and exchange data. Such devices can be analog or digital, e.g. cars, airplanes, cellphones, etc. By 2013, 20 billion of such devices were connected to the internet and this number is expected to grow to 32 billion by 2020, this means an increment from 7% to 15% of the total number of connected devices. The contribution of IoT to the digital universe is considerable. For example, data only from embedded systems accounted for 2% of the world's data in 2013, and is expected to hit 10% by 2020[3], Figure 1.1.

## 1.1 MOTIVATION

In its raw state, data contains latent information which can potentially be converted into knowledge (and value). However, as the amount of data and its complexity increases, its analysis becomes unfeasible for humans. To overcome this, *Artificial Intelligence* is the field of study that

1 *The digital universe in 2020*, John Gantz and David Reinsel, IDC, February 2013.
2 *The world's most valuable resource is no longer oil, but data*, The Economist, May 2017.
3 *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things*, IDC, April 2014.

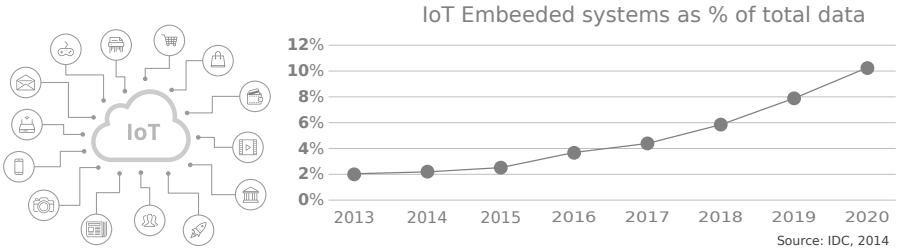



**Figure 1.1:** IoT explosion in the digital era.

aims to produce machines which can replicate human's intelligence without its biological limitations.

The ultimate goal is to convert data into knowledge which in turn translates into actions. In other words, we are interested in finding patterns in the data. *Machine Learning*, a sub-field of artificial intelligence, is a kind of data analysis that automates the process of finding and describing patterns in data by building analytical models. Machine learning is based on the assumption that machines can *learn* from data, identify patterns and make predictions with minimal human intervention. The minimal pipeline of machine learning, Figure 1.2, involves using the data that represents a phenomenon to build a mathematical model which represents its nature.

Artificial intelligence (including machine learning) is already a key element for competitiveness and growth in different industries, what's more, it has disrupted not only the private sector but the public sector as well. Several countries include artificial intelligence initiatives as a key axis for national development and competitiveness. For example, artificial intelligence is one of the pillars of *Horizon 2020*, the EU's Research and Innovation program that implements the *Europe 2020* initiative for Europe's global competitiveness. Similarly, in March 2018 France unveiled its national artificial intelligence strategy which not only aims at strengthening the role of France in this field, but also proposes an ethical framework to regulate it[4].




**Figure 1.2:** The minimal pipeline of machine learning. Data in a digital format is used during training to build a mathematical model that represents a phenomenon.

4 *Villani Report*, March 2018.

## 1.2 CHALLENGES AND OPPORTUNITIES

Machine learning is not new, in reality is a well established field of study with a strong scientific and technical background. However, the disruptive impact of big data and the challenges it poses has reinvigorated the research community. Furthermore, it has contributed to turn machine learning into a matter of great interest to the general public. Currently, one of the most active research directions in the ubiquitous data era is to facilitate the mechanisms to perform machine learning at scale. In this context, it is vital that learning methods are able to keep the pace with data, not only in terms of volume but also the speed at which it is generated and processed, in order to be useful to humans.

For example, online financial operations in the digital era are becoming the norm in multiple countries. In this context, automatic fraud detection mechanisms are required to protect millions of users. Training is performed on massive amounts of data, thus consideration of runtime is critical: waiting until a model is trained means that potential frauds may pass undetected. Another example is the analysis of communication logs for security, where storing all logs is impractical (and in most cases unnecessary). The requirement to store all data is an important limitation of methods that rely on doing multiple passes over the data.

Traditionally, the minimal pipeline depicted in Figure 1.2 is applied to batches or chunks of data. As new data becomes available, the same sequence is repeated and a new model is generated. This approach, known as *Batch Learning*, has been proven effective in many real-world applications. However, it represents important compromises when tackling big data problems. For example, if data is coming at high speed rates, we need to run the full pipeline over and over again to keep a sequence of models consistent with recent data. Since (in general) no knowledge is retained, we need to run the entire pipeline

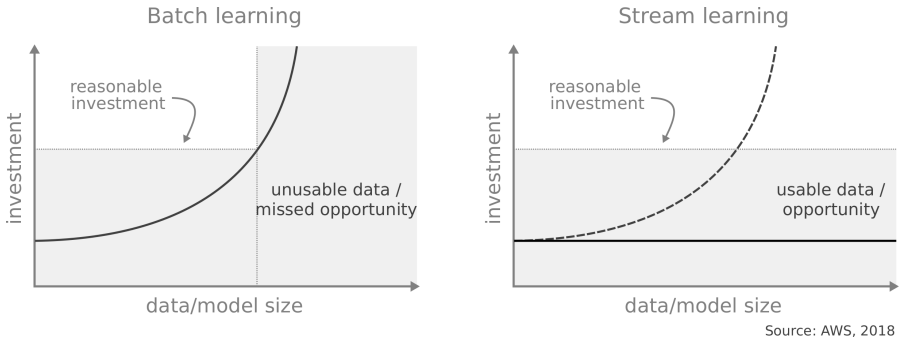



**Figure 1.3:** Investment vs data size. Batch learning often struggles to maintain investment (time, memory, cost) below a reasonable level. Stream learning addresses this situation by efficiently generating incremental models from infinite data streams.

again, which can represent a considerable waste of resources if the data only presents small variations over time. Also challenging is to define the amount of data to use or the time we are willing to wait while maximizing the chances of generating an optimal model.

The balance between the investment in resources (time, memory, costs) and the quality of the model is a critical element for the viability of a learning algorithm in real-world big data applications. *Stream Learning* is the emerging research field which focuses on learning from infinite data streams. A stream learning algorithm only sees data once, modifies its internal state (model) and then moves on to process the next data sample with the premise that past samples are not seen again. Stream learning emphasizes on an efficient use of resources without compromising learning, Figure 1.3. In this thesis, we focus on the overlapping region between batch and stream learning.

## 1.3 OPEN DATA SCIENCE

Data Science is an emerging interdisciplinary field in the digital era which unifies statistics, data analysis and machine learning to extract knowledge and insights from data. One of the major contributors to the fast adoption of data science is the *Open Source* movement. The designation "open source" is attributed to something that people, other than the author, can modify and share because its design is publicly available. In the context of software development it refers to the process used to develop computer programs. The machine learning community has benefitted from an ample number of open source frameworks focused on multiple topics and platforms (operative system and programming language). As examples of the advantages of open source research we can pinpoint:

- **Reproducible research**, an essential part of the scientific process.
- **Faster development**, since researchers can focus on the core elements of their work without getting sidetracked by technical details.
- **Fosters collaboration**, by providing a common platform on which a community can thrive.
- **Democratization** of machine learning by reducing the technical gap for non-expert individuals.
- **Maintainability** based on the community and not in isolated individuals or groups.

## 1.4 CONTRIBUTIONS

In this section we summarize the contributions of this thesis. In Table 1.1 we categorize each chapter by type of learning (batch and/or stream) and its focus (research and/or application).

In the context of **over-indebtedness prediction** (Chapter 3):

- Unlike other approaches we embrace over-indebtedness prediction as a multifaceted classification problem and provide a comprehensive data-driven solution.
- We focus on the feature-selection process rather than on specific hand-picked features. In the literature, several solutions rely on fixed sets of features assuming that: i) Data from different banking institutions have similar distributions. ii) Features are replicable. Assumptions that are difficult to fulfill in real-world applications.
- We study the impact of extreme class imbalance. Although the primary objective is to identify people at risk, the impact of misclassification on both classes is considered. i) We apply a multi-metric criterion to find the best compromise between a model's performance and fairness. ii) We discuss the associated cost of misclassifying instances.
- As far as we know, this study is the first to cast over-indebtedness prediction as a stream learning problem. This is an attractive solution since stream models adapt to changes in the data, a drawback of traditional batch learning where new models have to be generated over time to replace obsolete models.

Regarding **missing data imputation** (Chapter 4):

- We propose a new scalable and effective model-based imputation method that casts the imputation process as a set of classification/regression tasks.
- Different to well established imputation techniques, the proposed method is non-restrictive on the type of missing data, supporting: *Missing At Random* and *Missing Completely At Random* mechanisms, numerical and nominal data, small to large data sets, including high dimensional data.
- We provide a solution to impute multiple features at a time via multi-label learning. To the extent of our knowledge, our approach is the first to address multi-label learning for hybrid data types, meaning that it imputes numerical and nominal data concurrently.

The contributions of the **fast and slow learning** framework (Chapter 5) are:

- We propose a unified strategy for machine learning based on *fast* (stream) and *slow* (batch) learners. By considering learning as a continuous task, we show that batch learning methods can be effectively adapted to the stream setting under specific conditions.
- We show the applicability of this new paradigm in classification. Test results on synthetic and real-world data confirm that the Fast and Slow Classifier leverages stream and batch methods by combining their strengths: a fast model adapts to changes in the data and provides predictions based on current concepts while a slow model generates complex models built on wider data distributions.
- We believe that our research sheds a new light on the possibility of hybrid solutions where batch and stream learning positively interact. This has special relevance in real-world applications bounded to batch solutions and whose transition to the stream setting represents a big compromise.

For our work on **boosting for stream learning** (Chapter 6), we list the following contributions:

- We propose an adaptation of the eXtreme Gradient Boosting (XGBoost) algorithm for evolving data streams. The core idea is the incremental creation/update of the ensemble, i.e., weak learners are trained on mini-batches of data and then added to the ensemble.
- Additionally, we consider a simple batch-incremental approach where members of the ensemble are full XGBoost models trained on consecutive mini-batches. If resource consumption is a secondary consideration, this approach (after parameter tuning) may be a worthwhile candidate for application in practical data stream mining scenarios.
- We perform a thorough evaluation in terms of performance, hyper-parameters relevance, memory and training time. Although the main objective is learning from data streams, we believe that our stream method is an interesting alternative to the batch version for some applications given its efficient management of resources and adaptability. In addition, our evaluation serves as an update on studies found in the literature that compare instance-incremental and batch-incremental methods.

Finally, the contributions of our open source **stream learning framework** (Chapter 7):

- We introduce scikit-multiflow, an open source framework for learning from data streams and multi-output learning in Python. The source code is publicly available and distributed under the BSD 3-Clause.
- scikit-multiflow provides multiple state-of-the-art learning methods, data generators and evaluators for different stream learning problems, including single-output, multi-output and multi-label.
- scikit-multiflow is conceived to serve as a platform to encourage the democratization of stream learning research. For example, proposed methods in Chapter 5 and Chapter 6 are implemented on scikit-multiflow and the corresponding evaluation is (in most cases) performed on this platform.

**Table 1.1:** Field and focus of the content in this work.

| | Stream Learning | Batch Learning | Research | Application |
|---|---|---|---|---|
| Chapter 3 | ✓ | ✓ | | ✓ |
| Chapter 4 | | ✓ | ✓ | |
| Chapter 5 | ✓ | ✓ | ✓ | |
| Chapter 6 | ✓ | ✓ | ✓ | |
| Chapter 7 | ✓ | ✓ | ✓ | ✓ |

## 1.5 PUBLICATIONS

The research work presented in this thesis has been published on scientific venues in the corresponding field. In the following we provide a list of selected publications:

- Jacob Montiel, Albert Bifet, and Talel Abdessalem. "Predicting over-indebtedness on batch and streaming data." In: *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*. 2017, pp. 1504–1513
- Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. "Scalable Model-Based Cascaded Imputation of Missing Data." In: *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part III*. 2018, pp. 64–76
- Jacob Montiel, Albert Bifet, Viktor Losing, Jesse Read, and Talel Abdessalem. "Learning Fast and Slow: A Unified Batch/Stream Framework." In: *2018 IEEE International Conference on Big Data,*

*BigData 2018, Seattle, WA, USA, December 10-13, 2018*. IEEE. 2018, pp. 1065–1072

- Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. "Scikit-Multiflow: A Multi-output Streaming Framework." In: *Journal of Machine Learning Research* 19.72 (Oct. 2018), pp. 1–5

The following papers have been submitted to journals in the field of machine learning and are under review:

- Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. "A Hybrid Framework for Scalable Model-based Imputation." In: (2018). Submited
- Jacob Montiel, Rory Mitchell, Eibe Frank, Bernhard Pfahringer, Talel Abdessalem, and Albert Bifet. "Adaptive XGBoost for Evolving Data Streams." In: (2018). Submited

## 1.6 OUTLINE

We provide a general description of the content of this thesis, the relevant research fields and its application. Preliminaries and related work are discussed in Chapter 2. In Chapter 3 we introduce a data-driven early warning mechanism to predict over-indebtedness, an application of supervised learning with important implications for society. We explore batch and stream learning as independent approaches for the same problem. In Chapter 4 we focus on missing data, a common feature of data that can compromise learning if it is not properly handled. We present a scalable model-based imputation method for batch learning classification. Next, we focus on a research question that raises from the work in Chapter 3: *Can batch and stream learning solutions work together?* In contrast to the traditional conception that batch and stream learning are mutually exclusive, we propose an unified approach by integrating batch and stream techniques under a continuous learning framework. Following the same research direction, Chapter 6 introduces an adaption of eXtreme Gradient Boosting for mining evolving data streams. Then, in Chapter 7 we describe scikit-multiflow, an open source stream learning framework. scikit-multiflow fills the need of an open source stream learning platform in Python and is used to implement and evaluate the methods in Chapter 5 and Chapter 6. Finally, we present our conclusions and future directions in Chapter 8.

# 2 PRELIMINARIES AND RELATED WORK

In this Chapter, we discuss the technical background for the content of this thesis and present related work. This chapter covers three major topics: *streaming supervised learning*, *over-indebtedness prediction* and *missing data imputation*.

## 2.1 STREAMING SUPERVISED LEARNING

In this thesis, we focus on *classification*, a type of supervised learning where a model $h$ is generated (trained) from labeled data $(X, y)$. Each data sample $i$ in $(X, y)$ is composed of a features vector $\vec{x}_i$ and the corresponding response $y_i$, which in the case of classification is the class label to learn. The objective is to apply the model $h$ to unknown (unlabeled) data $X'$, where $\{X \cup X'\} = \emptyset$, to predict the corresponding class label

$$y' = h(X') \tag{2.1}$$

Two classes are considered in *binary* classification, $y \in \{0, 1\}$, while $K > 2$ classes are used in *multi-class* classification, $y \in \{1, \ldots, K\}$. For both *binary* and *multi-class* classification only one class is assigned per instance. In traditional single-output models, we deal with a single target variable $y_i$ for which one corresponding output is produced per test instance. Multi-output models can produce multiple outputs to assign to multiple target variables $\vec{y}_i$ for each test instance.

With the development of the internet of things, more data is continuously collected at faster rates. Most state-of-the-art data mining techniques, designed to work in *batches* of data, do not provide native support for continuous flows of data. From this, a new approach emerges: *stream learning*. The *stream data* model assumes a non-stopping, theoretically infinite, flow of data. In this context it is vital that learning methods are efficient in terms of resources, in particular **time** and **memory**. This leads to a set of special requirements, a type of trademark, for stream data mining methods [17]:

1. **Process one example at a time, and inspect it only once.** Different to batch learning, the assumption is that there is not enough time nor space to store multiple samples, failing to meet this requirement implies the risk of missing incoming data.

2. **Use a limited amount of memory.** Data streams are assumed to be infinite, storing data for further processing is impractical.

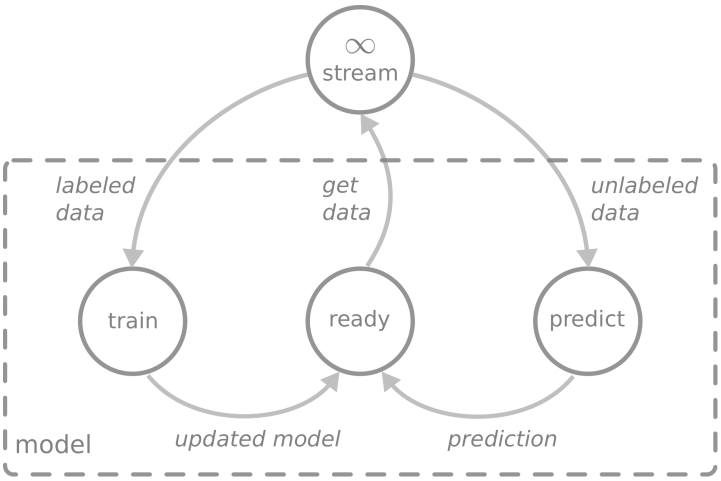



**Figure 2.1:** Stream supervised learning. A data stream is an infinite source of data whereas a predictive model transitions from *ready* to either *train* or *predict* and back. Labeled data is used to train/update the model. If data is unlabeled, then the model predicts the corresponding target value (class label in the case of classification).

3. **Work in a limited amount of time.** In other words, avoid bottlenecks generated by time consuming tasks which in the long run could make the algorithm fail.

4. **Be ready to predict at any point.** Different to batch models, stream models are continuously *updated* and shall be ready to predict at any point in time.

Learning from data streams is an important tool in the context of applications that require specialized methods to process and understand large volumes of data that are continuously generated. We can define learning from evolving data streams as follows. Consider a continuous stream of data $A = \{(\vec{x}_t, y_t)\}|t = 1, \ldots, T$ where $T \to \infty$. Input $\vec{x}_t$ is a feature vector and $y_t$ the corresponding target where $y$ is continuous in the case of regression and discrete for classification. The objective is to predict the target $y'$ for an unknown $\vec{x'}$. The stream learning process is outlined in Figure 2.1.

Different to batch learning, where all data $(X, y)$ is available during the training phase; in stream learning, training is performed incrementally as new data $(\vec{x}_t, y_t)$ becomes available. In stream learning, the training phase never ends and predictive models are continuously being updated. This is a key difference with respect to batch learning where models are generated after analyzing all the available data and where a *new* model is generated each time as more data is available.

Also important is to note that stream learning is resource-wise efficient, Figure 2.2. In terms of memory, the model size is desired to remain stable no mather the amount of data. This means that the model size fits in memory no mather the volume of seen data. On the

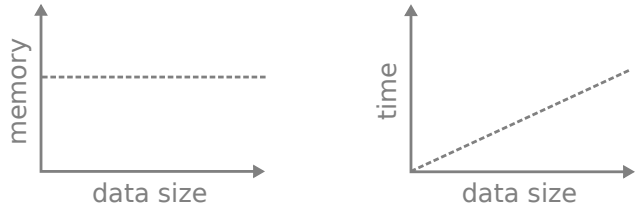



Figure 2.2: Stream learning represents an efficient management of resources (memory and time) without compromising model performance.

other hand, training time is expected to grow linearly with respect to the volume of data processed from the stream.

### 2.1.1 Performance Evaluation

The evaluation of stream learning methods requires special considerations due to its incremental nature. Performance $P$ of a given model is measured according to some loss function $\ell$ that evaluates the difference between expected (true) class labels $y$ and the predicted class labels $\hat{y}$.

$$P(h) = \ell(y, \hat{y}) \tag{2.2}$$

For example, a popular and straightforward loss function for classification is the *zero-one losss function* which corresponds to the notion of weather the model made a mistake or not when predicting.

$$\ell(y, \hat{y}) = \begin{cases} 0, & y = \hat{y} \\ 1, & y \neq \hat{y} \end{cases} \tag{2.3}$$

Two prevalent methods in the stream learning literature are *hold-out* and *prequential* evaluation. The hold-out evaluation is a popular evaluation method for both batch and stream learning. In hold-out evaluation, testing is performed on a separate set. On the other hand, prequential evaluation [32] or interleaved-test-then-train evaluation, is a popular performance evaluation method for the stream setting only. In prequential evaluation, tests are performed on new data samples *before* they are used to train (update) the model.

### 2.1.2 Concept Drift

Considering the theoretical infinite nature of data streams, an interesting phenomenon arises: the underlying relationship between features and their corresponding target(s) can evolve (change) over time. This phenomenon is known as *Concept Drift*. Real concept drift is defined as changes in the posterior distribution of the data $p(y|X)$. Real concept drift means that the unlabeled data distribution does not change,




**Figure 2.3:** Drift patterns[1], depicted as the change of mean data values over time. Note that an outlier is not a change but *noise* in the data.

whereas data evolution refers to the unconditional data distribution $p(X)$. This is a major difference between stream and batch learning given that in the latter the joint distribution of data $p(X, y)$ is, in general, assumed to remain stationary. In the context of evolving data streams, concept drift is defined between two points in time $t_o, t_1$ as

$$p_{t_0}(X, y) \neq p_{t_1}(X, y) \tag{2.4}$$

Depending on the pattern of the change, Figure 2.3, we can estimate the impact of concept drift on learning. We can categorize concept drift into the following patterns [50]:

- **Abrupt.** When a new concept is immediately introduced. The transition between concepts is minimal. In this case, adaptation time is vital since the old concept becomes is no longer valid.
- **Incremental.** Can be interpreted as the transition from an old concept into a new concept where intermediate concepts appear during the transition.
- **Gradual.** When old and new concepts concur within the transition period. Can be challenging since both concepts are somewhat valid during the transition.
- **Recurring.** If an old concept is seen again as the stream progresses. For example, when the data corresponds to a seasonal phenomenon such as the circadian rhythm.

Concept drift is of special interest when working with evolving data streams due to its direct impact on learning. Although the incremental nature of stream methods provides some robustness to concept drift, specialized methods have been proposed to explicitly detect drift. Multiple drift detection methods have been proposed in the literature, [50] provides a thorough survey of this topic. In general, drift detection methods work under a common framework: the objective is to accurately detect changes in the data distribution while being robust to noise and resource-wise efficient, in agreement with stream learning requirements. Some concept-drift-aware methods use drift detectors to react faster and efficiently to changes. For example, the Hoeffding Tree algorithm [64] (a kind of decision tree for streams) does not address concept drift directly. To solve this, the Hoeffding Adaptive Tree [14]

1 Figure is based on [50]

uses *ADaptive WINdowing* (ADWIN) [13] to detect drifts. If a drift is detected, the structure of the tree is updated by replacing old branches with new branches trained on the current concept.

ADWIN is a popular drift detection method with mathematical guarantees. ADWIN efficiently keeps a variable-length window of recent items; such that it holds that there has no been change in the data distribution. This window is further divided into two sub-windows $(W_0, W_1)$ used to determine if a change has happened. ADWIN compares the average between $W_0$ and $W_1$ to confirm that they correspond to the same distribution. Concept drift is detected if the distribution equality no longer holds. Upon detecting a drift, $W_0$ is replaced by $W_1$ and a new $W_1$ is initialized. ADWIN uses a confidence value $\delta =\in (0, 1)$ to determine if the two sub-windows correspond to the same distribution.

### 2.1.3 Ensemble Learning

Ensemble methods are a popular approach to improve performance and robustness[25, 54, 96]. Different to single-models, ensemble methods yield predictions by combining the predictions of multiple models, Figure 2.4. The reasoning for this is that finding a single top model is difficult, whereas multiple "weak" models are relatively easy to train and combined performance can be at par with state-of-the-art methods [44]. The superiority of ensemble classifiers over single classifiers is justified in[34] on three reasons: (i) *Statistical*, having multiple models can reduce the risk of selecting a single sub-optimal function. (ii) *Computational*, potentially avoid local optima in some methods by optimizing different functions. And (iii) *Representational*, new functions can be obtained from the combination of the base functions in the ensemble.

One of the first techniques to address concept drift with ensembles trained on streaming data is the *SEA* algorithm [113], a variant of bagging which maintains a fixed-size ensemble trained incrementally on chunks of data. *Online Bagging* [93] is an adaptation of bagging for data streams. Similar to batch bagging, $M$ models are generated and then trained on $N$ samples. Different to batch bagging where samples are selected with replacement, in Online Bagging samples are assigned a weight based on $Poisson(1)$. *Leveraging Bagging* [18], builds upon Online Bagging. The key idea is to increase accuracy and diversity on the ensemble via randomization. Additionally, Leveraging Bagging uses ADWIN to detect drifts, and if a change is detected, the worst classifier is dropped and a new one is added to the ensemble. The *Ensemble of Restricted Hoeffding Trees* [12] is an ensemble method that combines the predictions of multiple models using stacking. The base level models are Hoeffding Trees trained on different attribute-subsets and the meta level learners are perceptrons (one per class value)

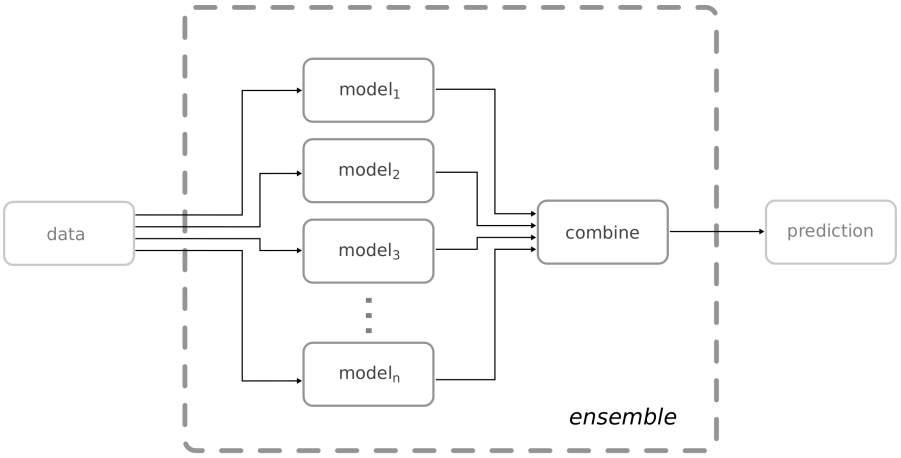



**Figure 2.4:** Ensemble model. Data (labeled or unlabeld) is passed to the $n$ members of the ensemble. If data is labeled then it is used to train each member of the ensemble. Diversity is a key element of ensembles, thus it is assumed that the trained models will be slightly different in order to profit from their combined predictions. If data is unlabeled, then each member of the ensemble yields its corresponding prediction which are combined to generate the final prediction for the ensemble.

trained on class probabilities from the base level models. The *Diversity for Dealing with Drifts* ensemble method in [83] uses a diversity control mechanism based on online bagging and a drift detector to speed-up adaptation. This method keeps two ensembles, one with low diversity and one with high diversity. Both ensembles are trained with incoming data but only one is used to provide predictions, the low-diversity ensemble for stable regions (no concept drift) and the high-diversity ensemble in regions after a drift is detected and until entering in a stable concept region.The *Self Adjusting Memory* algorithm [79] builds an ensemble with models targeting current or former concepts. SAM works under the *Long-Term — Short-Term* memory model (LTM-STM), where the STM focuses on the current concept and the LTM retains information about past concepts. A cleaning process is in charge of controlling the size of the STM while keeping the information in the LTM consistent with the information in the STM. *Adaptive Random Forest* [54] is an extension of Random Forest designed to work on data streams. The base learners are Hoeffding Trees, attributes are randomly selected during training, and concept drift is tracked using ADWIN on each member of the ensemble. If a member of the ensemble exhibits a drop in performance (presumably due to concept drift), then it is replaced by an alternate tree trained on the new concept.

Boosting is another popular ensemble strategy in the batch setting. The *Pasting Votes* method [22] is the first to apply boosting on large data sets by using different sub-sets on each iteration, thus it does not require to store all data and potentially can be used on stream

data. Predictions are obtained by aggregating many weak learners (batch classifiers) trained on different small subsets generated either by random sampling or importance sampling (equal number of correctly [easy] and incorrectly [difficult] classified samples). A similar approach based on boosting is *Racing Committees* [42]. Different to [22], Racing Committees includes an adaptive pruning strategy to manage resources (time and memory). Pruning is only applied on the last model in the boosting sequence to decide if a new model should be added. Additionally, the batch size used for training the ensemble can be chosen among several candidates based on "racing", i.e. candidates run concurrently. In the stream setting, a number of approaches for boosting have been proposed. *Learn++* [97], inspired in *AdaBoost* [44], generates a batch-based ensemble of weak classifiers trained on different sample distributions and combines weak hypotheses through weighted majority voting. The distribution update rule in AdaBoost is aimed at improving the ensemble's accuracy by focusing on difficult (misclassified) samples, whereas Learn++ focuses on incremental data that could introduce new classes. Alongside Online Bagging, Oza and Russell introduced an online versions of boosting in [93]. Different to the bagging version, when a member of the ensemble misclassifies a training example, the associated Poisson-distribution is increased when passing it to the next member of the ensemble, otherwise it is decreased. A boosting-like ensemble method explicitly designed to handle concept drift is proposed in [108]. The ensemble is generated using knowledge-based sampling where patterns are discovered iteratively. The distributions update rule measures the correlation between predictions and true class label to ensure that new distributions do not include knowledge already encoded in the weak models. *Online Boost-By-Majority* [11] is a stream version of the boost-by-majority algorithm [43] for binary classification, based on a weaker assumption of online weak learners. The authors propose an optimal version in terms of ensemble size and sample complexity and one sub-optimal version that is adaptive and parameter-free. The *Online Multiclass Boost-By-Majority* algorithm [66] is an extension for multiclass classification. Similarly, one optimal version and one sub-optimal version are introduced. *BoostVHT* [117] improves training time by taking advantage of parallel and distributed architectures without modifying the order assumption in boosting. To achieve this, they use as base learner the Vertical Hoeffding Trees[71], a model-parallel distributed version of the Hoeffding Tree.

### 2.1.4 Incremental Learning

Two main branches can be distinguished in stream learning depending on the schema used to train a model: *Instance-incremental* methods [12, 14, 18, 54, 79, 93], use one sample at a time for training.

*Batch-incremental* methods [22, 42, 97] use batches of data. Once a given number of samples are stored in the batch, they are used to train the model. The *Accuracy-Weighted Ensembles* [119], is a framework for mining streams with concept drift using weighted classifiers. The members of the ensemble are discarded by an instance-based pruning strategy if they are below a confidence threshold. A relevant study is [104], where the authors compare batch-incremental and instance-incremental methods for the task of classification of evolving data streams. While instance-incremental methods perform better on average, batch-incremental methods achieve similar performance in some scenarios.

## 2.2 OVER-INDEBTEDNESS PREDICTION

Over-indebtedness is a serious social and economical problem which has been subject of multi-disciplinary research for a long time. Risk prediction solutions have been proposed for bank and firms as well as consumers (households). The financial crisis of 2008 reinforced the commitment of governments and financial institutions to implement measures to control and minimize the risk of over-indebtedness [29, 30, 41, 53, 102].

Traditionally, economical and demographic factors are considered primary factors of debt, as suggested by empirical models. Empirical approaches for over-indebtedness prediction have been studied since the late 60s. We refer the reader to [103] for a review of empirical approaches for bankruptcy prediction in banks and firms from 1968 to 2005. Papers are categorized primary by the technique employed but consideration is also given to data sources, features and techniques to measure performance. Early approaches based on statistical techniques have been replaced by newer intelligent techniques which provide better performance such as neural networks [5, 116], genetic algorithms [110], discriminant analysis [24], and support vector machines [63, 81, 109], to mention a few.

Recent inter-disciplinary empirically driven approaches provide an alternative explanation in terms of economic behavior [1, 51, 62, 68, 92, 112], given that conventional economics have been proven to be insufficient to explain the multi-causal nature of over-indebtedness. These studies provide evidence of the relevance of psychological and behavioral factors on over-indebtedness.

Despite the potential benefits of feature selection [23, 55], its application is still limited in over-indebtedness prediction. In [115] the author points out that only 6 out of 18 studies published from 2001 to 2007 consider Feature Selection as part of their approaches. Traditionally, researchers rely on the development of new models to attain better prediction performance. We argue that recent developments on feature

selection techniques [76] represent an opportunity for data mining applications, such as over-indebtedness prediction.

## 2.3 MISSING DATA IMPUTATION

Missing data *mechanisms* describe the underlying nature of this phenomenon and are classified into three major categories [77]: I. *Missing Completely At Random* (MCAR), the strongest assumption of missingness, where the events behind missing values are independent of observable variables and the missing values themselves, in other words, they are entirely at random. II. *Missing At Random* (MAR), intermediate level of missingness, where missingness can be explained by observable variables. III. *Missing Not At Random* (MNAR), when data is not MCAR nor MAR and the reason for missingness of data is related to the value of the missing data itself. Consider $X$ the matrix of input attributes and $y$ the corresponding labels, let $D = (X, y)$. For the observed values (not missing) $D_{obs}$, let $D_{obs} = (X_{obs}, y)$. A *missingness* matrix $M$ with the same shape as $X$ indicates if a value is missing on $X$ by setting its $ij$th entry to 1. It follows that:

$$\begin{aligned} \text{MCAR:} \quad & P(M|D) = P(M) \\ \text{MAR:} \quad & P(M|D) = P(M|D_{obs}) \end{aligned} \tag{2.5}$$

In the presence of missing data, three approaches are usually considered [45]: (i) Discard instances with missing values, (ii) Let the learning algorithm deal with missing values, (iii) Impute (fill) missing data. Option (i) is reasonable if the amount of missing values is very small, otherwise discarding large number of instances can make the learning task harder or even infeasible. Additionally, for highly dimensional data sets, discarding a complete instance due to the presence of at least one missing value represents a significant compromise. Option (ii) is considered under the premise that the learning algorithm supports missing values in the data. Even if this is the case, we need to understand how the algorithm handles missing values and the limitations of such approach. Imputation (iii) is usually the recommended approach.

Although most imputation techniques focus on MCAR data [45, 69], we argue that imputation methods shall tackle both MCAR and MAR scenarios. However, manual imputation of MCAR and MAR data is a time consuming process and requires deep understanding of the data and the phenomena that it describes. On the other hand, manual imputation is recommended for MNAR data, given that data specialists are more likely to identify the reasons behind this type of missing data and the proper way to handle it. Finally, current trends in data generation and collection have shifted the data archetype in the machine learning community to larger and more complex data.

Under this scenario, imputing data manually is impractical, therefore *scalable* automatic imputation solutions are required for real-world applications, one of such applications is classification.

Various approaches have been proposed in the literature. Basic methods rely on simple statistical values such as *mean* [77, 90] and *covariance* [77]. Other methods *reconstruct* incomplete data incrementally: *kNN* based methods [9, 69, 120] impute data based on the neighborhood of a missing value and show good results when the data is small. However, these methods suffer the computational burden of the kNN algorithm making scalability an issue. *Self-Organizing Map Imputation* (SOMI) [39] is a neural network model that first ignores missing data when finding patterns, and then imputes missing data based on the weights of activation nodes of the previously estimated patterns. *Expectation Maximization Imputation* (EMI) [33] performs imputation based on the expectation-maximization algorithm. EMI imputes values based on mean and covariance during the expectation step, then updates them in the maximization step, this process continues until convergence. Kernel methods [98, 99, 124] build imputation models based on kernel functions. These non-parametric methods usually consist of two stages: kernel function selection and bandwidth adjustment. *Fuzzy C-Means Imputation* (FCMI) [75] and *k-Means clustering* (KMC) [122], use clusters generated from non-missing data. While some imputations methods use the entire data set (EMI, Mean Imputation, Most Frequent Value), others use only parts of it (kNNI, FCMI, KMI).

In the following, we summarize relevant imputation methods and categorize them in Table 2.1:

**Table 2.1**: Related imputation methods categorization

| | | (a) | (b) | (c) | (d) | (e) | (f) |
|---|---|---|---|---|---|---|---|
| Data Type | Numerical | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| | Nominal | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Mechanism | MAR | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | MCAR | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Performance Evaluation | Classification | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| | Regression | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | Imputation Error[1] | ✗ | ✓ | ✓ | ✗ | ✓ | ✓ |
| Data set size[2] | | S | S-L | S-L | S-M | S | S |
| Missing values ratio[3] | | S-L | S | S | M-L | M-L | M-L |

[1] With respect to the complete data set.
[2] Values count: Small [<200K], Medium [200K-600K], Large [>600K]
[3] Small [<10%], Medium: [10%-25%], Large [>25%]

(**a**) *Locally Linear Reconstruction* (LLR) [69] determines the number of neighbors $k$ and the weights given to the neighbors in kNN learning. Imputation is limited to numerical values. Imputed data sets are used to train classification/regression models. LLR assumes $k$ to be 'sufficiently large' which represents a compromise for large data sets.

(**b**) A combination of EMI with Decision Trees (DMI) and Decision Forest (SiMI) are proposed in [100]. The goal is to identify segments of data where instances have higher similarity and attribute correlation. A tree-based algorithm identifies partitions in the data, given that leaves are sets of mutually exclusive instances. Imputation of numerical and nominal data is performed via EMI and Majority Class respectively.

(**c**) FEMI [101] uses the General Fuzzy C-Means (GFCM) [74] clustering algorithm to find most similar instances for imputation via EMI. FEMI supports imputation of numerical and nominal data. The required number of $k$ clusters is manually set.

(**d**) A model-based approach is provided in [114]. Imputation of nominal data is performed via a classification approach, based only on observed values. Imputed data is used to train a classification model which is applied to complete test instances.

(**e**) A non-parametric iterative imputation method for numerical and nominal values is proposed in [124]. This method uses a mixed-kernel-based estimator and applies a grid search strategy for selecting the optimal bandwidth. Imputed values are used to impute missing values in subsequent iterations.

(**f**) An iterative model-based imputation method is presented in [107]. Imputation of numerical data is carried by iteratively applying regression functions in two stages. In Iteration 1, a regression model is constructed for each attribute with missing data based only on observed data, then missing values are predicted using these models. In Iteration 2, predicted values are merged with observed data and new models are generated. Iteration 2 is repeated until the difference between predictive values falls bellow a user defined threshold.

# Part II

# CORE CONTENT

# 3 OVER-INDEBTEDNESS PREDICTION

Household over-indebtedness is an important challenge for financial institutions and an interdisciplinary research topic. It is defined by the European Union (EU) as "difficulties meeting (or falling behind with) commitments, whether these relate to servicing secured or unsecured borrowing or to payment of rent, utility or other household bills". The multi-causal nature of over-indebtedness falls in the intersection of banking, budgeting, social and behavioral issues. On the other hand, this problem falls under the scope of the Big Data paradigm, given the large amount of data to process, the different types of data available and the continuous flow of incoming data.

The financial crisis of 2008 triggered household problems such as unemployment, unexpected lower income and welfare retrenchment which combined increased the risk of over-indebtedness in the EU [29]. In 2011, one in almost nine households (11.4%) across the EU were in arrears with their commitments [30]. According to the *Banque de France*, 223 700 cases of over-indebtedness were annually reported in average from 2010 to 2015, and in the third trimester of 2016 the average debt was estimated at 41 470 euros.

Lessons from the crisis resulted in changes to EU legislation aimed to ease and prevent household over-indebtedness. In France, measures for the prevention of over-indebtedness were enforced by the Banking Reform of 2013. The AFECEI (French Association of Credit Institutions and Investment Firms) Charter on banking inclusion and the prevention of over-indebtedness was updated with 2 additional support measures in 2015: (i) Establishing early warning mechanisms for clients in delicate financial positions. (ii) Offering suitable internal responses to clients identified as being in delicate financial positions. In alignment with these reforms, *Groupe BPCE*, the second largest French banking institution with 35 million clients, implemented an early warning mechanism used by two members of the group: *Caisse d'Epargne* (CE) and *Banque Populaire* (BP). The warning mechanism by Groupe BPCE predicts the risk of over-indebtedness 6 months before the event, using available customer data: contracts, bank activities, savings accounts, etc. If a client is identified as in risk the bank proceeds to propose targeted offers to avoid the fall in over-indebtedness. The warning mechanism is described in Table 3.1.

In this chapter we approach over-indebtedness risk prediction from batch and stream perspectives, and propose a data-driven framework (OVI1) as an early warning mechanism to prevent over-indebtedness.

| | Caisse d'Epargne | Banque Populaire |
|---|---|---|
| **Gini Coefficient** | 84.1% | 88.2% |
| **Population** | Clients with an active account and a personal loan. | |
| **DB generation** | Monthly. Previous data is lost | |
| **Data range** | 12-24 months | |
| **Features** | 8 categorical features | |
| **Balancing** | Random Under Sampling | |
| **ML Algorithm** | Logistic Regression | |

**Table 3.1:** Over-indebtedness warning mechanism used by the Groupe BPCE. Baseline performance for each bank is measured using the gini-coefficient.

The rest of the chapter is organized as follows: In Section 3.1 we discuss the challenges related to the over-indebtedness prediction problem. The proposed solution is presented in Section 3.2. Section 3.3 describes the methodology for our experimental evaluation.

## 3.1 A MULTIFACETED PROBLEM

Over-indebtedness prediction is a challenging research topic with important implications at personal, social and economical levels. Early warning mechanisms are deployed by financial institution as a predictive measure for this problem. In the literature, multiple solutions focus on applying a specific learning algorithm with a fixed set of features [115]. This is the case of the warning mechanism by Groupe BPCE where a popular algorithm (logistic regression) is applied on data from two banks within the group. In Table 3.1, we see a difference of $\sim 4\%$ in performance despite the fact that the model is trained using the same learning algorithm and the same set of features. This shows that data from each bank has its own internal structure which results in different performance. Thus, we argue that the practice of *fitting* the data to a pre-defined machine learning pipeline represents a significant compromise.

Under this scenario, we present a framework that addresses the multiple challenges in the prediction of over-indebtedness, namely: feature selection, data balancing, learning method and model evaluation. The framework involves a common set of tasks with a tailored *configuration* for different banks. The framework is designed to adapt to the data and not the other way around. This is similar to the treatment of some diseases, where a treatment is personalized given the general health state of the patient, life habits, allergies, etc.

Furthermore, the complexity of financial markets results in continuous changes in data distribution and in the relationship between features and the class they represent. The traits that define over-indebtedness risk today might not be the same in a few months. Under this scenario, conventional *batch* models get obsolete over time and have to be replaced.

In the following we discuss the challenges related to designing and implementing a warning mechanism for over-indebtedness risk prediction.

### 3.1.1 Feature Selection

In supervised learning, the more information (features) to describe a phenomenon, the better. However, high dimensional data sets usually suffer of *redundancy* and *noise*. When considering the inclusion of multiple features we must ensure that (i) they are relevant to the problem we are facing and (ii) they provide additional information. Feature quality is preferred over quantity. *Redundant* features can hinder learning since we could end *over-fitting* over superfluous information. On the other hand, *unrelated* features introduce noise. Another aspect to consider is the computational burden associated to processing redundant or unrelated features.

*Feature Selection* consists in automatically finding a subset of features $f \in \mathcal{F}$, where $\mathcal{F}$ is the set of all available features, such that $f$ maximizes the prediction power of our model while keeping its generalization (avoids over-fitting). Feature selection also reduces computational burden and processing time by reducing dimensionality [55]. Feature selection techniques are divided [23] into:

- **Classifier dependent**: *Wrapper methods* use classifier accuracy as lead during the selection process. *Embedded methods* perform feature selection during the training stage of a classifier.
- **Classifier-independent**: *Filter methods* find generic subsets of statistically relevant features under the assumption that more information will lead to better learning and classification.

In terms of robustness against over-fitting, filters result in better generalization than embedded methods, which in turn prevail over wrappers. In terms of speed, filters are in general faster, followed by embedded methods, while wrappers are the slowest.

Solutions for over-indebtedness prediction in the literature are usually bounded to a specific set of features (either automatically or manually selected) and assume that they can be replicated. In practice, this is not the case, since data collection in real-world applications are difficult to change. Given that many applications are limited to available data we focus on the *selection process* rather than in the resulting set of selected features.

### 3.1.2 Data Balancing

Data for over-indebtedness prediction is inherently unbalanced. In a *normal* and *healthy* financial scenario, the majority of people belongs to the not-over-indebted class while a very small percentage falls in the over-indebted class. In both CE and BP, the over-indebted class represents less than 1% of the data. This scenario represents a challenge for learning since the over-indebted class is extremely under-represented. Most supervised learning algorithms do not perform well on unbalanced data, and those that support it may fail when the imbalance is extreme [59]. For the two-classes case, where $|C_{maj}|$ and $|C_{min}|$ are the size of the majority and minority classes respectively, extreme class imbalance is defined as:

$$|C_{maj}| \gg |C_{min}| \tag{3.1}$$

*Data Balancing* automatically reduces class imbalance while keeping the underlying distribution of the data; the goal is to provide enough information about each class to the learning algorithm. Reducing class imbalance can also reduce over-fitting. An extensive review of imbalanced learning with two-classes is provided in [59]. Data balancing techniques are divided into two major groups:

- **Under-sampling**: select a subset from the majority class $C'_{maj} \in C_{maj}$ such that

$$|C'_{maj}| \approx |C_{min}| \times ratio \tag{3.2}$$

- **Over-sampling**: synthetically add minority class samples via an over-sampling function $f(C_{min}) = C'_{min}$

$$|C_{maj}| \times ratio \approx |C'_{min}| \tag{3.3}$$

The desired class balance *ratio* in Eq. 3.2 and Eq. 3.3 is usually 1.0 for $1:1$ class balance.

### 3.1.3 Supervised Learning

Selecting a supervised learning algorithm is not a trivial task. Moreover, trying to *fit* the data into a pre-selected algorithm compromises the information contained in the data. Understanding how learning algorithms work and their limitations is fundamental. Other factors to consider are:

- The variety or form of the **data**; the availability and capacity to get new/more data; data reliability; the linear or non-linear nature of the data, etc.

  **Model** interpretability and complexity are key factors for real-world applications. Interpretability is crucial when decisions

based on predictive models have a direct impact on people's lives. This discourages the usage of *black-box* models since there are no reliable means to explain the internal mechanisms applied and the corresponding results.

- **System constraints** and **resources** (usually over-looked) play a major role on the design and implementation of real-world solutions. It is essential to keep in mind constraints in terms of computational power, data storage, scalability, etc.

*Logistic Regression* (LR) is a well known statistical method extensively used in economics due to its interpretability and ease of use (minimal parameter tuning). This technique is used in the warning mechanism by Groupe BPCE. The LR algorithm models the posterior probabilities of the class $y$ via linear functions of $\vec{x}$ inputs. The cost function minimized for the binary class case is:

$$cost(\theta) = -\frac{1}{m}\Big[\sum_{i=1}^{m} y^{(i)} log(h_\theta(\vec{x}^{(i)}) + (1-y^{(i)}) log(1-h_\theta(\vec{x}^{(i)})\Big] \tag{3.4}$$

where $h_\theta = 1/(1+e^{-\theta^\intercal \vec{x}})$ is the logistic (sigmoid) function and $\theta$ is the set of parameters to fit.

On the other hand, *Tree Ensembles* such as *Random Forest* (RF) and *Boosting Trees* (BT) are state-of-the-art machine learning methods for classification [25], popular in the research community. LR is a good option when the number of samples is small and features show linear nature and simple interactions. Tree Ensembles are robust to noise and handle well non-linearity and complex feature interactions, specially if the number of samples is large. Tree Ensembles interpretability,

**Table 3.2:** Comparison between logistic regression and tree ensembles.

| | **Logistic Regression** | **Tree Ensembles** |
|---|---|---|
| Parametric | ✓ | ✗ |
| Decision boundary | Single Linear | Single/Multiple Non-linear |
| Learns feature interactions | ✗ | ✓ |
| Robust to noise (irrelevant features) | ✗ | ✓ |
| Performs well with few samples | ✓ | ✗ |
| Interpretable | ✓ | ✓* |
| Tuning Parameters | Minimal | Some (RF) Several (BT) |

* Affected by the ensemble size.

although easy, is affected by the size and number of trained trees. In terms of tuning parameters, LR is the simplest, only requiring one parameter for regularization. RF has parameters to control model complexity and computational resources. Boosting Trees are known for the large number of tuning parameters and finding the best setting for a specific task is a dedicated research topic. A comparison between logistic regression and tree ensembles is available in Table 3.2.

### 3.1.4 Stream Learning

Stream data mining algorithms are a compelling option for over-indebtedness prediction since they handle big data sets in an efficient way and the generated model is continuously updated. In the current approach (Table 3.1) Groupe BPCE collects data over time (monthly). Given that class definitions *evolve* to reflect changes in the financial system, batch models have a finite time span of optimal performance. Furthermore, batch models require additional work to evaluate and track performance over time, determine when the model has been rendered obsolete and trigger the generation of a new model. Since batch models only consider data from the current month, previous knowledge is lost. On the contrary, stream models are continuously updated taking into account the chronological order of data. Newer data has a bigger impact in the updated model than older data.

Bagging is an ensemble method used to improve accuracy. *Online Bagging* (OB) [93] is a popular stream technique where each example is given a weight according to $Poisson(1)$ instead of sampling with replacement as in the batch case. Two base learners commonly used along OB are *Hoeffding Trees* (HT) [35] and *Random Hoeffding Trees* (RHT). HT are a stream version of Decision Trees. RHT use just a part of the features taken randomly, combined with OB the result is a kind of random forest for data streams.

## 3.2 A DATA-DRIVEN WARNINING MECHANISM

In this section, we describe Over-Indebtedness Risk Prediction 1 (OVI1), a novel framework designed for inter-bank application. Depicted in Figure 3.1, OVI1 consists of four common tasks over three stages:

(i) **Pre-processing**, involves feature selection and data balancing. The input is a high dimensional and extremely unbalanced data set and the output is a lower dimension and balanced data set.

(ii) **Training**, where a supervised learning algorithm trains a predictive model using the pre-processed data.

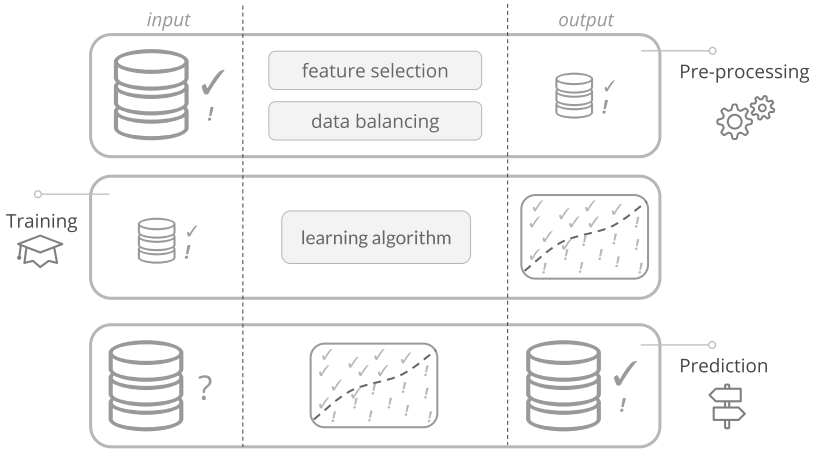



**Figure 3.1:** The OVI1 framework incorporates a set of common tasks, whereas a tailored configuration ensures optimal performance over different data distributions for inter-bank application.

(iii) **Prediction**, where a highly unbalanced test data is classified by the trained model.

We provide a tailored configuration for each bank based on the intrinsic characteristics of their data. Finally, two versions of the framework are available depending on the type of learning paradigm used: OVI1-BATCH and OVI1-STREAM.

Selecting the proper criteria to evaluate performance is vital. The performance of the current approach is measured using the *gini coefficient* (*gini ratio* or *gini index*) which is an interpretation of the Area Under Receiver Operating Characteristic (AUROC) curve

$$\textit{gini coefficient} = (2 \times \text{AUROC}) - 1 \tag{3.5}$$

The gini coefficient discerns between *discriminative* models and those that are only *representative*. This is particularly important when dealing with unbalanced data. Nonetheless, using a single metric to compare predictive models performance is sometimes obscure and can be misleading, specially since the gini coefficient may provide overly optimistic performance results on highly skewed data [59].

Table 3.3 shows an example of the performance paradox on unbalanced data. Using a single metric criteria (gini coefficient), model

**Table 3.3:** Different to single-metric criteria (gini coefficient), multi-metric criteria (including sensitivity and specificity) shows that model *B* is biased against the positive (over-indebted) class.

| *model* | *rank single* | *rank mult.* | *gini-coeff.* | *sn* | *sp* |
|---|---|---|---|---|---|
| *A* | 3rd. | 2nd. | 92.5% | 92.9% | 88.7% |
| *B* | 1st. | 3rd. | 93.4% | **69.9%** | 91.2% |
| *C* | 2nd. | 1st. | 93.3% | 92.3% | 89.8% |

*B* seems the best option compared to *A* and *C*. If additional metrics are considered, namely *sensitivity* (sn) and *specificity* (sp), we see that model *B* is biased against the positive (over-indebted) class, it sacrifices sensitivity over specificity to achieve a better gini coefficient. Model *C* represents the best compromise, it keeps a good gini coefficient while being fair among classes.

The usage of sensitivity and specificity follows two practical reasons:

(i) They are linearly related to the gini coefficient, the ROC space is also known as the sensitivity vs (1 - specificity) plot, and

(ii) They provide insight into the model fairness: sensitivity is the ratio of correctly identified over-indebted instances while specificity is the ratio of correctly identified not-over-indebted instances.

The specificity-sensitivity trade-off is especially important considering the extreme class imbalance and the *associated cost* to each class. Although the main objective is to identify people in risk of over-indebtedness, it is also important to consider the implications of misclassifying a person as being at risk. From Eq. 3.1 it follows that

$$N_{FP} \gg N_{FN} \tag{3.6}$$

where $N_{FP}$ and $N_{FN}$ are the number of *false positives* (FP) and *false negatives* (FN) respectively. The *associated cost* is defined as

$$cost = cost_{FP} + cost_{FN} \tag{3.7}$$

Individual cost for FP and FN are given by

$$cost_{FP} = N_{FP} \times c_{fp} \tag{3.8}$$

$$cost_{FN} = N_{FN} \times c_{fn} \tag{3.9}$$

where, $c_{fp}$ is the cost of misclassifying a person as in risk and $c_{fn}$ the cost of misclassifying a person as not in risk. Although the value of $c_{fp}$ can be simply determined in financial terms, $c_{fn}$ might not be measurable only by economic factors. The estimation of the associated cost is out of the scope of this research, nevertheless we take it into consideration when evaluating the performance of predictive models. A model is considered good if it optimizes the gini coefficient while optimizing **both** sensitivity and specificity

$$\arg\max_{sn,sp} \Big( gini\ coefficient(sn, sp) - |sn - sp| \Big) \tag{3.10}$$

We define a multi-metric evaluation criteria where:

1. Gini-coefficient is the over-all performance metric.

2. Higher sensitivity is preferred (identify people in risk of over-indebtedness). Models that sacrifice sensitivity over specificity are ignored.
3. Given the class imbalance, $specificity \gtrsim sensitivity$ is expected. Lower specificity means that the model fails to retain discriminative information from the non-over-indebted class.
4. Considering Eq. 3.6, small variations from highest sensitivity are justified in the specificity-sensitivity trade-off given their impact on the associated cost (Eq. 3.7).

### 3.2.1 Generalization

OVI1 is a comprehensive framework for a multifaceted problem and is intended to function as a template for defining custom solutions for over-indebtedness prediction. The generic nature of OVI1 is designed with replicability in mind. In this chapter, we show how we find two tailored configurations for two banks using real-data, a process that can be applied to other banking institutions inside and outside the group. Additionally, we believe that OVI1 can be easily adapted to other domains under similar conditions. One example is failure prediction on airplanes where misclassifying failures has major implications, the amount of failure examples is extremely small, and the set of relevant features may vary depending on the type of failure to predict and the components of the aircraft.

## 3.3 EXPERIMENTAL EVALUATION

Real-world data sets provided by Groupe BPCE are outlined in Table 3.4. Data is fully anonymized and comprises economic and demographic features. Additional to a common subset of features (used in the baseline), specific features for each bank are included in the data sets. It is important to point out that the data sets are similar in the sense that both contain economic and demographic data and are generated by institutions within the same financial group, but there are differences in terms of size (number of instances and number of features) and the intrinsic characteristics of the data.

Due to privacy policies, we cannot provide a detailed description of the features in the data sets, we refer the reader to [5, 81, 82] for examples of similar financial features and [72] for demographic features. It is also important to remember that the automatic feature selection process should be replicated rather than the set of features. The rationale for this is two-folded:

(i) Selected features are not transferable between data sets with different data distributions.

**Table 3.4:** Characteristics of real-world data provided by Groupe BPCE. Notice the extreme imbalance between over-indebted and not-over-indebted classes.

| | *Caisse d'Epargne* | *Banque Populaire* |
|---|---|---|
| **Instances** | 1 676 453 | 705 736 |
| **Features** | 193 | 83 |
| **Over-Indebted** | **0.37%** | **0.24%** |
| **Not-Over-Indebted** | 99.63% | 99.76% |
| **Size (MB)** | 1776 | 254 |
| **Date** | September 2014 | June 2015 |

(ii) Data collection mechanisms in real-world applications are complex and hard to change. We are generally bounded to *available* features, as in the case for the BP and CE data.

In order to define the best configuration for each bank, we compare framework performance using multiple configurations against the baseline corresponding to the mechanism by BPCE. We use real-world labeled data and validate results by 10-fold cross-validation. Table 3.5 lists the techniques for feature selection, data balancing and supervised learning used in our tests.

For feature selection, we test 2 embedded methods and 4 filter methods. Embedded methods: *rfe* uses an external estimator (we use LR in our tests) to recursively select a subset of features. *ig* uses the feature ranking from tree-based algorithms to select relevant features. We use different tree based classifiers: decision trees, extreme random trees, random forest, gradient boosting trees, and extreme gradient boosting. Tested filter methods include 2 uni-variate methods, $\chi^2$ and *var*; and 2 mutual information selection methods, *jmi* and *cmim*. *jmi* looks for complementary features in the feature space and *cmim* iteratively selects features that maximize mutual information. We use 10 feature selection methods $\times$ 50 subset sizes $\times$ 6 ML algorithms, for a total of 300 feature selection tests.

For data balancing, we test 6 under-sampling techniques and 5 over-sampling techniques. Under-sampling: *rus*, where a subset of the majority class is selected at random. *nm1*, *nm2*, *nm3* use the kNN classifier to drive under-sampling. *ee* generates several subsets to be used in an ensemble learning system. For our tests we modify *ee*, instead of training an ensemble system, a single classifier is trained using the merge of all subsets obtained from *ee*. This way more samples of the majority class are used for training. Although duplicated samples from the minority class are introduced (to achieve class balance), they do not provide additional information and have no impact on the model. *bc* systematically performs under-sampling in a supervised learning fashion. Over-sampling: *SMOTE* creates minority

**Table 3.5**: Techniques used in our tests.

| | |
|---|---|
| *Feature Selection* | |
| Recursive Feature Elimination (rfe) | Embedded method |
| Information Gain (ig) | Embedded method |
| Chi-squared test ($\chi^2$) | Filter method |
| Variance (var) | Filter method |
| Joint Mutual Information (jmi) [121] | Filter method |
| Conditional Mutual | Filter method |
| Information Maximization (cmim) [40] | |
| *Data Balancing* | |
| Random under-sampling (rus) | Under-sampling |
| NearMiss-1,2,3 (nm1,2,3) [123] | Under-sampling |
| EasyEnsemble (ee) [78] | Under-sampling |
| BalanceCascade (bc) [78] | Under-sampling |
| Synthetic Minority | Over-sampling |
| Over-sampling Technique (SMOTE) [26] | |
| SMOTE Borderline 1,2 (smoteb1,2) [58] | Over-sampling |
| SMOTE+Edited Nearest Neighbors (senn) [10] | Over-sampling |
| SMOTE+Tomek links (stl) [10] | Over-sampling |
| *Model Training* | |
| Logistic Regression (LR) | Batch Learning |
| Decision Trees (DT) | Batch Learning |
| Random Forest (RF) | Batch Learning |
| Extremely Randomized Trees (ET) [52] | Batch Learning |
| Gradient Boosting Trees (GBT) | Batch Learning |
| Extreme Gradient Boosting (XGB) [28] | Batch Learning |
| Online Bagging (OB) [93] | Stream Learning |
| Hoeffding Trees (HT) [35] | Stream Learning |
| Random Hoeffding Trees (RHT) | Stream Learning |

class samples by adding noise to existing samples. *smoteb1, smoteb2* favor synthetic samples in the border between classes. *senn* and *stl* include data cleaning to reduce overlapping. We set as goal a 1:1 class ratio. In the case of over-sampling, 2 steps are performed due to extreme class imbalance. In Step 1, we use over-sampling to increase the minority class from $<$ 1% to 5% and 10%. In Step 2, we reduce the majority class (reaching 1:1 ratio) via under-sampling (*rus, nm1, nm2 or nm3*). Under-sampling tests include 6 under-techniques whereas over-sampling tests are composed by 5 over-sampling techniques $\times$ 2 over-sampling ratios $\times$ 4 under-sampling techniques. In total 46 data balancing tests are performed.

For batch learning, we test logistic regression (LR) and tree based algorithms decision trees (DT), extreme random trees (ET), random forest (RF), gradient boosting trees (GBT) and extreme gradient boosting (XGB). As discussed in Section 3.1.3, parameter tuning is a characteristic of tree ensembles and can provide advantage (or disadvantage) against other techniques such as LR. OVI1 is designed to be independent of parameter tuning, thus, for all tree ensembles algorithms, the number of estimators is set to 100 and the rest of parameters are kept to default values.

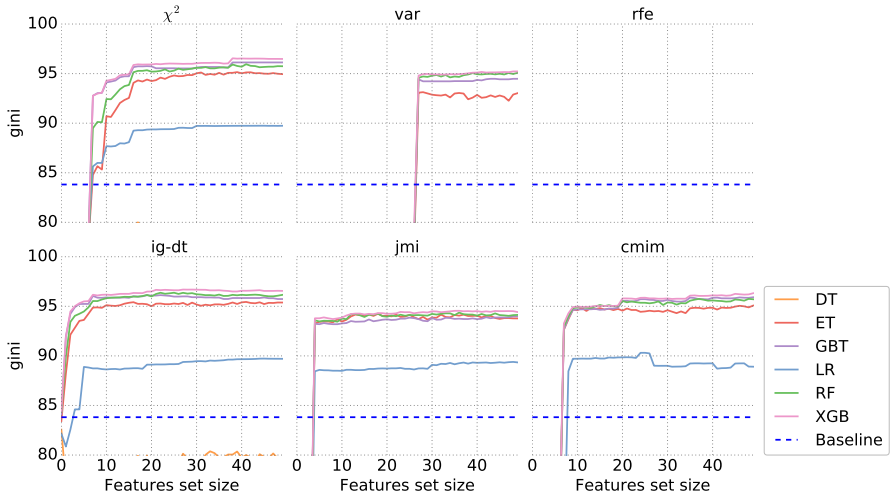
Finally, for stream learning we test OB with HT and RHT as base learners. As discussed in Section 3.1.4, stream algorithms process a single instance at a time and the model is updated over time as more samples are processed. A standard procedure to transform batch data into data streams is to perform multiple passes over the data to simulate data flow (usually in the order of 50,000+ samples). Class concepts are reinforced by exposing stream learners to repeated patterns over time. In our tests we perform 10 passes over the CE data and 30 passes for BP. Since the number of training samples in the BP data set is small in the context of stream learning, an additional test is performed by introducing data collected in the previous month.

## 3.4 RESULTS

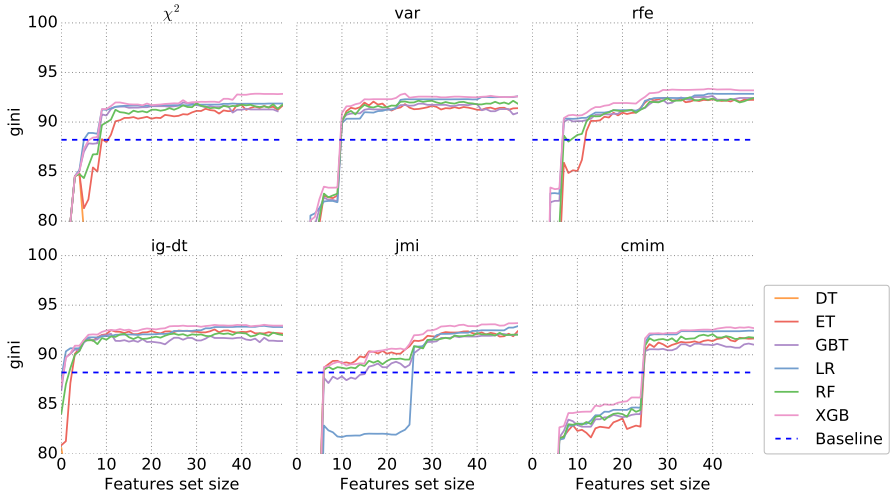
In this section we discuss the result from our experiments. Results are analyzed by the challenge addressed: feature selection, data balancing and learning paradigm.

### 3.4.1 Feature Selection

Figure 3.2 shows the impact of feature selection on predictive models performance. Performance increases as more features are added. We found that <50 features are required to reach the performance plateau where gains from additional features are marginal. $\chi^2$ performs as expected on both data sets, while *var* has a *cold-start* on CE. Nonetheless, the over-simplicity of these two methods implies an important compromise in terms of exploiting hidden information or feature interaction in the data. Unexpectedly, although *rfe* performs well in BP, it fails for CE. *cmim*, *jmi* and *ig-dt* perform well on both data sets. We use *ig-dt* to represent tree-based feature selection techniques since test results show that performance gains from different tree classifiers are marginal. Notice that while in BP most algorithms eventually reach a similar plateau, in CE, Figure 3.2a, LR reaches a lower plateau than all the ensemble trees. This is evidence of the non-linear nature of the CE data, which explains why *rfe* (using a linear estimator) fails. We also




**(a)** CE data



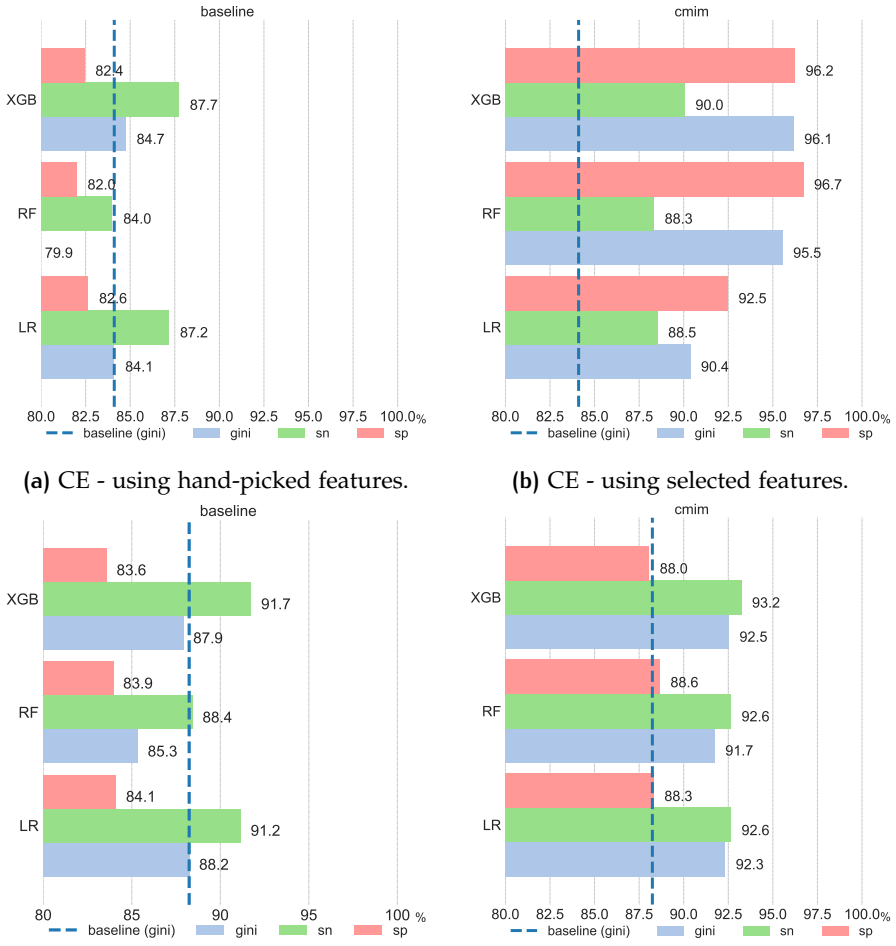

**(b)** BP data

**Figure 3.2:** Feature selection tests. Performance over increasing number of selected features.

observe that the plateau for *jmi* is slightly below the plateaus for *cmim* and *ig-dt*.

In order to find the appropriate feature selection technique for inter-bank application within the group, performance on both data sets are considered. We apply the multi-metric criteria discussed inSection 3.2 to evaluate model performance. Although *ig-dt* performs well in both data sets it is important to note the model dependency and risk of over-fitting inherent to embedded methods. *cmim* represents the best compromise between inter-bank application and model performance, with RF and XGB as top performers.

Figure 3.3 shows the performance of different learning algorithms using the sets of numerical and categorical features selected using *cmim*, due to space limitations we only display the baseline and the top performers (LR, RF, and XGB). Interestingly, selected features for CE




**(a)** CE - using hand-picked features.

**(b)** CE - using selected features.

**(c)** BP - using hand-picked features.

**(d)** BP - using selected features.

**Figure 3.3:** Feature selection tests. Performance of baseline hand-picked features vs automatically selected features (*cmim*), using 22 (CE) and 34 (BP) numerical and categorical features.

only include 2 out of the 8 features in the hand-picked set. This shows that the non-linear characteristics of the data and the complex interaction between features is hard for humans to grasp. On the other hand, selected features for BP include 5 out of 8 features in the hand-picked set. Additionally, we observe that LR performance in BP is close to RF and XGB, which provides further evidence of the linear nature of the BP data. In general, we see the benefit of using automatic feature selection, performance gains result from features already available which are ignored in the baseline solution. Regarding the impact of feature selection on the specificity-sensitivity trade-off. In CE, specificity becomes greater than sensitivity which is desired since it means that discriminative information from the majority class is retained. In BP, we see that the gap is reduced without sacrificing sensitivity but specificity still trails behind. We conclude that available features fail

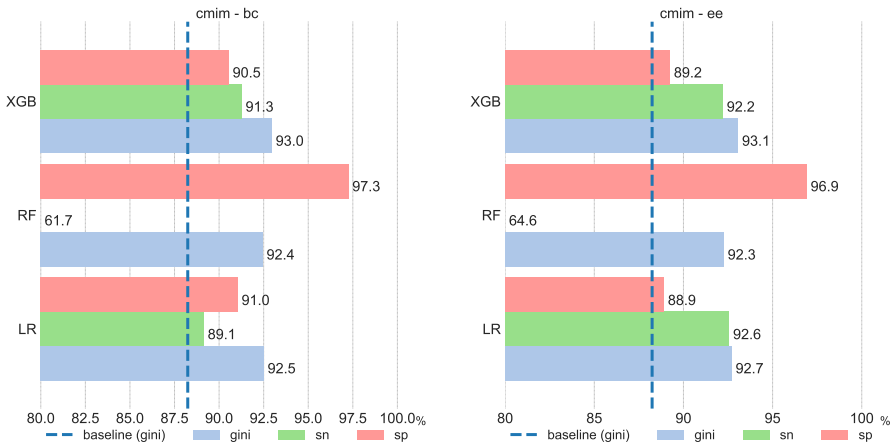
to optimally describe the majority class. Further feature engineering, e.g. latent and temporal features, could correct this behavior.

### 3.4.2 Data Balancing

Results in Section 3.4.1 show that proper feature selection plays a major role in performance. Data balancing can be used to *fine tune* a predictive model by retaining discriminative information from the majority class. In Figure 3.3b, we see that *cmim* + *rus* results in a good specificity-sensitivity trade-off for CE. We found that applying other balancing techniques for CE results in marginal gains or even lower performance. On the other hand, predictive models using *cmim* + *rus* on BP show better sensitivity, see Figure 3.3d. Therefore, we focus on the BP case for discussing fine tuning predictive models via data balancing.

Chaining *over-sampling* (minority class) and *under-sampling* (majority class) results in adding noisy instances to the minority class while adding even more instances to the majority class. Test results show that this harms performance, since the trained model is less certain about the minority class (due noise) and more certain about the majority class (due to additional instances), it sacrifices sensitivity over specificity.

From *under-sampling* tests, we find that *nm3* performs better than *nm1* and *nm1* performs better than *nm2* but all three fail to outperform *rus*. Moreover, *nm2* complexity restricts its applicability on big data sets. The best *under-sampling* performers are *bc* and the modified *ee*. Due to space limitations only the top performers (*cmim* + [*bc*,*ee*]) are shown in Figure 3.4. Notice the negative impact of *bc* and *ee* on RF, which improves gini-coefficient by further increasing the sensitivity-




**(a)** BP - *BalanceCascade* **(b)** BP - Modified *EasyEnsemble*

**Figure 3.4:** BP data balancing. In general, balance cascade and easy ensemble reduce the specificity-sensitivity gap. This is not the case for RF, which tends to favor specificity over sensitivity.

specificity gap. On the other hand, XGB improves gini-coefficient while reducing the gap. Although *bc* does a better job at reducing the sensitivity-specificity gap, its classifier dependency adds one degree of freedom to the framework. Therefore we recommend *ee* given its easy implementation and classifier independence. Considering the extreme class imbalance, the small loss in sensitivity (-1.7 instances correctly classified as in risk), is compensated by the gains in specificity (+850.3 instances correctly classified as NOT in risk). The decision to use *rus* or *ee* depends on the associated cost given by Eq. 3.7. Despite the fact that data balancing can reduce the specificity-sensitivity gap, feature engineering is recommended in order to find features with better discriminative power.

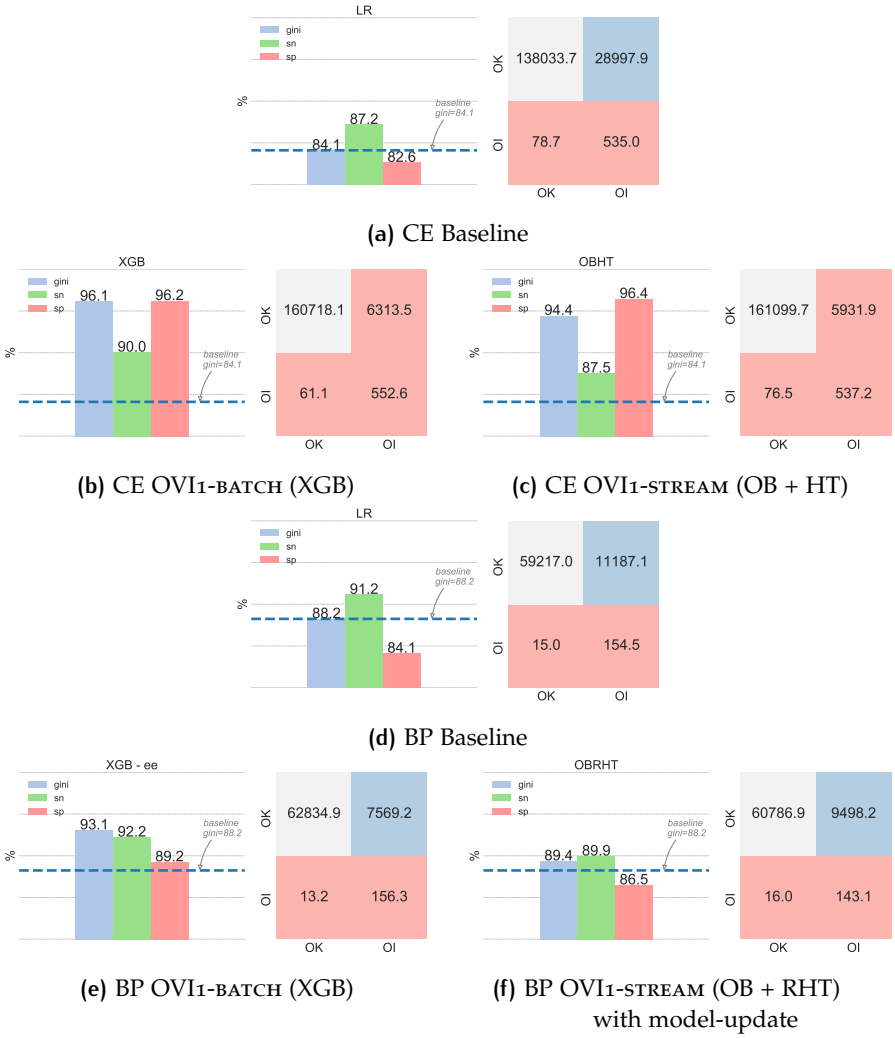
### 3.4.3 Batch vs Stream Learning

Table 3.6 summarizes the top results achieved by OVI1-BATCH using RF and XGB and by OVI1-STREAM using OB with HT and RHT with and without model-update. In Figure 3.5, the top results are shown, including the actual number of clients classified (confusion matrix). For both tests, batch and streaming, we use the best set of features (per bank) found via *cmim*.

For the batch setting (Figure 3.5), XGB over-performs RF while reducing the sensitivity-specificity gap. CE and BP data show improvements in both specificity and sensitivity compared with the baseline. For CE, OVI1-BATCH with XGB reaches performance gains of +12.0% gini coefficient, +2.8% sensitivity and 14.4% specificity. Large gains are explained by the non-linear nature of the CE data, which is better handled by tree ensembles. Additionally, greater specificity (while

**Table 3.6:** OVI1 framework (batch and stream) against baseline.

| | mechanism | f-sel | d-bal | model | gini | sn | sp |
|---|---|---|---|---|---|---|---|
| *Caisse d'Epargne* | baseline | manual | rus | LR | 84.1 | 87.2 | 82.6 |
| | OVI1-BATCH | cmim | rus | RF | 95.5 | 88.3 | **96.7** |
| | OVI1-BATCH | cmim | rus | XGB | **96.1** | **90.0** | 96.2 |
| | OVI1-STREAM | cmim | rus | OB HT static | **94.4** | **87.5** | **96.4** |
| | OVI1-STREAM | cmim | rus | OB RHT static | 92.7 | 86.0 | 95.2 |
| *Banque Populaire* | baseline | manual | rus | LR | 88.2 | 91.2 | 84.1 |
| | OVI1-BATCH | cmim | rus | RF | 91.7 | **92.6** | 88.6 |
| | OVI1-BATCH | cmim | ee | XGB | 93.1 | 92.2 | **89.2** |
| | OVI1-STREAM | cmim | rus | OB RHT static | 86.2 | 86.6 | 85.3 |
| | OVI1-STREAM | cmim | rus | OB RHT updated | **89.4** | 89.9 | **86.5** |




**(a)** CE Baseline

**(b)** CE OVI1-BATCH (XGB)

**(c)** CE OVI1-STREAM (OB + HT)

**(d)** BP Baseline

**(e)** BP OVI1-BATCH (XGB)

**(f)** BP OVI1-STREAM (OB + RHT) with model-update

**Figure 3.5:** Baseline vs top performers. For CE, OVI1-BATCH and OVI1-STREAM provide over-all better performance and correct the low specificity in the baseline. Gains in BP are lower which is expected given that the baseline performance is better than in CE. The specificity-sensitivity gap is reduced but specificity is lower, better features are required to describe the majority class. Although gains for OVI1-STREAM are marginal in BP, it is important to remember the major role that automatic model updates play when predicting over evolving financial data. Confusion matrices are the average of 10-fold cross validation.

also improving sensitivity) implies the proper handling of the majority class (not-over-indebted). For BP, OVI1-BATCH with XGB achieves +4.9% gini coefficient, +1.0% sensitivity and +5.1% specificity. Even though data balancing (modified *ee*) reduces the specificity-sensitivity gap, specificity trails behind sensitivity as in the baseline. It is very likely that the available *pool* of features fails to optimally describe the majority class. In general, performance gains for BP are lower than those for CE primary due to lower specificity. However, it should be

noted that, in the baseline, performance is already better in BP compared to CE. We see that OVI1-BATCH represents optimal performance for CE, and a boost in performance for BP.

Figure 3.5 shows the results of OVI1-STREAM. The same behavior in the sensitivity-specificity trade-off is observed, specificity is greater than sensitivity in CE and lower in BP. For CE, OVI1-STREAM with a static model generated using the current month data and OB with HT outperforms the baseline with gains of +10.3% gini coefficient, +0.3% sensitivity and +13.8% specificity. On the contrary, the smaller number of samples on the BP data set results in under-performance of stream learning techniques which fall short against the baseline when a static model is trained using only the data from the current month. When consecutive months (current + previous) are considered, OVI1-STREAM using OB and RHT results in +1.3% gini coefficient, -1.1% sensitivity and +2.5% specificity. The small loss in sensitivity is compensated by the gain in specificity (and gini coefficient). In CE we see that OVI1-STREAM provides a large performance boost similar to OVI1-BATCH. On the other hand, performance of OVI1-STREAM for BP is lower than OVI1-BATCH but still above the baseline. Despite this, we consider that OVI1-STREAM for BP is a good option considering the model is continuously updated (adapts) adding robustness against changes in debt data, which is not be handled by *batch* models. Additionally, it is worth noting the smaller size of the BP data and the limitations of our tests to 2 months of data, while stream learning methods are known to perform better when exposed to large amounts of data (as in CE). Finally, two characteristics of stream learning are exploited when using consecutive data: (i) the model is automatically updated and (ii) chronological order is retained ensuring that the predictive model is consistent with current data.

# 4 MISSING DATA IMPUTATION AT SCALE

Missing data is a common phenomenon in real-world applications. It can be introduced during data collection by human manipulation or by sensor failures, or by hardware/software failures during data storage and/or transmission. The average amount of missing data is estimated in a range between 5% and 20% [101, 114]. Missing data has a negative effect on performance of supervised learning methods, according to [2], ratios between 5–15% require the usage of sophisticated methods while above 15% of missing values can compromise data interpretation.

Under the big data paradigm, scalable and practical approaches are required to handle missing data. Proper imputation methods are essential to provide complete data to supervised learning algorithms that rely on the quality and quantity of data. In this chapter, we present an effective and scalable imputation method for numerical and nominal data, which mitigates the impact of MAR and MCAR data (Section 2.3) on binary and multi-class classification.

The rest of this chapter is organized as follows. Section 4.1 introduces our imputation method. The methodology for the experimental evaluation is described in Section 4.2 and results are discussed in Section 4.3.

## 4.1 A MODEL-BASED IMPUTATION METHOD

A predominantly characteristic of data used to train predictive models is the underlying presence of correlation/interaction between attributes [23, 55, 76]. Under this assumption, we introduce Cascade Imputation (CIM), a model-based incremental imputation method. CIM casts the imputation process as a set of classification/regression tasks where unobserved values are imputed on a supervised learning fashion, in other words, a predictive model for missing data is generated based on input-response samples.

Similarly, we introduce an extension of CIM to impute multiple attributes at a time. This is consistent with the multi-label learning problem where a set of labels are associated to each instance. Accordingly, we develop two multi-label versions of CIM: Multi-label Cascade Imputation (Multi-CIM) which builds upon Classifier Chains [105], and Ensemble-Multi-label Cascade Imputation (eMulti-CIM) which aims to improve generalization by training a collection of models in the form of an ensemble. To the extent of our knowledge, this is the first work to address the simultaneous learning/prediction of nominal

and numerical data in multi-label learning, previous works only focus on a single type of data at a time.

### 4.1.1 Cascade Imputation

In the following, we describe the main steps performed by CIM. Given an incomplete data set $D = (X, y)$, we want to find the corresponding imputed data set $D' = (X', y)$. Figure 4.1a shows the original positions of missing data (in red) in $X$. First, CIM splits $X$ column-wise, keeping complete data to the left and incomplete data to the right. Columns are sorted (ascending order) based on the amount of missing values. Figure 4.1b shows the updated column order after sorting.

CIM iterates trough the columns with missing values, incrementally imputing attribute-columns via predictive models, hence cascade imputation. For each column with missing values $i$, CIM trains a classification model if the attribute is nominal or a regression model if it is numerical. On each iteration $i$, CIM sorts rows in $X$, placing all instances where the value of $i$ is known (observed) at the top and instances with unknown (missing) values at the bottom. Figure 4.1c shows the updated row positions after sorting.

Columns $0 \rightarrow i$ correspond to the setup of a classification/regression problem. Known inputs $X_{train}$ and their responses $y_{train}$ are used to train a model $h_i$. Imputed values result from applying the corresponding model $imputed = h_i(X_{predict})$. This process is repeated until all attributes with missing values have been processed. Imputed values (in green) are used on following iterations of the algorithm, Figure 4.1d. When done, $X'$ rows are sorted back to the original order and we have $D' = (X', y)$. The pseudo code for CIM is shown in Algorithm 1.

Additionally, CIM calculates for each instance $j$ a *missingness* weight $w_j$, Eq. 4.1, ranging from 0 (all values missing) to 1 (no missing values). In practice, $0 < w_j \leq 1$. $w_j$ represents the level of noise that may be

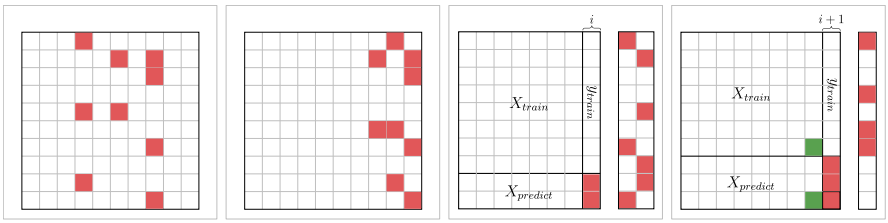



**Figure 4.1:** CIM Steps. (4.1a) Original data with missing values marked in red. (4.1b) Updated positions after sorting attributes by count of missing values. (4.1c-4.1d) Imputation iterations, repeated until data set is complete. Imputed values in green are used in following iterations of the algorithm.

**Algorithm 1:** CASCADE IMPUTATION

**Input** : Matrix $X$ with missing values
**Output**: Matrix $X'$ with imputed values

**Function** `CASCADE(`$X$`)`
  $n$ = number of instances in $X$;
  $d$ = number of attributes in $X$;
  $rows_{original} \leftarrow$ original row order of $X$;
  **for** $i = 0$ **to** $n$ **do**
    $count_{mis}$ = number of missing attributes in row $i$ in $X$;
    $weights[i] = (d - count_{mis})/d$ ; `/* Missingness weights */`
  $X \leftarrow$ sort columns per number of missing values ; `/* Ascending order, left to right */`
  **for** $k = 1$ **to** $d$; `/* Need at least one column for learning */`
  **do**
    **if** *attribute k has missing values* **then**
      $m$ = number of missing values for attribute $k$;
      $rows_{obs} \leftarrow$ rows with observed values for attribute $k$;
      $rows_{mis} \leftarrow$ rows with missing values for attribute $k$;
      $X_{train} = X[rows_{obs}][0 \text{ to } k-1]$;
      $y_{train} = X[rows_{obs}][k]$;
      $X_{predict} = X[rows_{mis}][0 \text{ to } k-1]$;
      **if** *attribute k is categorical* **then**
        $learner \leftarrow$ classifier;
      **else**
        $learner \leftarrow$ regressor;
      $h =$`TRAIN(`$X_{train}, y_{train}$`)` ; `/* Train model */`
      $X[rows_{mis}][k] = h(X_{predict})$ ; `/* Imputation */`
  $X' =$`REINDEX(`$X, rows_{original}$`);`
  $X' =$`CONCATENATE(`$X', weights$`)` ; `/* Include weights */`
  **return** $X'$;

introduced by the imputation process, and can be used in the final classification task to assign higher importance to complete instances over imputed ones.

$$w_j = \frac{|X_j| - |X_{j-mis}|}{|X_j|} \tag{4.1}$$

Although CIM can be paired with different classifiers/regressors, we propose two configurations based on popular algorithms in the research community [25]. CIM-LR uses logistic regression and linear regression, while CIM-RF uses random forest. Logistic and linear

regression are both types of generalized linear models which which can be formulated as:

$$y = X\beta + \epsilon \tag{4.2}$$

where $y$ is a vector of observed values, $X$ is a matrix of row-vectors $x_i$, $\beta$ is a parameter vector and $\epsilon$ is the error or noise. $y$ is continuous in the regression case; while for classification, $y$ is the probability of a categorical outcome, a two states variable for the most basic case. Random forest, a type of tree ensemble, is robust to noise and handles well non-linearity and complex attributes interactions, specially if the number of samples is large. Random forest creates $n$ independent and fully grown decision trees $T_i$. For regression, the average value of the trees is calculated. For classification, a majority vote is applied for the class predictions $C_i(x) = k$ of each tree.

$$\text{Regression,} \quad f(x) = \frac{1}{n}\sum_{i=1}^{n} T_i(x) \tag{4.3}$$

$$\text{Classification,} \quad C(x) = \arg\max_{k} \sum_{i=1}^{n} (C_i(x) = k) \tag{4.4}$$

Resources required by CIM are upper bounded by the classification task on the complete attributes set $X$. On each independent iteration, a subset $X_{train} \in X$ is used. The number of iterations $m$ required is limited by the number of attributes $d$ where $m \leq d$. If all attributes have at least one missing value, then CIM starts the cascade immediately after the attribute with less missing values. For each iteration, if an instance in $X_{train}$ has missing values, it is removed; if a missing value is in $X_{predict}$, it is replaced with zero. Once the cascade ends, CIM imputes the first attribute. The worst case scenario is if all the attributes in a high dimensional data set have missing values. We discuss this scenario in the test section.

### 4.1.2 Multi-label Cascade Imputation

In the previous section, we described how the cascade method works by incrementally imputing a single attribute at time. A natural extension of this process, is to impute multiple attributes during the same step within the cascade. In other words, we can consider the imputation problem as a set of multi-label learning problems. Recall that in single-label learning there is a single target to learn for each instance $j$, this is $(X_j, y_j)$. On the other hand, in multi-label learning, the same instance corresponds to a subset of $l$ possible labels $(X_j, Y_j)$ where $Y_j = \{y_{0_j}, y_{1_j}, \ldots, y_{l_j}\}$. Additionally, labels in $Y_j$ are not mutually

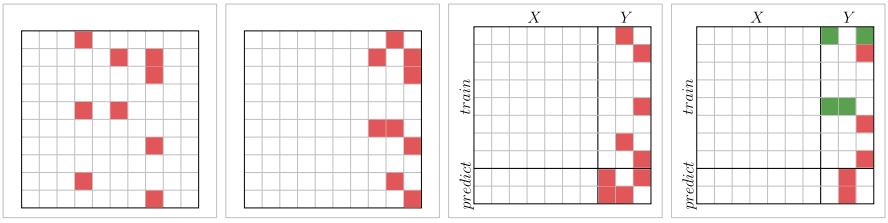



**Figure 4.2:** MULTI-CIM Steps. (4.2a) Original data with missing values marked in red. (4.2b) Updated positions after sorting attributes by count of missing values. (4.2c-4.2d) Multi-label imputation iterations, repeated until data set is complete. Imputed values in green are used in following iterations of the algorithm. Different to CIM, multiple cascade passes might be required.

exclusive (as indeed, such a case is known as multi-*class*). A popular method for multi-label learning is Classifier Chains (CC) [105]. CC exploits correlation between labels by incrementally building binary classifiers for each label. The $i$-th classifier $h_i$ is constructed on $(X, y_i)$, where $y_i \in Y$ and $i = \{0, 1, \ldots, l\}$. After the $h_i$ model is trained, its predictions are used to extend the attribute space for model $h_{i+1}$ corresponding to the next label in the chain. In the case of multi-label regression, a number of regressors are constructed in similar fashion, this is known as Regressors Chains (RC).

MULTI-LABEL CASCADE IMPUTATION (MULTI-CIM), is an extension of CIM to allow imputation of multiple attributes at a time. It is important to note that MULTI-CIM handles different setups where targets (labels) to predict can either be numeric, nominal or a combination of both. The combination of numerical and nominal is a special case given that traditional multi-label solutions have focused on homogeneous data types. As far as we know, this is the first work to employ chain-multi-label models for heterogeneous target types. For the rest of the chapter, we refer to the combination of CC and RC as HYBRID CHAINS (HC).

The steps performed by MULTI-CIM are shown in Figure 4.2. The sorting of columns and rows are performed in a similar way as in CIM, Figure 4.2a shows a data set with missing values. First, MULTI-CIM sorts the columns by the number of missing values, in ascending order from left to right, see Figure 4.2b. In the next step, Figure 4.2c, rows are sorted to build the learning task setup. Different to CIM, MULTI-CIM takes $l$-attributes to impute, from the resulting data arrangement, the learning task corresponds to building a model based on $(X_{train}, Y_{train})$ to impute the missing values as $Y_{predict} = h(X_{predict})$.

Given that the location of missing values across multiple attributes are not necessarily the same, this means that missing values might be introduced into the $X_{train}, Y_{train}$ and $X_{predict}$ matrices. During the learning task, train data instances with missing values are ignored,

while in $X_{predict}$ missing values are replaced with zero. On the other hand, it might be the case that *not* all values in $Y_{predict}$ are actually missing, if a value is known then the prediction is ignored. Consider a missingness matrix $M$ with the same shape as $Y_{predict}$ which indicates if the $ij$-th value is missing in $Y_{predict}$ by setting $M_{ij} = 1$ or $M_{ij} = 0$ if the value is known, then

$$Y_{predict_{ij}} = \begin{cases} h_i(X_{predict_j}), & \text{if } M_{ij} = 1 \\ Y_{predict_{ij}}, & \text{otherwise} \end{cases} \tag{4.5}$$

After imputing the current set of attributes $Y_k$, the cascade continues with the next set of attributes $Y_{k+1}$, Figure 4.2d. Different to CIM, multiple passes of the cascade might be required to impute all missing values. The number of cascade passes is proportionally related to the number of labels $l$ to impute and the ratio of missing values. Notice that, as in CIM, imputed values are used in subsequent iterations within the cascade.

Similar to CC, MULTI-CIM performance is strongly related to the order of the labels within the chain.To overcome this, ENSEMBLE-MULTI-LABEL CASCADE IMPUTATION (EMULTI-CIM) increases diversity by building an ensemble of HC, Figure 4.3. The steps performed by EMULTI-CIM are the same as MULTI-CIM, except that it uses an ensemble of $m$ HC base models instead of a single one to impute multiple attributes simultaneously. To enforce diversity, the order of labels within the chain are randomly permuted, this means that each base models is trained based on different label-order. The final imputation values are obtained by combining the predictions ($P$) of the base models within the ensemble; calculating the average if the attribute contains numeric data or using majority vote if the attribute contains nominal data:

$$Y_{ij} = \begin{cases} \frac{1}{m}\sum_{k=1}^{m} P_{ij}, & \text{if the } j\text{-th attribute is numeric} \\ \arg\max_k \sum_{i=1}^{m} (P_{ij} = k), & \text{if the } j\text{-th attribute is nominal} \end{cases} \tag{4.6}$$

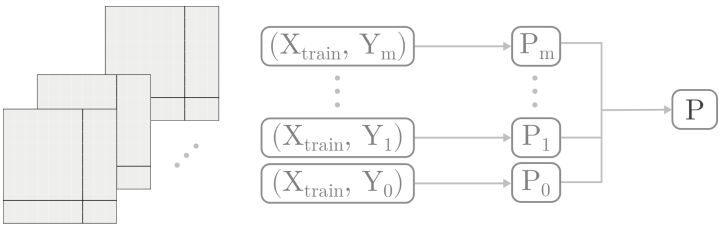



**Figure 4.3:** In EMULTI-CIM, $m$ base models (HC) are used to create an ensemble. Final imputation (prediction) is calculated by merging the outputs of the members of the ensemble.

**Table 4.1:** Data sets. [Domain] B: biology, D: demography, I: image, M: music, T: text; [Type] B: binary, MC: multi-class.

| Name | Domain | Instances | Attributes | | Source[1] | Type |
|---|---|---|---|---|---|---|
| | | | Num. | Nom. | | |
| Adult | D | 48,842 | 6 | 8 | UCI | B |
| Census-IKDD[2] | D | 299,285 | 7 | 33 | UCI | B |
| Music | M | 593 | 72 | 0 | MEKA | B[3] |
| Enron | T | 1,702 | 0 | 1,001 | MEKA | B[3] |
| Genbase | B | 661 | 0 | 1,186 | MULAN | B[3] |
| Llog | T | 1,460 | 0 | 1004 | MEKA | B[3] |
| Medical | T | 978 | 0 | 1,449 | MEKA | B[3] |
| Scene | I | 2,407 | 294 | 0 | MEKA | B[3] |
| Yeast | B | 2,417 | 103 | 0 | MEKA | B[3] |
| Covtype | B | 581,012 | 10 | 44 | UCI | MC |

[1] Repository
[2] Census-Income (KDD)
[3] Multi-label data set. We use the first class-label to test binary classification.

## 4.2 EXPERIMENTAL EVALUATION

We are interested in the impact of imputation on classification. In order to thoroughly evaluate our proposed method, we use 10 data sets (Table 4.1) from a variety of domains and with the following characteristics:

- **Classification type**: Our tests focus on binary and multi-class classification. Although we use multi-label data sets, we test binary classification by learning and evaluating only one label (first label by default), this is equivalent to learning and evaluating under the baseline binary relevance classifier (see Chapter 2). This also serves to control the complexity of our tests given that different classification tasks require different metrics to measure performance.
- **Data type**: Numerical, nominal or a combination of both.
- **Size**: Defined by the number of instances $\times$ the number of attributes. Current trends in data collection, processing and storage have shifted the attention of the research community to larger data sets. Data sets range from small ($< 200K$ values) to large ($> 600K$ values).

**ADULT** Data extracted from the US 1994 Census database. The classes correspond to whether a person makes over 50K a year. Also known as 'Census Income'.

**CENSUS-INCOME KDD** Contains weighted census data from the 1994 and 1995 Current Population Surveys by the US Census Bureau. Each instance includes 41 demographic and employment related variables.

**MUSIC** Corresponds to labeled audio tracks with different moods, e.g. amazed-surprised, happy-pleased, etc. Instances are described by 72 numerical attributes.

**ENRON** Contains annotations for for a subset (1700 messages) of the Enron Email Data set, labelled by the UC Berkeley Enron Email Analysis Project.

**GENBASE** The purpose of this data set is to classify proteins into the most important protein families. Each protein sample is described by 1186 nominal attributes.

**LLOG** A text data set where the task is to classify articles into one or more categories. Includes 1460 articles from the 'Language Log' blog, organized into 75 categories.

**MEDICAL** Information extracted from free text reports by radiologists and labeled into 45 diagnosis codes from the ICD-9-CM (International Classification of Diseases, Ninth Revision, Clinical Modification).

**SCENE** A data set for semantic scene categorization, where each of the instances (images) can belong to one or more classes.

**YEAST** A set of genes, described by 103 attributes, labeled into 14 groups of the Functional Catalogue (annotation scheme for the functional description of proteins).

**COVTYPE** This data set contains data collected over time by the US Forest Service. Classes correspond to cover type in forest on squares of $30 \times 30$ meters.

Different types of tests are performed in order to evaluate multiple aspects of the proposed method. We categorize our tests into: *impact of imputation on classification*, *scalability*, *imputed vs incomplete data for model training* and *multi-label imputation*.

### 4.2.1 Impact on Classification

We compare the performance of classifiers trained on imputed data with multiple ratios of missing data vs the (baseline) performance of a classifier trained using complete data. This test is performed as follows:

1. **Generate missing data.** First, we remove all original missing values from each data set. Then, for each complete set we generate incomplete versions with 4 missing values ratios (5%, 10%, 25% and 50%) and 2 mechanisms (MCAR and MAR). For MCAR, we draw at random a number from a uniform distribution $f_U(v_{i,j}) = x_{i,j}$ for each value $v_{i,j}$ in the data set, if $x_{i,j} \leq t$ then we mark the value as missing. The threshold $t$ is defined by the ratio of missing values. For MAR, we draw at random a pair of attributes $(A, B)$ and a threshold $t_A \in A$. A value $B_j$ is marked as missing if $A_j \leq t_A$. This process is repeated until the ratio of missing values is reached. We generate 10 different versions of each configuration for a total of $(10 \times 4 \times 2 = 80)$ incomplete sets for each complete data set.

2. **Impute missing data.** We compare CIM against 4 well established imputation techniques. *Constant imputation* (Constant) where a constant value is used to fill missing values. We use 0 for numerical values and define a 'missing' class for nominal values. *Simple imputation* (Simple), fills missing values using the *mean value* for numerical attributes and the *most-frequent value* for nominal attributes. *Expectation-Maximization Imputation* (EMI) [33, 61, 70] is an iterative method with two steps. Expectation (E), where values are imputed based on observed values. And Maximization (M), where imputed values are evaluated and updated if necessary according to the data distribution. The EM algorithm converges to imputed values consistent with the observed distribution. *k-Nearest Neighbor Imputation* (kNNI) [9] uses the neighborhood of a missing value to estimate the corresponding imputation value. Defining the optimal $k$ value is challenging and has important implications on performance at the cost of computational burden [36, 80]. In our tests we use $k = 3$, as a compromise given the range of data set sizes. Although parameter tuning can improve performance of predictive models, it would increase the complexity of CIM. In our tests, we set the classifiers/regressors in CIM-LR and CIM-RF to default values, using 100 trees for random forest. In total, we generate $80 \times 6 = 480$ imputed versions of each data set.

3. **Use imputed data for classification.** We train classification models using the imputed data and compare the performance of these models against the baseline performance of models generated using complete data. In order to control the complexity of our tests we use logistic regression and random forest as final classifiers. Our focus is to measure the impact of imputed data, thus, we use default parameters for each classifier, again using 100 trees for random forest. We use 10-fold

cross validation for each classification test, and perform in total $(480 \times 2 \times 10 = 9600)$ tests for each data set.

### 4.2.2 Scalability

To test the scalability of CIM, we focus on three large data sets, Adult, Covtype and Llog. We compare the two versions of CIM against EMI and kNNI. Simple and Constant are not included given their low complexity. For each complete data set $D$, we create subsets $D_i \in D$ corresponding to 5%, 10%, 25%, 50%, 75% and 100% of the complete data set sizes. Then, for each complete subset $D_i$ we generate 10 incomplete versions with 5%, 10%, 25% and 50% missing values ratios using MCAR and MAR. In total, we measure imputation time on $(6 \times 10 \times 4 \times 2 = 480)$ incomplete sets. Reported times are the average of imputing each incomplete sets for each combination of set size, missing values ratio and missingness mechanism.

### 4.2.3 Imputed vs Incomplete Data

Since some algorithms handle missing data internally, we are interested in comparing performance of models trained on incomplete data vs models trained on imputed data. eXtreme Gradient Boosting (XGB) [28] handles missing values by defining a default split direction on each tree node. An instance is classified into the default direction if the attribute value needed for the split is missing. We generate 3 classification models using XGB, one model is trained on incomplete data and the other two models are trained on imputed data from CIM-LR and CIM-RF. Again, we focus on the Adult, Covtype and Llog data sets. For each complete set we generate 10 incomplete versions with 5%, 10%, 25% and 50% missing values ratios using MCAR and MAR. Final classification task is evaluated using logistic regression and random forest with 10-fold cross validation. In total we perform $(4 \times 10 \times 2 \times 3 \times 10 = 2400)$ tests for each data set.

### 4.2.4 Multi-label Imputation

We compare the impact on classification when using single-label imputation vs multi-label imputation, in other words CIM vs Multi-CIM/eMulti-CIM. Preliminary tests confirm that the distribution of missing values in MCAR data have a considerable impact on the number of cascade passes in Multi-CIM and eMulti-CIM. This is because, differently to MAR, missing values are equally distributed across the data, reducing the chances of getting large regions of missing values to impute. In consequence, Multi-CIM performs more passes increasing considerably the imputation time, undermining the potential of multi-

label imputation on MCAR data. For testing multi-label imputation, we focus on MAR data on the representative data sets, namely Adult, Covtype and Llog.

First, we compare CIM against MULTI-CIM. To provide a fair comparison we use the same base learners the internal learning tasks in both techniques. We use as base classifier/regressor either logistic regression/linear regression (CIM-LR, MULTI-CIM-LR) or random forest (MULTI-CIM-RF, MULTI-CIM-RF). We test using three different number of labels (targets) to impute simultaneously, $l = \{3, 5, 7\}$. 4 ratios of missing values 5%, 10%, 25% and 50% are used and 10 incomplete versions are generated for each pair of data set-ratio. The final classification task is performed using logistic regression and random forest with 10-fold cross validation. In total we perform $(10 \times 4 \times 3 \times 10 = 1200)$ tests for each data set.

Second, we test CIM against EMULTI-CIM. The number of labels to impute is set to $l = 7$ and two ensemble sizes are tested $m = \{10, 30\}$. Since we are creating an ensemble, we opt to use single models as base learners for the HC, in our tests we use decision trees. We compare CIM paired with random forest (CIM-RF) against EMULTI-CIM with decision trees as base learner for the chains (EMULTI-CIM-DT). The same ratios of missing values are used. For the final classification task, we use logistic regression and random forest with 10-fold cross validation. In this case, $(10 \times 4 \times 2 \times 10 = 600)$ tests are performed per data set.

### 4.2.5 Measuring Performance

An appropriate metric is vital to compare predictive models. Simple metrics such as accuracy can be misleading when classes (labels) are unbalanced [59], a trait of real-world data. For example, in binary classification, if a data set has only 4% of positive instances and we train a model that classifies all (100%) test instances as negative. Although the classifier fails for all positive instances its accuracy is 96%, which is misleading. The data sets in our tests, Table 4.1, present different degrees of class imbalance, being particularly high in multi-label data sets, since we only consider one class for binary classification the rest of classes-labels are grouped into a larger (negative) class.

We use two metrics that account for the actual amount of correctly classified instances, namely: Area Under the Receiver Operating Characteristic Curve (AUROC) [38] to evaluate performance of binary classifiers and F1-Score for multi-class classifiers. The ROC curve is a popular tool used to measure the performance of binary classifiers. It represents the trade-off between true positives and false positives as the positive threshold ($t$) changes. The ROC Space is defined by the false positive rate ($fpr$) and the true positive rate ($tpr$). The Area

Under the Curve (AUC) reduces the interpretation of the ROC curve to a single metric.

$$\text{AUROC} = \int_{-\infty}^{\infty} tpr(t) fpr(t) dt \tag{4.7}$$

$$tpr = \frac{tp}{tp + fn} \qquad fpr = \frac{fp}{fp + tn} \tag{4.8}$$

The perfect classifier will have AUROC $= 1.0$, while a random classifier is the one with AUROC $= 0.5$. Any value lower than 0.5 means a classifier performs worse than a random guess.

Similarly, F1-Score is a popular metric to measure the performance of binary classifiers and can be extended to the multi-class case. F1-Score measures performance as a trade-off between the ratio of true positives against all actual positives (precision) vs the ratio of true positives against all predicted positives (recall).

$$\text{F1-Score} = 2\frac{pr}{p + r} \tag{4.9}$$

$$precision = p = \frac{tp}{tp + fp} \qquad recall = r = \frac{tp}{tp + fn} \tag{4.10}$$

where $tp$, $fp$, $fn$, $tn$ correspond to the number of True Positives, False Positives, False Negatives and True Negatives, respectively.

Given the multiple configurations in our test setup, we define an *Overall Performance Ranking* to simplify the interpretation of results. Since we are interested in robust methods, we focus on the general behavior of imputation methods over multiple configurations rather than on isolated cases. We use the Root Mean Square Error (RMSE) to measure the performance difference between a classifier trained on complete data $(z_{base})$, and the same classifier trained on imputed data $(z_i)$, where $i$ corresponds to $n$ missing values ratios. The RMSE is calculated as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(z_{base} - z_i)^2} \tag{4.11}$$

Then, we rank each imputation method based on its RMSE within a test set. One test set contains results grouped by [data-set, supervised learning algorithm, missing data mechanism] for $n$ missing values ratios. We award points based on the following criteria:

3: *Top*: If the method is the best performer and there are no *ties*.

**2**: *Tie*: If multiple methods fall in a 5% band from the *top* performance.

**1**: *Runner-up*: Methods bellow 5% from the *top*.

**0**: If the imputation method *fails* for all the files in the test set.

**0**: If the method is the last among non-*fail* methods and the difference with the best *runner-up* is larger than 1x the difference between *top* and best *runner-up*.

## 4.3 RESULTS

Consider the *operating range* of an imputation method as the conditions under which imputation is successful. Such conditions include the characteristics of the data (number of instances, dimensionality, type –nominal or numerical–, etc.), as well as the nature and amount of the missing values. Interestingly, we observe that the operating range of EMI and kNNI is rather small, and they fail under multiple test configurations.

Table 4.2 shows the number of successful imputations over 40 incomplete data sets for each mechanism. kNNI fails when the number of values in the data set is large (Adult, Census-IKDD, Covtype, Enron, Genbase, Llog, Medical, Scene and Yeast), regardless of the missing data mechanism or missing values ratio. On the contrary, when the

**Table 4.2**: Number of successful imputations (out of 40) for each test set.

| | MAR | | | | | | MCAR | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Data set** | CIM-LR | CIM-RF | Constant | EMI | kNNI | Simple | CIM-LR | CIM-RF | Constant | EMI | kNNI | Simple |
| Adult | 40 | 40 | 40 | 40 | 0 | 40 | 40 | 40 | 40 | 31 | 0 | 40 |
| Census-IKDD | 40 | 40 | 40 | 40 | 0 | 40 | 40 | 40 | 40 | 31 | 0 | 40 |
| Music | 40 | 40 | 40 | 11 | 40 | 40 | 40 | 40 | 40 | 10 | 40 | 40 |
| Enron | 40 | 40 | 40 | 0 | 0 | 40 | 40 | 40 | 40 | 0 | 0 | 40 |
| Genbase | 40 | 40 | 40 | 0 | 0 | 40 | 40 | 40 | 40 | 0 | 0 | 40 |
| Llog | 40 | 40 | 40 | 0 | 0 | 40 | 40 | 40 | 40 | 0 | 0 | 40 |
| Medical | 40 | 40 | 40 | 0 | 0 | 40 | 40 | 40 | 40 | 0 | 0 | 40 |
| Scene | 40 | 40 | 40 | 15 | 0 | 40 | 40 | 40 | 40 | 0 | 0 | 40 |
| Yeast | 40 | 40 | 40 | 36 | 40 | 40 | 40 | 40 | 40 | 10 | 40 | 40 |
| Covtype | 40 | 40 | 40 | 0 | 0 | 40 | 40 | 40 | 40 | 0 | 0 | 40 |
| *Total* | 400 | 400 | 400 | **142** | **80** | 400 | 400 | 400 | 400 | **82** | **80** | 400 |

number of values is small (Music,Yeast), kNNI performance is in the top-tier. On the other hand, EMI is sensitive to high dimensionality (Enron, Genbase, Llog, Medical), and success ratio drops as the missing values ratio increases (Adult, Census-IKDD, Music, Scene, Yeast), especially on MCAR data. This indicates that EMI is sensitive to data size, missingness mechanism and ratio of missing values. In contrast, CIM-LR, CIM-RF, Constant and Simple methods successfully impute data for all test configurations. However, Constant's performance is mostly inconsistent with worst performance on small data sets and on MCAR. Simple performs remarkably well through our tests. It is interesting that such simplistic solution performs in overall better than EMI and kNNI across multiple test configurations. Finally, the two versions of CIM show the best overall performance, being CIM-RF the top performer.

The ranking of overall performance across multiple test configurations is available in Table 4.3. Top performer is CIM-RF, followed by CIM-LR and Simple. Notice that optimal performance is achieved by CIM without using parameter tuning for the internal classification/regression tasks. Although Constant imputes data successfully for all tests, its overall performance is low. kNNI is next as it shows good performance with small to medium size data sets but fails on large data sets. The worst performer in our tests is EMI given its narrow operating range. However, it is important to remark its good performance when the missing values ratio and the data set size are small. The Scene data set is an odd case since EMI provides above the baseline performance for 5%, 10% and 50% missing values. Moreover, reservation is required since not all files are successfully imputed (only one file with 50% missing values is successfully imputed). We also see a boost in performance above the baseline in other cases, predominantly on those where the missing values ratio is small (Music, Llog, Scene). Detailed information for these results is available in Appendix 4.

**Table 4.3:** Overall performance ranking over multiple test configurations. Larger values are better. Top performer is CIM-RF, followed by CIM-LR and Simple.

| **Classifier** | **Mechanism** | CIM-LR | CIM-RF | Constant | EMI | kNNI | Simple |
|---|---|---|---|---|---|---|---|
| Logistic Regression | MAR | 13 | **17** | 14 | 1 | 2 | 12 |
| | MCAR | 15 | **21** | 5 | 0 | 4 | 10 |
| Random Forest | MAR | 16 | 11 | 13 | 1 | 2 | **18** |
| | MCAR | 13 | **18** | 7 | 0 | 3 | 16 |
| | *Total* | 57 | **67** | 39 | 2 | 11 | 56 |

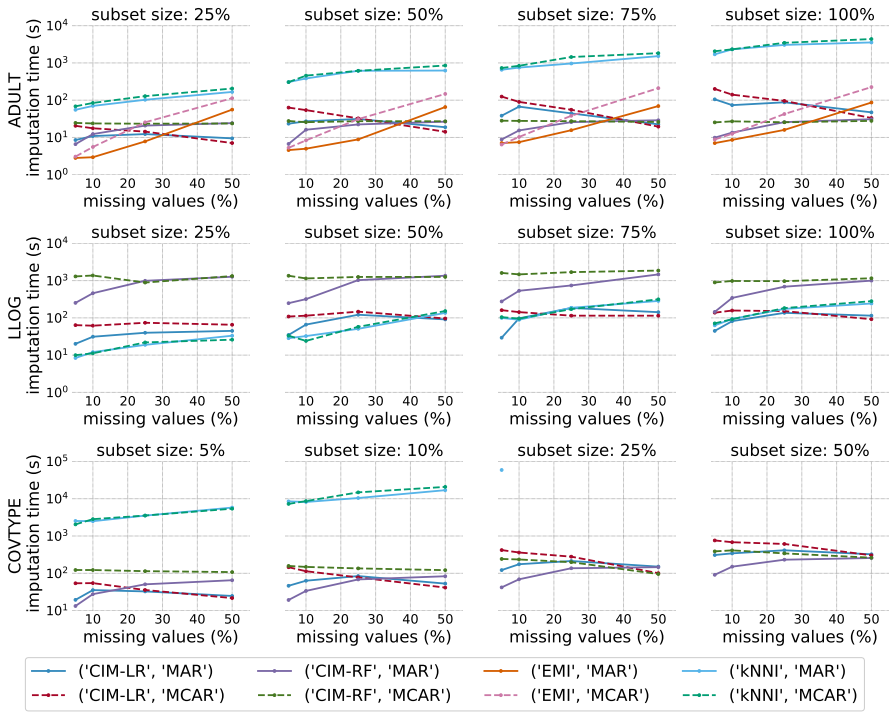
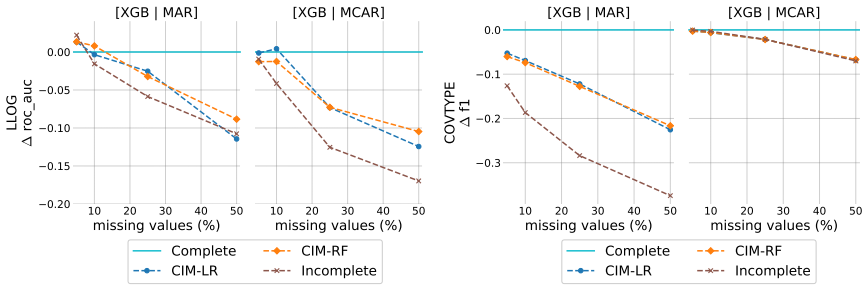



**Figure 4.4:** Scalability test results for CIM, EMI and kNNI. CIM imputation time decreases with larger missing values ratios, while EMI and kNNI are weak against large data set sizes and ratio of missing values.

The impact of the supervised learning algorithm on performance is also important to consider. In Table 4.3 we see that logistic regression and CIM-RF are a good combination for both MAR and MCAR. Results show that random forest does a better job at exploiting imputed data from CIM-LR and Simple. In particular, we see a significant boost in performance when Simple is paired with random forest. This explains the close gap between CIM-LR and Simple in the overall performance ranking. Covtype with more than 31M values provides insight on the suitability of sophisticated imputation methods for big data sets. The top performer in this case is CIM-LR, followed closely by Simple and CIM-RF. This suggests that Simple represents a good compromise for *extremely* large data sets, given the computational burden of sophisticated imputation methods.

*Scalability* test results are shown in Figure 4.4 for Adult, Llog and Covtype. We observe that CIM takes longer to impute MCAR data. This is expected given that missing values are equally distributed across the data attributes, requiring more iterations of the cascade. CIM is faster on MAR data with low missing values ratio, but as the ratio increases ($\geq 25\%$) there are more attributes to process and imputation time gets closer to the time for MCAR. Notice that for kNNI and EMI, imputation time increases along with the missing values ratio, while the contrary happens for CIM-LR and CIM-RF.




(a) (b)

**Figure 4.5:** Performance comparison between classification models trained using incomplete vs imputed data. Models are created using XGB which handles missing values in the train data.

This is explained by the fact that there is less training data as the number of missing values increases, which results in less training time for the classification/regression models within CIM. In Adult, kNNI is the slowest imputation method. On the other hand, Llog presents the worst case scenario for CIM given its high dimensionality, especially with MCAR data. Nonetheless, notice that the gap between CIM and kNNI decreases as data size and missing values ratio increase. For Covtype, kNNI is again the slowest to impute data and starts to fail at 25% of the original data set size, while CIM-LR and CIM-RF successfully impute data for all subset sizes. EMI fails to impute data for all subsets of Llog and Covtype.

In Figure 4.5 we show the results of training a classification model (XGB) using *incomplete data vs imputed data*. For Llog, we see that the classifier trained on incomplete data trails behind as the ratio of missing values increases. This is more noticeable on MCAR data. Covtype provides a large number of instances to learn from, still we see a considerable performance gap on MAR data. On the contrary, the 3 classifiers for MCAR data behave similarly. In Adult, we find similar performance for both MAR and MCAR data with a $\Delta$AUROC $\leq 0.02$. We conclude that, although there are some instances (Adult) where training on incomplete data can provide good results, this is not the general case and automatic imputation methods shall be considered, especially if the missing values ratio is above 5%.

Finally, single-label vs multi-label test results are shown in Figure 4.6 for Multi-CIM and in Figure 4.7 for eMulti-CIM. We omit plots for the Adult given that the difference in performance is marginal for both Multi-CIM and eMulti-CIM. Considering the overhead from multiple cascade passes we find that CIM represents the best compromise between learning performance and imputation time. In the case of Multi-CIM, Figure 4.6, we observe gains in Llog (a high dimensional data set) compared with CIM when using multi-label imputation, specially when the ratio of missing values is $\leq$ 25%. However, the combination of CIM-RF with logistic regression, is particularly hard to

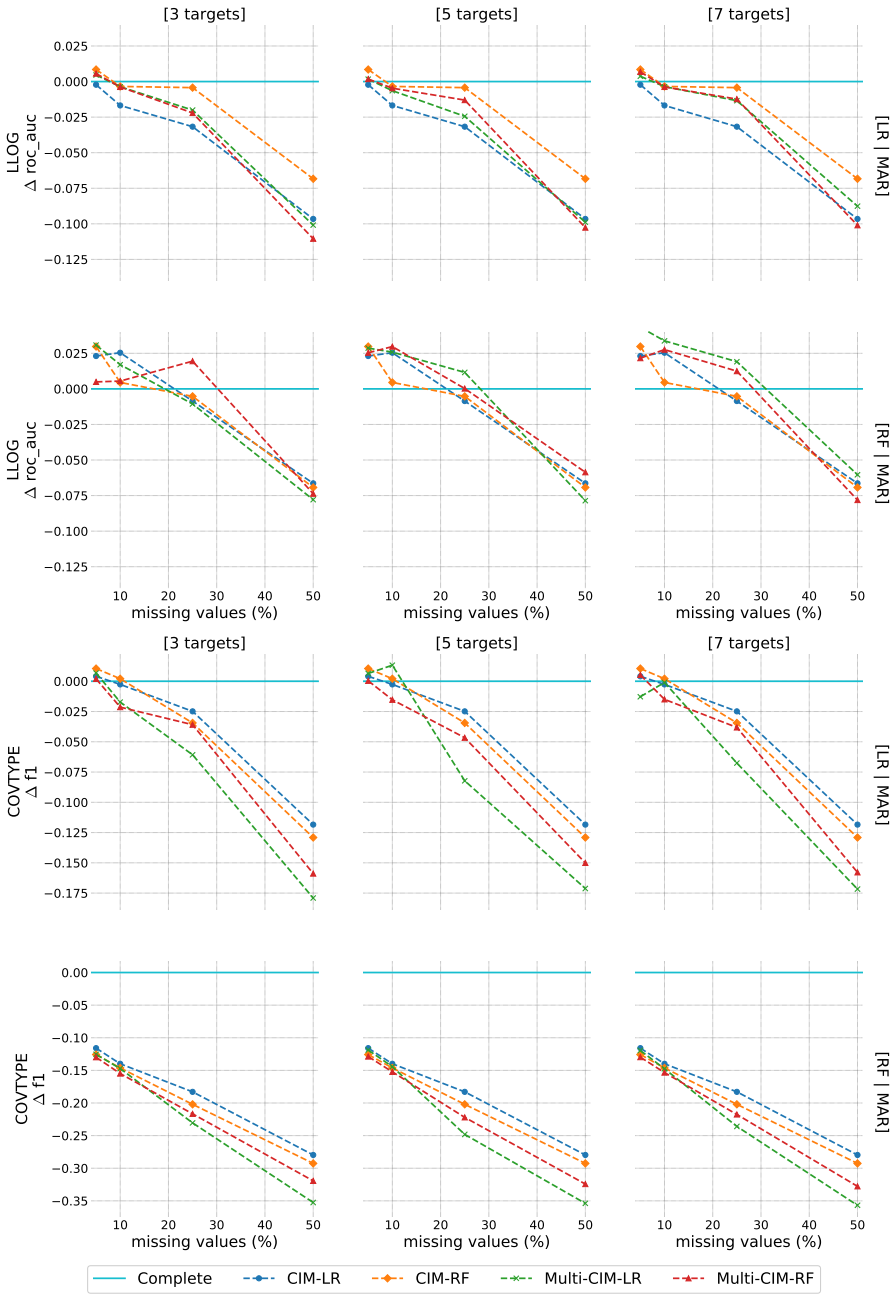
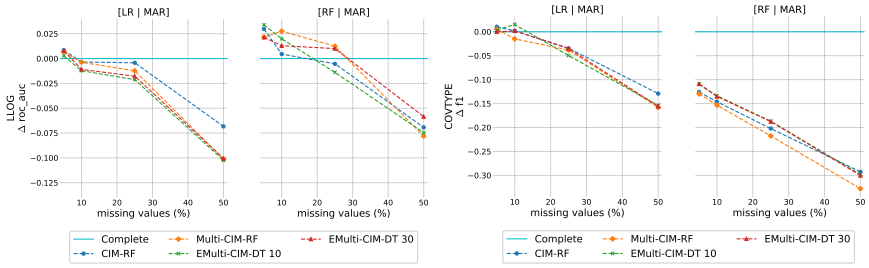



**Figure 4.6:** CIM vs Multi-CIM test. Three number of labels (3, 5, 7) are used for Multi-CIM. While we observe performance gains in high dimensional data (Llog), the impact of the final learner shows that combining CIM-RF with logistic regression is a good compromise. Notice that larger number of labels (7) can improve performance on small-medium missing values ratios.

outperform for Multi-CIM. This is consistent with our previous observations regarding the impact of the final learner on the interpretation of imputed data. In Covtype, Multi-CIM consistently under-performs with respect to CIM. In Figure 4.7, we compare eMulti-CIM using decision trees and two ensemble sizes $m = \{10, 30\}$ against CIM-RF




(a) (b)

**Figure 4.7:** eMulti-CIM test, setting to 7 the number of labels to impute at a time for Multi-CIM and eMulti-CIM and using two ensemble sizes (10, 30) for eMulti-CIM.

and Multi-CIM-RF. In Llog we observe improvement over Multi-CIM when $m = 30$ but still fails to outperform the combination of CIM-RF and logistic regression. On the other hand, in Covertype, we observe that the ensemble approach in eMulti-CIM reduces the gap observed between CIM and Multi-CIM, without outperforming CIM.

As previously discussed, a major source of unreliability in Multi-CIM and eMulti-CIM is the distribution of missing values which leads to the introduction of missing values during the cascade. Considering the rather unsatisfactory results we believe that further research is need, such as optimizing the learning tasks creation to maximize regions with missing values. Although multi-label imputation results are below expectations, we note two interesting outcomes: First, we confirm the viability of using HC on heterogeneous data types. Second, these results indicate that CIM is a simple, yet scalable and efficient imputation method.

Test results show that CIM performs well on a variety of conditions, expanding beyond the operating range of EMI and kNNI. An additional consideration is the straightforward implementation of an *imputation+classification* pipeline using CIM, given that it does not require extra tools to the ones used for the classification problem.

# 5 LEARNING FAST AND SLOW

A common premise (*communis opinio*) in machine learning is that batch and stream learning are mutually exclusive. A typical start point when tackling machine learning problems, is to categorize them as either batch or stream, and from that point most studies treat them separately. This separation often arises from specific requirements and expectations from both types of learning.

In batch learning, constraints on resources such as time and memory are somewhat relaxed (within acceptable limits) in favor of complex models that generalize well the data that describes a phenomenon. For example, neural networks (NN) and tree ensembles, such as random forest (RF) and extreme gradient boosting (XGB), are popular techniques in classification that require large amounts of data for training. Additionally, as the amount and complexity of the data increases, these techniques are known to be computationally demanding [91]. For example, in [73], a large deep neural network is used to learn high-level features. Training time on 10 million images is 3 days for a 9-layer locally connected sparse autoencoder, using 1000 machines (16000 cores).

On the other hand, requirements for stream learning [15] are based on the assumption of a continuous flow of data. In this context, storing data is impractical, thus models are incrementally trained, data is processed as it arrives (one at a time) and never re-used nor stored. Streaming techniques prioritize the optimization of two resources: time and storage. Since streams are assumed to be potentially infinite, stream models are required to be *always available*, this means that they must provide predictions at any time.

Different to the batch setting, where data distribution is assumed static, in data streams the relationship between input data and the corresponding response (label) may change over time, this is known as *Concept Drift*. Without intervention, batch learners will fail after drift occurs, since they were essentially trained on data that does not correspond to the current concept. A common intervention is to train a new batch model introducing the new concepts. Stream techniques, on the other hand, adapt to new concepts as the model is updated while some techniques even include special mechanisms designed to handle drifts.

In this chapter, we consider that learning is a *continuous activity* and aim to show that there exists an overlapping region where batch and stream learning can positively interact, benefiting from their strengths while compensating their weaknesses. Stream learners are

easier and cheaper to maintain, they adapt (react) to changes in the data distribution and provide predictions based on the most recent data. Batch learners require time to collect sufficient data before they can build a model, but once available, they typically generate more complex and accurate models.

While storing a complete data stream is impractical, current trends in (cheap) data storage [27, 111] provide the means to store large subsets of the stream. In fact, big data sets for batch learning are usually generated by collecting data over time. In this scenario, where data is continuously arriving, we propose a framework composed of stream and batch learners.

This unified approach is similar to the model proposed by Nobel Prize in Economics laureate Daniel Kahneman in his best-selling book *Thinking, Fast and Slow* [67] to describe the mechanisms behind human decision-making. The central thesis of this book is a dichotomy between two modes of thought: *System 1* is fast, instinctive and emotional; *System 2* is slower, more deliberative, and more logical. Equivalent systems for Machine Learning, are outlined in Table 5.1.

**Table 5.1:** The Fast and Slow systems for machine learning.

| FAST SYSTEM | SLOW SYSTEM |
|---|---|
| Cheap (mem., time) | Expensive (mem., time) |
| Always ready | Trains on large batches |
| Robust to drifts, adapts | Complex and robust models |
| Focus on the present | Generalizes the larger scheme |

The rest of the chapter is organized as follows: Section 5.1 gives a brief overview of the working memory model and its application in machine learning. Section 5.2 describes the FAST AND SLOW LEARNING framework and its application on classification. Section 5.3 outlines the experimental evaluation. In Section 5.4 we discuss our test results.

## 5.1 FROM PSYCHOLOGY TO MACHINE LEARNING

In Psychology, the Atkinson-Shiffrin model [6] describes the human memory as composed of three elements: *Sensory Memory*, *Long-Term memory* and *Short-Term memory*. Contrary to the Atkinson-Shiffrin model that describes the Short Term Memory as a static store, the working memory model proposed by Baddeley and Hitch in the seminal work [7], describes the Short Term Memory as dynamic process. According to this working memory model, a Central Executive controls the actions of the other components and serves as a supervisory system that becomes involved when cognitive processes go astray.

The *Long-Term — Short-Term* memory model (LTM-STM) has been used in the field of machine learning in an attempt to replicate the way in which the human brain works. This model is found in the *Long Short Term Memory* networks (LSTM) [60], a well known type of recurrent neural network which is known to perform, for many tasks, better than the standard version. A couple of drift-aware ensembles, also work under the LTM-STM model. The *Two-Layer System* presented in [47] is composed of a *learning layer* (Layer 1), where a model is learned for the original problem, and a *control layer* (Layer 2) that monitors the learning process of the first layer. In the second layer, a drift detection mechanism is used to identify regions of stable concepts which are stored in a pool of long-memory models. Similarly, the *Self Adjusting Memory* (SAM) model [79] builds an ensemble with models targeting current or former concepts. SAM is built using two memories: STM for the current concept, and the LTM to retain information about past concepts. A cleaning process is in charge of controlling the size of the STM while keeping the information in the LTM consistent with the information in the STM.

## 5.2 FAST AND SLOW LEARNING FRAMEWORK

In this chapter, we introduce the Fast and Slow Learning (FSL) framework. Similarly to [47] and [79], we approach learning as a combination of two systems: Fast and Slow. Moreover, we consider learning as a continuous task where batch and stream learning co-exist. The Fast system, consists of a stream model $M_f$ which is continuously updated over time. The Slow system consists of a sequence of batch models $\{M_{s_1}, M_{s_2}, \ldots, M_{s_n}\}$ that are replaced as they become obsolete and a data buffer that stores most recent data for training. Under this schema, Fast and Slow learners complement each other: The Fast learner, incrementally trained as data becomes available, provides predictions consistent with the current concept and adapts quickly to concept drift. Whereas the Slow learner first collects a large amount of data to generate complex models, which are potentially superior in the presence of stable concepts. An overview of the FSL framework is available in Figure 5.1.

### 5.2.1 Fast and Slow Classifier

In the following, we describe an application of the FSL framework for classification, the Fast and Slow Classifier (FSC). In the FSC, the internal models in the Fast and Slow systems correspond to classification models. Consider a continuous stream of data $A = \{(\vec{x}_i, y_i)\} | i = 1, \ldots, t$ where $i$ represents the current time and $t \to \infty$. $\vec{x}_i$ is a feature vector and $y_i$ the corresponding target class such that

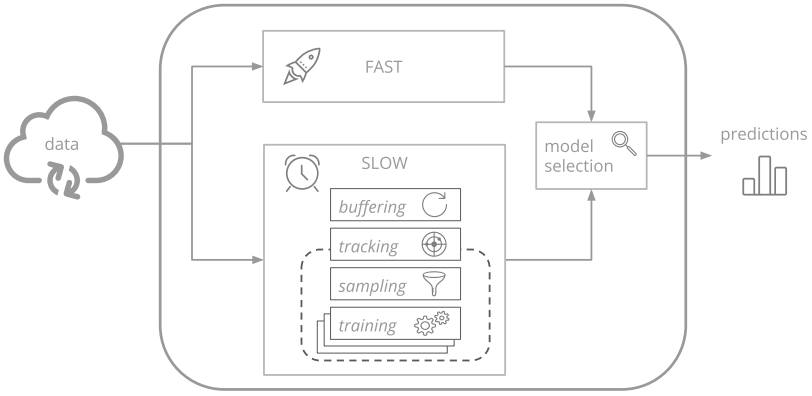



**Figure 5.1:** Fast and Slow Learning framework overview.

$y_i \in \{C_1, C_2, \ldots, C_k\}$. A Fast (stream) model $M_f$ is trained on single instances as they arrive. A Slow (batch) model $M_s$ is trained using the data buffer $B = \{(\vec{x}_i, y_i)\} | i = b - n + 1, \ldots, b\}$ such that $B$ contains the last $n$ samples in $A$ with respect to a given point in time $b$.

$$\text{STREAM} : train_f(M_f, (\vec{x}_i, y_i)) \tag{5.1}$$

$$\text{BATCH} : train_s(M_s, B) \tag{5.2}$$

The objective is to accurately predict the class of the current feature $\vec{x}_t$, as its label is presumed to be initially unknown. The predicted label of a model $M$ is denoted as $\hat{y}$. As soon as the true label $y$ gets revealed the performance $P$ is measured according to some loss function $\ell$.

$$P(M) = \ell(y, \hat{y}) \tag{5.3}$$

Performance of stream models is measured using *prequential evaluation*, where an instance's class is predicted before using it for training. This is, for the $i$-th instance:

$$\hat{y}_i = M_{i-1}(\vec{x}_i) \tag{5.4}$$

$$train_f(M_{i-1}, (\vec{x}_i, y_i)) \tag{5.5}$$

FSC works in two modes, outlined in Figure 5.2, where Fast and Slow systems perform slightly different tasks:

- **Single mode**: Corresponds to the beginning of the stream, from $i = 0$ until $M_s$ is trained (provides predictions). In this mode, only the Fast system is active and FSC effectively performs as a stream classifier, whereas the Slow system stores incoming data into its buffer $B \leftarrow (\vec{x}_i, y_i)$, and triggers the training of $M_s$ when the buffer reaches its limit.

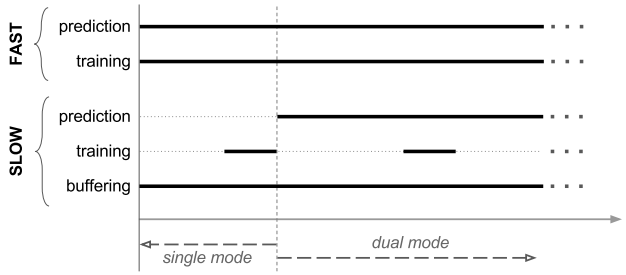



**Figure 5.2:** FSC operation. In *single-mode*, the FAST model is used for prediction while SLOW is buffering the data. In *dual-mode*, both models predict each sample but only one is selected based on their recent performance. SLOW models are replaced in presence of concept drift.

- **Dual mode**: Begins once training of $M_s$ is complete. From this point, FSC tracks the performance of $M_f$ and $M_s$ and selects the one with best (current) performance $P_t$ over most recent data. Data buffering in $B$ continues in a sliding window fashion, if concept drift occurs the data in the buffer is used to train new batch models. Upon entering into *dual mode*, FSC remains in that mode.

Notice that when we refer to $M_s$ as *slow*, we refer to the training time as well as the time required for data collection. Since data arrives on time intervals $\Delta t$, the time for the SLOW classifier to be ready to predict is:

$$idle\,time = (\Delta t \times n) + time(train(M_s)), \tag{5.6}$$

where $n$ is the number of samples. The worst case scenario is when data arrives at slow rates (large $\Delta t$) and training time is high, for example a complex deep neural network with hundreds of layers.

#### 5.2.1.1 *Training Batch learners on Data Streams*

If concept changes in the stream, $M_s$ becomes obsolete and has to be replaced by a new model trained on the current concept. We propose a strategy to track $M_s$ performance and trigger the training of new models automatically. FSC uses the ADWIN [13] change detector to track changes in the performance of $M_s$. ADWIN keeps one dynamic size window that grows when there is no change and shrinks otherwise. If the difference between the means of two possible adjacent sub-windows exceeds a delta depending on a predefined confidence parameter $\delta$ then a change is detected and the dynamic window reduces its size. As soon as ADWIN detects a change, the training of new models is triggered.

FSC uses the buffer $B$ to train and evaluate new models. $B$ is split into two adjacent sets $B_{train}$ and $B_{test}$ where $|B_{train}| \gg |B_{test}|$, keeping

the data arrival order so $B_{test}$ contains the last samples from the stream. In addition to training a model on $B_{train}$, FSC trains $K$ candidate models on different samplings of $B_{train}$, in order to improve the chances of training an optimal model. FSC applies three different sampling strategies:

- Probabilistic Adaptive Window (PAW) [19], keeps in logarithmic memory a sample of items $W$, giving more weight to newer items but maintaining some older items. Each new item $i$ is stored into $W$ then for each $j$ element in $W$, a random number $r$ in $(0,1)$ is obtained, then $j$ is removed from $W$ if $r > 2^{-1/|W|}$.
- Reservoir Sampling (RS) [118], samples $R$ random items from a stream (without replacement) and is equally biased towards old items and new items. For a sample size $R$ with $i$ items seen so far, the chance of any item to be chosen is $R/i$. For the next item, a chosen item has a probability to remain of $R/(i+1)$ and a new item has a probability of $R/(i+1)$ to be kept.
- Weighted Reservoir Sampling (WRS)[37]. Samples from a weighted distribution where each items has a corresponding weight in the stream. A *key* value $k_i$ is assigned to each item $i$ in the stream, by $k_i = u_i^{1/w_i}$ , where $w_i$ is the weight of the item and $u_i$ is a random number in $(0,1)$. Finally, the $R$ items with largest keys $k_i$ are kept. FSC assigns weights to instances in $B_{train}$ based on their arrival order, with larger weights given to newer instances.

Once all candidate models ($M_{s'_{1,\cdots,K+1}}$) are trained, FSC uses $B_{test}$ to compare their performance against $M_s$. FSC picks between the top candidate $M_{s'_{top}}$ and the current model $M_s$ based on their performance:

$$M_s = \begin{cases} M_s, & \text{if } P(M_s) > P(M_{s'_{top}}) \\ M_{s'_{top}}, & \text{otherwise} \end{cases} \tag{5.7}$$

#### 5.2.1.2 *Model selection*

During *dual mode* operation, FSC applies the prequential method to evaluate the performance $P_t$ of $M_f$ and $M_s$ at time $t$ to identify the current top performer. Then, FSC yields predictions from the current top performer for the next $n$ instances. Similar to humans, FSC uses the FAST system by default. Thus predictions from FSC are given by:

$$\text{FSC}(\vec{x}) = \begin{cases} M_f(\vec{x}), & \text{if } P_t(M_f) \geq P_t(M_s) \\ M_s(\vec{x}), & \text{otherwise} \end{cases} \tag{5.8}$$

One way to measure performance is using sliding windows such that only last $n$ samples are taken into account. Let $y_W$ and $\hat{y}_W$ be the true and predicted classes in the window $W$, then the prequential error at time $t$ is

$$E_t(W) = \ell_t(y_W, \hat{y}_W) \tag{5.9}$$

A consequence of excluding older samples is that measurements can vary considerably between consecutive windows, which translates in more transitions between $M_f$ and $M_s$.

An alternative to sliding windows are *fading factors* [49], which are faster and memory-less. This method decreases the influence of older samples on the measurement as the stream progresses. Computing the loss function $\ell$ for each sample, the prequential error at time $t$ using fading factors is estimated by

$$\begin{aligned} E_t &= S_t / B_t \\ S_t &= \ell(y_t, \hat{y}_t) + (\alpha \times S_{t-1}) \\ B_t &= 1 + (\alpha \times B_{t-1}) \end{aligned} \tag{5.10}$$

Where $\alpha \in \mathbb{R} : 0 \ll \alpha \leq 1$ is the forgetting factor. In Section 5.3, we show how using fading factors effectively reduces the number of FAST-SLOW model transitions while keeping the performance of FSC optimal.

We propose two main variants of FSC based on the method used to measure performance. In the following, we refer as FSC to the variant that measures performance over sliding windows, while $\text{FSC}_{ff}$ uses fading factors.

A third option is to consider the model selection as an instance of an *infinitely repeated* game, where a player $j$ ($M_f$, $M_s$) receives a stage-game payoff $u_j^t$ period-by-period. In the *average discounted criterion* [46], the discount factor $\delta \in \mathbb{R} : 0 < \delta < 1$ indicates how patient is a player. The assumption is for each player to maximize a weighted sum of per-period payoffs, where later periods weight less than earlier periods. The average discounted value is calculated as

$$(1 - \delta) \sum_{t=0}^{\infty} \delta^t u_j^t \tag{5.11}$$

In our experiments, we apply discount to evaluation measurements using sliding windows by setting $u_j^t = E_j^t(W)$. This option is more conservative, resulting in a substantial drop in the number of model transitions. A possible application for this option is the case where one of the models is intended only as a fallback alternative, a kind of safety net for classification performance.

## 5.3 EXPERIMENTAL EVALUATION

We are interested in the task of classification in presence of concept drift. In order to thoroughly evaluate FSC, we select data sets from a variety of domains, for both binary and multi-class problems, see Table 5.2. We use 7 synthetic data sets and 5 real-worlds data sets. The synthetic data sets include different types of drift, including: abrupt (concept changes suddenly), gradual (concept changes slowly over a region where past and new concepts are present) and incremental-fast (there exist multiple intermediate concepts).

**AGRAWAL** Based on the Agrawal generator [3], represents a data stream with 6 nominal and 3 numerical features. Different functions map instances into two different classes. 3 abrupt drifts are simulated for $\text{AGR}_a$ and 3 gradual drifts for $\text{AGR}_g$.

**HYPER** A stream with fast incremental drifts where a $d$-dimensional hyperplane changes position and orientation. Obtained from a random hyperplane generator [64].

**MIXED DRIFT** A combination of 3 streams: Interchanging RBF, Moving Squares and Transient Chessboard. Contains incremental, abrupt and virtual drifts.

**SEA** A data stream with 3 numerical features where only 2 attributes are related to the target class. Created using the SEA generator [113]. 3 abrupt drifts are simulated for $\text{SEA}_a$ and 3 gradual drifts for $\text{SEA}_g$.

**Table 5.2:** Data sets. [Type] S: synthetic data; R: real world data. [Drifts] A: abrupt, G: gradual; $\text{I}_f$: incremental fast.

| | # instances | # features | # classes | Type | Drift |
|---|---|---|---|---|---|
| $\text{AGR}_a$ | 1000000 | 9 | 2 | S | A |
| $\text{AGR}_g$ | 1000000 | 9 | 2 | S | G |
| $\text{HYPER}_f$ | 1000000 | 10 | 2 | S | $\text{I}_f$ |
| $\text{MDRIFT}_a$ | 600000 | 2 | 10 | S | A |
| $\text{SEA}_a$ | 1000000 | 3 | 2 | S | A |
| $\text{SEA}_g$ | 1000000 | 3 | 2 | S | G |
| $\text{TCHESS}_a$ | 200000 | 2 | 8 | S | A |
| AIRL | 539383 | 7 | 2 | R | - |
| COVTYPE | 581012 | 54 | 7 | R | - |
| ELEC | 45312 | 6 | 2 | R | - |
| POKER | 829201 | 10 | 10 | R | - |
| WEATHER | 18159 | 8 | 2 | R | - |

**TRANSIENT CHESSBOARD** A stream with abrupt drifts designed to penalize algorithms that discard older concepts. Data is generated by randomly selecting squares in a chessboard.

**AIRLINES** Real-world data containing information from commercial flights scheduled departures within the US. The objective is to predict if a flight will be delayed.

**ELECTRICITY** Data from the Australian New South Wales electricity market where prices are not fixed but change based on offer and demand. The 2 target classes represent changes in the price (up or down).

**COVER TYPE** This data set contains data collected over time by the US Forest Service. Classes correspond to cover type in forest on squares of $30 \times 30$ meters.

**POKER** Randomly drawn poker hands [13]. Drift is introduced by sorting hands by rank and suit.

**WEATHER** Contains weather information collected between 1949 and 1999 in Bellevue, Nebraska. The objective is to predict if it will rain or not on a given date.

For further details, we refer the reader to [54] for AGRAWAL, HYPER and SEA, and to [79] for MIXED DRIFT and TRANSIENT CHESSBOARD.

We configure FSC using Hoeffding Trees (HT) [35] as $M_f$ and eXtreme Gradient Boosting (XGB) [28] as $M_s$. In this section we use the labels FAST and SLOW to refer to $M_f$ and $M_s$ respectively. Using this configuration, we aim to show that FSC is an effective collaboration strategy for learning from evolving data streams. Given that neither HT nor XGB have an explicit mechanism to handle drifts, robustness in performance on drift zones can be reasonably attributed to the drift detection mechanism in FSC which triggers the creation of new SLOW models. In our tests, we assume the training time for SLOW models to be smaller than the time $\Delta t$ between incoming samples, see Eq. 5.6. This means that $M_s$ is ready to predict from instance $n+1$. The size of the buffer $B$ is set $n = 10000$ for all data sets except for the smaller ones (ELEC and WEATHER) where $n = 5000$. The size of the window $W$ to compare SLOW models is set to 200 samples, the same number of samples used to periodically compare FAST vs SLOW performance and perform model selection. For sampling the data in $B_{train}$, we set the window size in PAW equal to $|B_{train}|$ (remember that the actual sample size is dynamic) where $|B_{train}| = |B| - |B_{test}| = 9800$. Two different sample sizes, corresponding to 50% and 75% of $B_{train}$ are used

for reservoir sampling ($RS_{50}$, $RS_{75}$) and weighted reservoir sampling ($WRS_{50}$, $WRS_{75}$), Finally, we set the confidence $\delta = 0.002$ for ADWIN and the forgetting factor $\alpha = 0.999$ for fading factors. We perform two main tests:

1. Compare FSC against the performance of FAST and SLOW over the stream, this is, measure the impact in performance from the collaboration between models. During *single mode* operation, only the FAST model is active, thus FSC defaults to this model for predictions and performance is the same. Once SLOW is trained, FSC enters into *dual mode* and further model trainings are triggered by the drift detector. It is important to notice that FAST and FSC cover the entire stream while SLOW only covers part of it. We compare performance for the models during both modes.

2. Compare FSC and $FSC_{ff}$ against other learning methods: Passive Aggressive Classifier (PAC) [31], an incremental learning method that modifies its prediction mechanism in response to feedback from current performance. The Perceptron Classifier (PRCP). Hoeffding Adaptive Trees (HAT) [14], which updates the tree structure in response to drift changes. And Self Adjusting Memory (SAM) [79], described in Section 5.1. It is important to note that HAT and SAM are methods designed to handle concept drift.

FSC is implemented using scikit-multiflow [89], a stream learning framework in Python (described in Chapter 7). Tests are performed using the implementations of PAC and PRCP in scikit-learn [95], the Python implementation of XGB and the implementations of HT, HAT and SAM from scikit-multiflow. All classification methods are set to default parameters. To measure performance, we use *accuracy* for binary classification and *kappa statistics* for multi-class classification.

## 5.4 RESULTS

An example of FAST and SLOW models working independently is shown in Figure 5.3. Test results for the FAST and SLOW models correspond to HT and XGB respectively. Plots in Fig. 5.4, 5.5 show model performance over time (*top*), the currently selected model indicating the source (FAST or SLOW) of predictions yield by FSC (*middle*) and the drift detection flag (*bottom*). Due to space limitation not all plots are shown.

Results for FSC are summarized in Table 5.3 and 5.4, divided depending on the operation mode (*single+dual*, *dual*) in which they where measured. Results indicate that FSC performs better than FAST for all

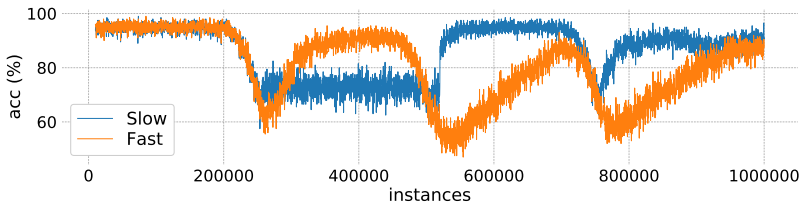



**Figure 5.3:** FAST (orange) and SLOW (blue) models sliding accuracy on $AGR_g$ which contains 3 gradual drifts at the 25%, 50% and 75% marks.

data sets across the entire stream (*single+dual*). On the other hand, FSC performs better than FAST and SLOW during *dual mode* in most cases. It should be noted that when we say that FSC performs better we refer to the optimal integration of predictions from FAST and SLOW models. We report performance in terms of *accuracy* and *kappa statistics* but is important to note that the main performance indicator used by FSC is *accuracy* for binary streams and *kappa statistics* for multi-class streams.

**Table 5.3:** Test results (mean accuracy) for FSC using sliding windows for performance tracking and model selection. Numbers in bold are top values. Values in *single+dual* are the average performance of FAST and FSC for the whole stream. Type indicates if the stream is B:Binary or M:Multi-class. Main performance indicator is *accuracy* for binary streams and *kappa* for multi-class.

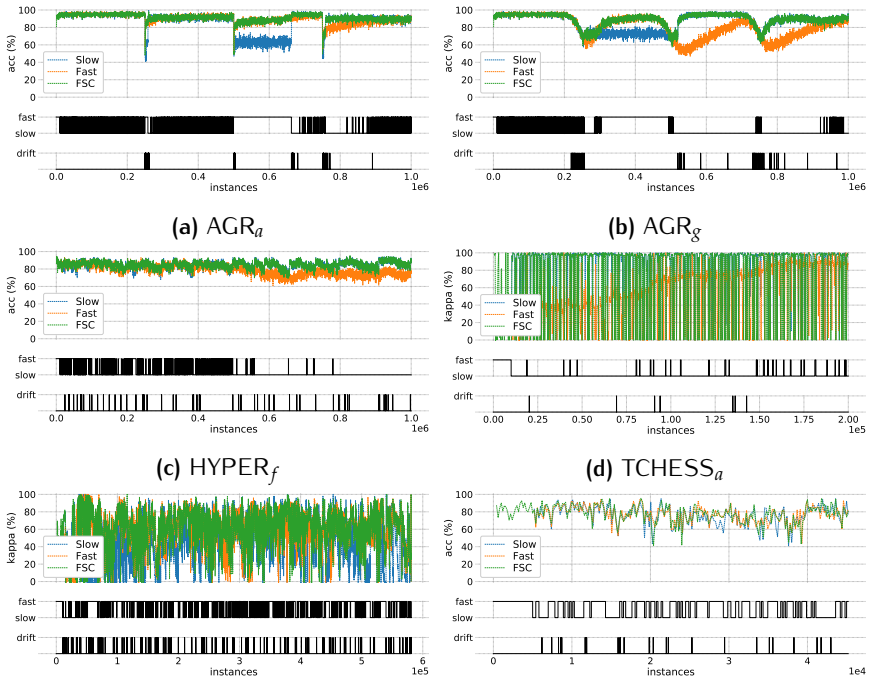
| | | **Accuracy** | | | | |
|---|---|---|---|---|---|---|
| | | *single+dual* | | *dual mode* | | |
| | Type | FAST | FSC | SLOW | FAST | FSC |
| $AGR_a$ | B | 90.07 | **91.23** | 86.97 | 90.04 | **91.22** |
| $AGR_g$ | B | 80.79 | **89.05** | 85.95 | 80.67 | **89.01** |
| $HYPER_f$ | B | 78.76 | **84.73** | 84.49 | 78.70 | **84.72** |
| $MDRIFT_a$ | M | 44.95 | **53.54** | 52.08 | 44.77 | **53.50** |
| $SEA_a$ | B | 86.43 | **88.41** | 88.37 | 86.43 | **88.43** |
| $SEA_g$ | B | 86.43 | **88.09** | 88.06 | 86.43 | **88.11** |
| $TCHESS_a$ | M | 68.89 | **95.66** | 98.35 | 69.76 | 97.96 |
| *synth avg* | | 76.62 | **84.39** | 83.47 | 76.69 | **84.71** |
| AIRL | B | 63.81 | **65.15** | 63.95 | 63.86 | **65.23** |
| COVTYPE | M | 82.33 | **83.23** | 77.75 | 82.73 | **83.64** |
| ELEC | B | 77.88 | **77.96** | 76.06 | 77.12 | **77.24** |
| POKER | M | 74.15 | **89.07** | 88.30 | 74.24 | **89.33** |
| WEATHER | B | 73.97 | **76.16** | **77.19** | 74.10 | 76.66 |
| *real avg* | | 74.43 | **78.31** | 76.65 | 74.41 | **78.42** |
| *overall avg* | | 75.71 | **81.86** | 80.63 | 75.74 | **82.09** |

**Table 5.4:** Test results (mean kappa) for FSC using sliding windows for performance tracking and model selection. Numbers in bold are top values. Values in *single+dual* are the average performance of FAST and FSC for the whole stream. Type indicates if the stream is B:Binary or M:Multi-class. Main performance indicator is *accuracy* for binary streams and *kappa* for multi-class.

| | | **Kappa** | | | | |
|---|---|---|---|---|---|---|
| | | *single+dual* | | *dual mode* | | |
| | Type | FAST | FSC | SLOW | FAST | FSC |
| $AGR_a$ | B | 78.62 | **81.01** | 73.86 | 78.57 | **80.99** |
| $AGR_g$ | B | 61.02 | **76.82** | 70.40 | 60.79 | **76.76** |
| $HYPER_f$ | B | 57.43 | **69.37** | 68.87 | 57.28 | **69.34** |
| $MDRIFT_a$ | M | 35.71 | **45.29** | 43.66 | 35.48 | **45.22** |
| $SEA_a$ | B | 71.10 | **75.23** | 75.14 | 71.11 | **75.29** |
| $SEA_g$ | B | 71.08 | **74.58** | 74.51 | 71.09 | **74.63** |
| $TCHESS_a$ | M | 50.11 | **78.75** | **84.46** | 50.89 | 81.07 |
| *synth avg* | | 60.72 | **71.58** | 70.13 | 60.74 | **71.90** |
| AIRL | B | 12.67 | **16.51** | **17.75** | 12.79 | 16.70 |
| COVTYPE | M | 59.19 | **60.70** | 47.35 | 59.47 | **61.00** |
| ELEC | B | 51.68 | **52.26** | 49.21 | 50.54 | **51.25** |
| POKER | M | 32.95 | **76.54** | **79.29** | 33.16 | 77.29 |
| WEATHER | B | 31.98 | **38.96** | **41.91** | 32.65 | 40.79 |
| *real avg* | | 37.69 | **48.99** | 47.10 | 37.72 | **49.41** |
| *overall avg* | | 51.13 | **62.17** | 60.53 | 51.15 | **62.53** |

In $AGR_a$ (Figure 5.4a) and $AGR_g$ (Figure 5.4b) we can distinguish the regions where drifts are simulated and their impact on the classifiers. In both cases, the SLOW system reacts faster than FAST due to the change detection mechanism that triggers the training of new batch models. In $AGR_a$, the SLOW system recovers quickly after the second drift but it is not able to train an optimal model for the concept and FSC chooses FAST. This is corrected later as a new drift is detected (around the 700K mark). In $AGR_g$, FAST takes longer to adapt since the concept drift is gradual. In this case, we see two clear regions after the second and third drifts where FSC opts for the SLOW model.

A more complex scenario is seen in COVTYPE (Figure 5.4e), where FSC constantly changes between FAST and SLOW with only small regions dominated by one model. Despite this, we notice that FSC yields predictions from the current top performer, resulting in an overall improvement. In ELEC (Figure 5.4f), model selection is easier to observe given the smaller size of the stream. In these cases we see




(a) AGR$_a$ (b) AGR$_g$

(c) HYPER$_f$ (d) TCHESS$_a$

(e) COVTYPE (f) ELEC

**Figure 5.4:** Current (sliding) performance over the last 200 instances for FSC (green) vs Fast (orange) vs Slow (blue). Below each plot are the flags corresponding to model selection and drift detection. *Accuracy* is shown for binary classification and *kappa statistics* for multi-class classification.

that FSC benefits from continuously picking predictions from both Fast and Slow models.

The results from HYPER$_f$ (Figure 5.4c) are interesting since they exemplify the case where one of the classifiers becomes superior. Fast performance drops around the middle of the stream and FSC defaults to Slow to provide predictions. This finding has important implications, it highlights the scenario where a batch model is a better predictor for a stream, but requires time to become optimal. The first half of the stream can be considered as a kind of *tuning* phase where FSC yields predictions from Fast and Slow. When an optimal Slow model is trained FSC defaults to it. We have successfully trained an optimal batch model for the stream while keeping the requirement of providing predictions at any time.

As previously indicated, TCHESS$_a$ (Figure 5.4d) is designed to handicap classifiers that favour new concepts over old ones. Abrupt and frequent drifts are challenging for Fast which starts with very low performance and takes most of the stream to perform better although still below Slow. On the other hand, Slow performs consistently better despite sudden (and very short) drops due to the nature of the stream. This results further strengthens the hypothesis that batch and stream

**Table 5.5:** Average ratio of SLOW model selection per sampling strategy. *Buffer* corresponds to using all instances in the buffer (no sampling).

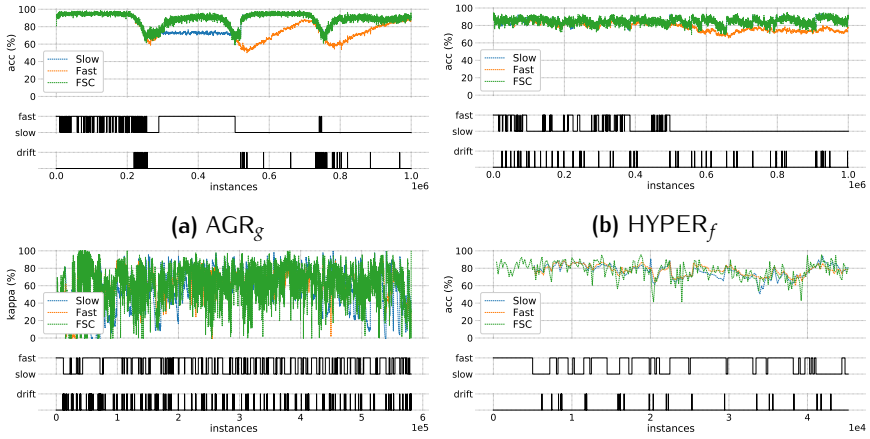
| | Buffer | PAW | $RS_{50}$ | $RS_{75}$ | $WRS_{50}$ | $WRS_{75}$ |
|---|---|---|---|---|---|---|
| $AGR_a$ | 0.00 | 4.00 | 14.00 | 6.00 | 34.00 | 42.00 |
| $AGR_g$ | 0.00 | 7.27 | 16.36 | 9.09 | 18.18 | 49.09 |
| $HYPER_f$ | 1.67 | 3.33 | 3.33 | 1.67 | 3.33 | 86.67 |
| $MDRIFT_a$ | 0.00 | 2.75 | 0.00 | 2.75 | 0.92 | 93.58 |
| $SEA_a$ | 6.67 | 20.00 | 6.67 | 6.67 | 20.00 | 40.00 |
| $SEA_g$ | 5.26 | 0.00 | 5.26 | 10.53 | 10.53 | 68.42 |
| $TCHESS_a$ | 0.00 | 12.50 | 0.00 | 25.00 | 0.00 | 62.50 |
| AIRL | 1.56 | 1.56 | 12.50 | 7.81 | 12.50 | 64.06 |
| COVTYPE | 0.75 | 1.50 | 2.26 | 3.01 | 3.76 | 88.72 |
| ELEC | 3.70 | 0.00 | 14.81 | 3.70 | 11.11 | 66.67 |
| POKER | 0.90 | 1.19 | 3.28 | 5.97 | 7.46 | 81.19 |
| WEATHER | 20.00 | 0.00 | 0.00 | 0.00 | 20.00 | 60.00 |
| *average* | 3.38 | 4.51 | 6.54 | 6.85 | 11.82 | **66.91** |

models can positively interact and shows how we can effectively apply a batch model into the stream learning setup.

It is important to notice that the proposed mechanisms in charge of creating new SLOW models is effective on most cases, enabling the positive interaction between FAST and SLOW. The average ratio of selected candidate SLOW models per sampling strategy is displayed in Table 5.5. We observe that $WRS_{75}$ is the sampling strategy that results in more optimal SLOW models, followed by $WRS_{50}$.

In Figure 5.5 we see that using fading factors to measure performance reduces the number of translations between FAST and SLOW models. Although the impact on overall performance is marginal with gains $< 1\%$ on $FSC_{ff}$ over FSC, the usage of fading factors represents a good compromise between performance, number of model transitions and resources. Due to space limitations we only show plots for selected data sets.

In $AGR_g$, Figure 5.5a, we see a significant drop in the number of model transitions in regions where one of the models starts showing better performance around the 300K and 550K marks. Another interesting example is $HYPER_f$, Figure 5.5b, as previously discussed, SLOW is the overall top performer for the second half of the stream. Using sliding windows we still see some model transitions whose impact on performance is presumably marginal, Figure 5.4c. These transitions are not observed (after the middle mark) using fading factors, in this case FSC yields predictions from SLOW for the rest of the stream. As expected, COVTYPE in Figure 5.5c remains as a complex case where both models interact continuously. However, we

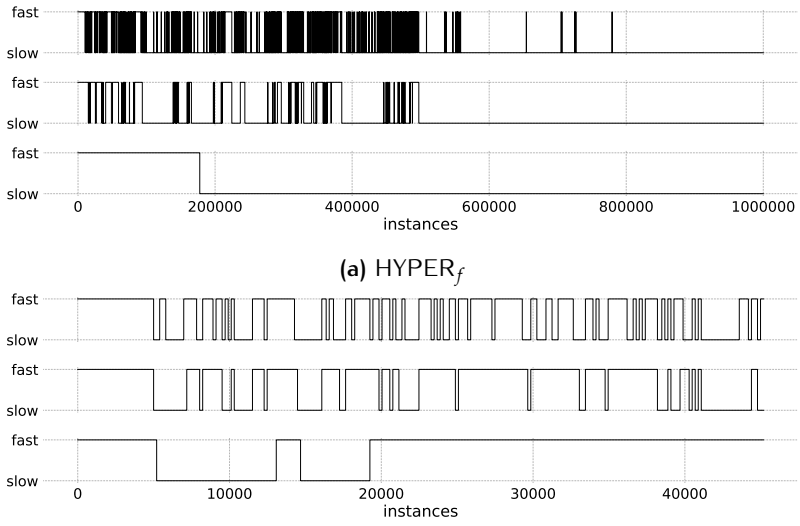



**(a)** AGR$_g$ **(b)** HYPER$_f$

**(c)** COVTYPE **(d)** ELEC

**Figure 5.5:** Current (sliding) performance of FSC$_{ff}$ with fading factors (green) vs Fast (orange) vs Slow (blue). Notice that using fading factors results in less transitions between models.

see an overall reduction in the number of model transitions without compromising performance. Finally, although in ELEC fading factors reduces performance, the difference is marginal. This can be attributed to the small size of the stream and the reduced number of model transitions. Consequently FSC sometimes misses the mark on the top performer for incoming samples. A possible solution for small streams is to reduce the evaluation window size (200 samples in our tests) used to track performance of both models.




**(a)** HYPER$_f$

**(b)** ELEC

**Figure 5.6:** Model selection based on different performance measurement criteria. *Top*: Sliding window. *Middle*: Fading factors. *Bottom*: Sliding windows with discount.

**Table 5.6:** Test results (mean accuracy) for FSC and $FSC_{ff}$ against stand alone stream methods. Numbers in bold are top values. Type indicates if the stream is B:Binary or M:Multi-class.

| | | Accuracy | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Type | FSC | $FSC_{ff}$ | PAC | PRCP | HT | HAT | SAM |
| $AGR_a$ | B | 91.23 | **91.35** | 54.83 | 55.55 | 90.07 | 80.67 | 68.62 |
| $AGR_g$ | B | 89.05 | **89.10** | 54.22 | 54.90 | 80.79 | 79.19 | 68.89 |
| $HYPER_f$ | B | 84.73 | 84.83 | 81.89 | 82.53 | 78.76 | 86.89 | **86.97** |
| $MDRIFT_a$ | M | 53.54 | 53.20 | 38.00 | 17.28 | 44.95 | 45.47 | **86.66** |
| $SEA_a$ | B | 88.41 | **88.48** | 70.82 | 74.93 | 86.43 | 82.68 | 87.60 |
| $SEA_g$ | B | 88.09 | **88.18** | 70.73 | 74.85 | 86.43 | 82.47 | 87.29 |
| $TCHESS_a$ | M | 95.66 | **96.03** | 57.58 | 57.60 | 68.89 | 32.26 | 93.76 |
| *synth avg* | | 84.39 | **84.45** | 61.15 | 59.66 | 76.62 | 60.80 | 82.83 |
| AIRL | B | 65.15 | **65.30** | 58.12 | 57.55 | 63.81 | 60.80 | 60.47 |
| COVTYPE | M | 83.23 | 83.72 | **95.32** | 94.59 | 82.33 | 93.99 | 95.12 |
| ELEC | B | 77.96 | 77.83 | 85.95 | 85.83 | 77.88 | **87.44** | 79.89 |
| POKER | M | **89.09** | 88.01 | 77.98 | 77.06 | 74.15 | 69.91 | 81.84 |
| WEATHER | B | 76.16 | 76.30 | 68.06 | 68.05 | 73.97 | 69.29 | **78.10** |
| *real avg* | | 78.32 | 78.23 | 77.09 | 76.62 | 74.43 | 76.29 | **79.08** |
| *overall avg* | | **81.86** | **81.86** | 67.79 | 66.73 | 75.71 | 72.59 | 81.27 |

We show the difference in the number of model transitions between Fast and Slow models in Figure 5.6. Plots correspond to $HYPER_f$ (Figure 5.6a) and ELEC (Figure 5.6b) and outline the difference in the number of transitions when using sliding windows (*top*), fading factors (*middle*) and sliding windows with discount (*bottom*). Given the small size of ELEC, it serves as a good example to observe the impact of using different strategies to select the top performer. In the case of $HYPER_f$, we see that after half the stream $FSC_{ff}$ defaults to the Slow model which performs consistently better. In general, we observe that the number of transitions is greater when using sliding windows and smaller when applying discount. Fading factors is placed in the middle, representing a good compromise between number of transitions and performance. This behavior is consistent across all the data sets used in our tests. On the other hand, test results show that the discount strategy represents a very conservative approach. In most of our tests this results in under-performance with respect to the other two strategies, which can be reasonably attributed to the reduced number of transitions between Fast and Slow models. However, we believe that this is a valid option in some cases, e.g. if one of the models is expected to work as a fallback option to protect against drops in performance due to concept drifts. By using a conservative approach

**Table 5.7:** Test results (mean kappa) for FSC and $FSC_{ff}$ against stand alone stream methods. Numbers in bold are top values. Type indicates if the stream is B:Binary or M:Multi-class.

| | | Kappa | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Type | FSC | $FSC_{ff}$ | PAC | PRCP | HT | HAT | SAM |
| $AGR_a$ | B | 81.01 | **81.24** | 9.31 | 10.84 | 78.62 | 61.23 | 36.18 |
| $AGR_g$ | B | 76.82 | **76.93** | 8.08 | 9.52 | 61.02 | 58.29 | 32.67 |
| $HYPER_f$ | B | 69.37 | 69.57 | 63.78 | 65.05 | 57.43 | 73.77 | **73.94** |
| $MDRIFT_a$ | M | 45.29 | 44.87 | 11.66 | 7.32 | 35.71 | 39.16 | **85.07** |
| $SEA_a$ | B | 75.23 | **75.39** | 39.96 | 47.81 | 71.10 | 63.53 | 73.85 |
| $SEA_g$ | B | 74.58 | **74.77** | 39.77 | 41.31 | 71.08 | 63.09 | 73.19 |
| $TCHESS_a$ | M | 78.75 | 79.39 | 51.52 | 51.55 | 50.11 | 22.56 | **92.87** |
| *synth avg* | | 71.58 | **71.74** | 32.01 | 33.34 | 60.72 | 54.52 | 66.82 |
| AIRL | B | 16.51 | 16.73 | 15.18 | 14.08 | 12.67 | **19.62** | 18.61 |
| COVTYPE | M | 60.70 | 62.24 | **92.48** | 91.31 | 59.19 | 90.33 | 92.17 |
| ELEC | B | 52.26 | 51.89 | 71.24 | 70.99 | 51.68 | **74.18** | 58.61 |
| POKER | M | **80.35** | 78.37 | 60.91 | 59.35 | 32.95 | 46.15 | 66.92 |
| WEATHER | B | 38.96 | 39.19 | 25.81 | 25.86 | 31.98 | 30.05 | **44.88** |
| *real avg* | | 49.76 | 49.68 | 53.12 | 52.32 | 37.69 | 52.07 | **56.24** |
| *overall avg* | | 62.49 | **62.55** | 40.81 | 41.25 | 51.13 | 53.50 | 62.41 |

we ensure that a different model is selected only if it represents the best approach in the long term.

Test results displayed in Table 5.6 and Table 5.7 show how FSC and $FSC_{ff}$ stand against established stream methods. The objective is to show that the Fast and Slow Classifier is an effective leveraging strategy where a stream model (HT) collaborates with a batch model (XGB), thus we consider the performance of HT as the baseline in our tests.

First, we observe that both FSC and $FSC_{ff}$ outperform the baseline, in average $FSC_{ff}$ is slightly better than FSC in terms of kappa and performs equally in terms of accuracy. Although differences in performance are marginal, we consider that $FSC_{ff}$ has the upper hand given that fading factors are cheaper to maintain and result in less model transitions.

We see that PAC and PRCP are the overall worts performers, both below the baseline. This can be justified in part by their lack of support for concept drift. However, it is important to point that PAC is the top performer for COVTYPE. Interestingly, although HAT shows good performance in some data sets, in average performs closely ($-3.13\%$ accuracy, $+2.37\%$ kappa) to the baseline despite the fact that it includes a drift detection mechanism. SAM on the other hand, consistently

performs above the baseline (+5.56% accuracy, +11.28% kappa). Thus, it is relevant to analyze how FSC stands out against SAM.

Experimental results confirm that $FSC_{ff}$ leverages the performance of a stand alone stream model (HT) by means of a batch model (XGB). We see that $FSC_{ff}$ outperforms other incremental learners for most data sets and in average performs slightly above SAM.

It is important to remind the reader that we explicitly use methods that are not designed to handle drift (HT and XGB) as base models for $FSC_{ff}$. This is because the objective of our tests is to show how the proposed framework performs in the presence of concept drift by means of the positive collaboration between stream and batch models. It is expected that, in actual applications, the proper batch and stream models will be used to configure FSC based on the type of data, available resources and other problem-specific requirements. This is why FSC is designed to be model-agnostic.

Based on test results we conclude that $FSC_{ff}$ represents a good compromise in terms of resources, model transitions and overall performance. The variety in our data sets outlines different scenarios where the Fast and Slow Classifier can effectively handle the interaction of Fast and Slow models as well as the regions where the default usage of only one of the models is the best option.

# 6 ADAPTIVE XGBOOST

The eXtreme Gradient Boosting (XGBoost or XGB) algorithm is a popular method for supervised learning tasks. XGB is an ensemble learner based on boosting that uses decision trees as weak learners. During training, new weak learners are added to the ensemble in order to minimize the objective function. Different to other boosting techniques, the complexity of the trees is also considered when adding weak learners: trees with lower complexity are desired. The wide-spread use of XGB in the machine learning community can be attributed to multiple factors: it has shown state-of-the-art performance in supervised learning tasks including classification and regressions, models are somewhat easy to interpret, is scalable to large datasets and, as an open source project, is available on multiple platforms. Also important, XGB provides (among other options) the flexibility to define the objective function to minimize. Although configuring the multiple hyper-parameters in XGB can be challenging, if done properly, it represents an advantage over other state-of-the-art methods.

An emerging approach to machine learning comes in the form of learning from evolving data streams. It provides an attractive alternative to traditional batch learning in multiple scenarios. An example is fraud detection for online banking operations, where training is performed on massive amounts of data. In this case, consideration of runtime is critical: waiting until the model is trained means that potential frauds may pass undetected. Another example is the analysis of communication logs for security, where storing all logs is impractical (and in most cases unnecessary). The requirement to store all data is an important limitation of methods that rely on doing multiple passes over the data.

Stream learning comprises a set of additional challenges to batch learning. The potentially infinite amount of data means that it is not practical to store it: stream methods have access to the data only once. Another important resource is time: stream methods are expected to process the data on the go given that new data arrives continuously. Different to batch learning, in stream learning, it is expected that models are always ready, in other words they can provide predictions at any moment in time. Finally, a common phenomenon in data streams, with direct impact on a model's performance, is the change in the relationship between features and learning targets, known as concept drift. Under this scenario, and without proper intervention, batch methods will fail because they will be trained on outdated data.

In contrast, stream methods continuously update their models and react to concept drift.

In this chapter we propose an adaptation of the XGB algorithm suitable to learning from evolving data streams. The proposed method is an instance of batch-incremental methods for data streams, where mini-batches of data are used during training. We aim to show how the proposed method stands in terms of the aforementioned challenges in stream learning. In our experimental evaluation, we compare the proposed method against batch-incremental and instance-incremental methods on data streams with concept drift.

This chapter is organized as follows: The proposed method is introduced in Section Section 6.1. Section 6.2 describes the methodology for our experiments whereas results are discussed in Section 6.3.

## 6.1 ADAPTING XGBOOST TO STREAM LEARNING

In this section, we present an adaptation of the XGB algorithm [28] suitable for learning from data streams.

### 6.1.1 Preliminaries

The goal of supervised learning is to predict the responses $Y = \{y_i\} : i \in \{1, 2, \ldots, n\}$ corresponding to a set of feature vectors $X = \{\vec{x}_i\} : i \in \{1, 2, \ldots, n\}$. Ensemble methods yield predictions $\hat{y}_i$ corresponding to a given input $\vec{x}_i$ by combining the predictions of all the members of the ensemble E. In this chapter, we focus on binary classification, that is, $y \in \{C_1, C_2\}$.

In the case of boosting, the ensemble E is created sequentially. In each iteration $k$, a new base function $f_k$ is selected and added to the ensemble so that the loss $\ell$ of the ensemble is minimized:

$$\ell(\mathrm{E}) = \sum_{k=1}^{K} \ell(Y, \hat{Y}^{(k-1)} + f_k(X)) + \Omega(f_k). \tag{6.1}$$

Here, $K$ is the number of ensemble members and each $f_k \in \mathcal{F}$ with $\mathcal{F}$ being the space of possible base functions. Commonly, this is the space of regression trees, so each base function is a step-wise constant predictor and the ensemble prediction $\hat{Y}$, which is simply the sum of all $K$ base functions, is also step-wise constant. The regularization parameter $\Omega$ penalizes complex functions.

The ensemble is created using forward additive modeling, where new trees are added one at a time. At step $k$, the training data is evaluated on existing members of the ensemble and the corresponding prediction scores $Y^{(k)}$ are used to drive the creation of new members of the ensemble. The base functions predictions are combined additively:

$$
\begin{aligned}
\hat{Y}^{(0)} &= 0 \\
\hat{Y}^{(1)} &= f_1(X) = \hat{Y}^{(0)} + f_1(X) \\
\hat{Y}^{(2)} &= f_1(X) + f_2(X) = \hat{Y}^{(1)} + f_2(X) \\
&\dots \\
\hat{Y}^{(k)} &= \sum_{k=1}^{K} f_k(X) = \hat{Y}^{(k-1)} + f_k(X)
\end{aligned}
\tag{6.2}
$$

The final prediction for a sample $\hat{y}_i$ is the sum of the predictions for each tree $f_k$ in the ensemble.

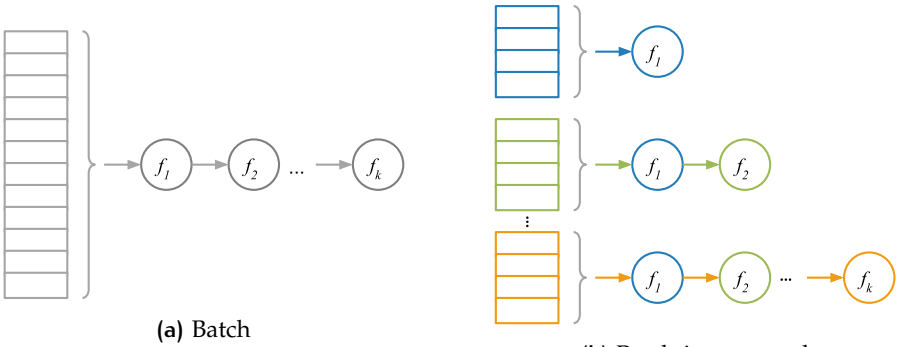
$$
\hat{y}_i = \sum_{k=1}^{K} f_t(x_i) \tag{6.3}
$$

### 6.1.2 Adaptive eXtreme Gradient Boosting

In the batch setting, XGB training is performed using all available data $(X, Y)$. However, in the stream setting, we receive new data over time, and this data may be subject to concept drift. A continuous stream of data can be defined as $A = \{(\vec{x}_t, y_t)\} | t = 1, \dots, T$ where $T \to \infty$, $\vec{x}_t$ is a feature vector, and $y_t$ the corresponding target $y_t$. We now describe a modified version of the XGB algorithm for this scenario, called Adaptive eXtreme Gradient Boosting (AXGB). AXGB uses an incremental strategy to create the ensemble. Instead of using the same data to select each base function $f_i$, it uses sub-samples of data obtained from non-overlapping (tumbling) windows. More specifically, as new data samples arrive, they are stored in a buffer $w = (\vec{x}_i, y_i) : i \in \{1, 2, \dots, W\}$ with size $|w| = W$ samples. Once the buffer is full, AXGB proceeds to train a single $f_k$. We can rewrite Eq. 6.2 as:

$$
\begin{aligned}
\hat{Y}^{(0)} &= 0 \\
\hat{Y}^{(1)} &= f_1(w_1) = \hat{Y}^{(0)} + f_1(w_1) \\
\hat{Y}^{(2)} &= f_1(w_2) + f_2(w_2) = \hat{Y}^{(1)} + f_2(w_2) \\
&\dots \\
\hat{Y}^{(k)} &= \sum_{k=1}^{K} f_k(w_k) = \hat{Y}^{(k-1)} + f_k(w_k)
\end{aligned}
\tag{6.4}
$$

The position $i$ of the new base function within the ensemble defines the way in which this function is obtained. If it is the first member of the ensemble, $f_1$, then the data in the buffer is used directly. If $i > 1$, then the data is passed through the ensemble and the residuals from the first $i - 1$ models in the ensemble are used to obtain the new base function. The ensemble creation strategies for the static and the incremental setup, described above, are outlined in Figure 6.1.




**(a)** Batch

**(b)** Batch-incremental

**Figure 6.1:** Ensemble creation strategies. The standard batch approach, Figure 6.1a, uses all available data to create all the members of the ensemble. On the other hand, the batch-incremental approach, Figure 6.1b, uses data windows to incrementally create the members of the ensemble as data becomes available.

### 6.1.3 Ensemble Update

Given that data streams are potentially infinite and may change over time, learned predictors must be updated continuously. Thus, it is essential to define a strategy to keep the AXGB ensemble updated once it is full. In the following, we consider two strategies for this purpose:

- A *push* strategy ($\text{AXGB}_{[p]}$), shown in Figure 6.2a, where the ensemble resembles a queue. New models are added to the ensemble as they arrive, and once the ensemble is full, older models are removed from the ensemble to make space.
- A *replacement* strategy ($\text{AXGB}_{[r]}$), shown in Figure 6.2b, where older members of the ensemble are replaced with newer ones.

Notice that in both cases we have to wait $K$ iterations to have a completely new ensemble. However, in $\text{AXGB}_{[r]}$, newer models have a more significant impact on predictions than older ones, while the reverse is true for $\text{AXGB}_{[p]}$.




**(a)** Push

**(b)** Replace

**Figure 6.2:** Ensemble creation strategies.

A requirement in stream learning is that models shall be ready to provide predictions at any time. Given the incremental nature of AXGB, if the window (buffer) size $W$ is fixed, the ensemble will require $K \cdot W$ samples to create the full ensemble. A negative aspect of this approach is that performance can be sub-optimal at the beginning of the stream. To overcome this, AXGB uses a dynamic window size $W$ that doubles in each iteration from a given minimum window size $W_{min}$ until a maximum window size $W_{max}$ is reached. In other words, it grows exponentially until reaching $W_{max}$. The window size, $W(i)$, for the $i$th iteration is defined as:

$$W(i) = min(W_{min} \cdot 2^i, W_{max}) \tag{6.5}$$

From Eq. 6.5, we see that the number of iterations $i$ required to reach the maximum window size is:

$$i = \left\lceil \log_2 \left( \frac{W_{max}}{W_{min}} \right) \right\rceil \tag{6.6}$$

Similarly, we see that the number of samples required to create $K$ models to fill the ensemble is smaller when using the dynamic window size approach than when using a fixed window size because

$$\sum_{i=0}^{K-1} W_{min} \cdot 2^i \ll K \cdot W. \tag{6.7}$$

Because we monotonically increase the window size, we see that both our above updating strategies replace base functions trained on small windows with newer ones trained on more data.

### 6.1.4 Handling Concept Drift

A common phenomenon in data streams is that the relationship between features and targets changes over time. This is known as concept drift. Although the incremental strategy used by AXGB to create the ensemble indirectly deals with concept drift—new members of the ensemble are added based on newer data—it may be too slow to adjust to rapid drift. Hence, we use ADWIN [13], a popular concept drift detector, to track changes in the performance of AXGB, as measured by a performance metric such as classification accuracy. We use subscript $_A$ to denote ADWIN, therefore the concept-drift-aware version of AXGB is referred in the following as AXGB$_A$.

To detect change, ADWIN maintains a dynamically sized window that grows when there is no change and shrinks otherwise. If the difference in mean performance between two adjacent sub-windows exceeds a confidence parameter $\delta$, ADWIN will signal that a change

has occurred and reduce the window size. AXGB$_A$ uses the change detection signal obtained from ADWIN to trigger a mechanism to update the ensemble. This mechanism works as follows:

1. Reset the size of the window $w$ to the defined minimum size $W_{min}$. In other words, reset the dynamic window sizing.
2. Train and add new members to the ensemble depending on the chosen strategy:
   a) Push: New ensemble members are pushed into the ensemble while older ones are removed from it. Given that new models are trained on increasing window sizes, this means that new models will be added at a faster rate initially; this effectively works as a *flushing* strategy to update the ensemble.
   b) Replacement: The index used to replace old members of the ensemble is reset so that it points to the beginning of the ensemble. There are two considerations regarding this approach: First, new ensemble members replace older ones. Second, new ensemble models are trained without considering the residuals of old models that were trained on the older concept.

A comparison of how the ensemble is updated in the different versions of AXGB is presented in Table 6.1. For this example, the size of the ensemble is set to $|\mathrm{E}| = K = 3$. We see how new models are added to the ensemble before and after the ensemble is full. Additionally, we show how AXGB$_A$ proceeds upon detecting concept drift.

**Table 6.1:** Ensemble creation comparison.

| | AXGB | | AXGB$_A$ | |
|---|---|---|---|---|
| iteration | Push | replace | Push | Replace |
| 1 | $f_1$ | $f_1$ | $f_1$ | $f_1$ |
| 2 | $f_1, f_2$ | $f_1, f_2$ | $f_1, f_2$ | $f_1, f_2$ |
| 3 | $f_1, f_2, f_3$ | $f_1, f_2, f_3$ | $f_1, f_2, f_3$ | $f_1, f_2, f_3$ |
| 4 | $f_2, f_3, f_4$ | $f_4, f_2, f_3$ | $f_2, f_3, f_4$ | $f_4, f_2, f_3$ |
| 5 | $f_3, f_4, f_5$ | $f_4, f_5, f_3$ | $f_3, f_4, f_5$ | $f_4, f_5, f_3$ |
| Concept Drift- | | - | Reset $w$ size | Reset $w$ size |
| 6 | $f_4, f_5, f_6$ | $f_4, f_5, f_6$ | $f_4, f_5, f_6$ | $f_6, f_5, f_3$ |

## 6.2 EXPERIMENTAL EVALUATION

In this section, we describe the methodology of our tests, which we classify into the following categories: predictive performance, hyper-parameter relevance, memory usage / model complexity, and training time.

1. **Predictive performance** . Our first set of tests evaluates the predictive performance of AXGB. For this we use both, synthetic and real-world data sets. We then proceed to compare AXGB against other learning methods. This comparison is defined by the nature of the learning method as follows:
   a) *Batch-incremental methods*. In this type of learning methods, batches of samples are used to incrementally update the model. We compare AXGB against a batch-incremental ensemble created by combining multiple per-batch base models. New base models are trained independently on disjoint batches of data (windows). When the ensemble is full, older models are replaced with newer ones. Predictions are formed by majority vote. In order to compare this approach with AXGB, we use XGB as the base batch-learner to learn an ensemble for each batch. Thus, our batch-incremental model is an ensemble of XGB ensembles. We refer to this batch-incremental method as BXGB. We also consider the Accuracy-Weighted Ensemble with Decision Tree as the base batch-learner. We refer to this method as AWE-J48. We choose this configuration since AWE-J48 is reported as the top batch-incremental performer in [104], so it serves as a baseline for batch-incremental methods.
   b) *Instance-incremental methods*. We are also interested in comparing AXGB against methods that update their model one instance at a time. The following instance-incremental methods are used in our tests: Adaptive Random Forest (ARF), Hoeffding Adaptive Tree (HAT), Leverage Bagging with Hoeffding Tree as base learner ($LB_{HT}$), Oza Bagging with Hoeffding Tree as base learner ($OB_{HT}$), Self Adjusting Memory with kNN ($SAM_{kNN}$) and the Ensemble of Restricted Hoeffding Trees (RHT). In [104], $LB_{HT}$ is reported as the top instance-incremental performer.
2. **Hyper-parameter relevance**. The XGB algorithm relies on multiple hyper-parameters which can make the model hard to tune for different problems. In this context, we are interested in analyzing the impact of hyper-parameters in AXGB. For this purpose, we use a tuning setup where a model is trained on the first 30% of the data stream using different combinations of hyper-parameters. Then, the best performers during the training phase

are evaluated on the remaining 70% of the stream. To evaluate the influence of hyper-parameters we compare performance between AXGB and BXGB methods.

3. **Memory usage and model complexity**. The potentially infinite number of samples in data streams requires resources such as time and memory to be properly managed. We use the total number of nodes in the ensemble to gain insight into memory usage and model complexity as AXGB is trained on a stream. We compare the proposed versions of AXGB against a baseline batch-model trained using XGB on all the data of the stream. Thus, the baseline number of nodes in the batch-model is expected to be larger than the number of nodes in incremental-models which evolve with the stream. By analyzing memory usage and model complexity we get insights on the evolution of the model over time (samples).

4. **Training time**. Another relevant way to analyze the proposed method is in terms of training time. We compare the training time of the different versions of AXGB against XGB, reporting results in terms of training time (seconds) and in terms of throughput (samples per second).

Our implementation of AXGB is based on the official XGB C-API[1] on top of scikit-multiflow[2] [89] (described in Chapter 7). Tests are performed using the official XGB implementation, the implementations of ARF, RHT and AWE in MOA [16], and for the rest of the methods, the implementations available in scikit-multiflow. Default parameters of the algorithms are used unless otherwise specified.

In the following, we provide a short description of the synthetic and real world data sets used in our tests. All data sets are publicly available.

**AGRAWAL** Based on the Agrawal generator [3], represents a data stream with six nominal and three numerical features. Different functions map instances into two different classes. Three abrupt drifts are simulated for $AGR_a$ and three gradual drifts for $AGR_g$.

**HYPER** A data stream with fast incremental drifts where a $d$-dimensional hyperplane changes position and orientation. Obtained from a random hyperplane generator [64].

**SEA** A data stream with three numerical features where only two attributes are related to the target class. Created using the SEA generator [113]. Three abrupt drifts are simulated for $SEA_a$ and three gradual drifts for $SEA_g$.

1 https://github.com/dmlc/xgboost
2 https://github.com/scikit-multiflow/scikit-multiflow

**Table 6.2:** Data sets. [Type] S: synthetic data; R: real world data. [Drifts] A: abrupt, G: gradual; $I_f$: incremental fast.

| | # instances | # features | # classes | Type | Drift |
|---|---|---|---|---|---|
| $AGR_a$ | 1000000 | 9 | 2 | S | A |
| $AGR_g$ | 1000000 | 9 | 2 | S | G |
| $HYPER_f$ | 1000000 | 10 | 2 | S | $I_f$ |
| $SEA_a$ | 1000000 | 3 | 2 | S | A |
| $SEA_g$ | 1000000 | 3 | 2 | S | G |
| AIRL | 539383 | 7 | 2 | R | - |
| ELEC | 45312 | 6 | 2 | R | - |
| WEATHER | 18159 | 8 | 2 | R | - |

**AIRLINES** Real world data containing information from scheduled departures of commercial flights within the US. The objective is to predict if a flight will be delayed.

**ELECTRICITY** Data from the Australian New South Wales electricity market where prices are not fixed but change based on supply and demand. The two target classes represent changes in the price (up or down).

**WEATHER** Contains weather information collected between 1949–1999 in Bellevue, Nebraska. The goal is to predict if it will rain or not on a given date.

A summary of the characteristics of the above data sets is available in Table 6.2. For further details on AGRAWAL, HYPER and SEA, we refer the reader to [54].

## 6.3 RESULTS

The results discussed in this section provide information about predictive performance, parameter relevance, and resources consumption (memory and time) for the different versions of AXGB. The subsections follow the order used to described the test categories in Section 6.2.

### 6.3.1 Predictive Performance

We evaluate the performance of AXGB against other batch-incremental methods and against instance-incremental methods. We use prequential evaluation [32], where predictions are generated for a sample in the stream before using it to train/update the model. We use classification

accuracy as the metric in our tests in order to measure performance. First, we compare the different versions of AXGB ($AXGB_{[p]}$, $AXGB_{A[p]}$, $AXGB_{[r]}$ and $AXGB_{A[r]}$) against two batch-incremental methods: BXGB and AWE-J48. The parameters used to configure these methods are available in Table 6.3.

Results comparing against batch-incremental methods are available in Table 6.4. We see that the overall top performer in this test is $AXGB_{A[r]}$, but it is barely distinguishable from $AXGB_{[r]}$. Next are the versions of AXGB using the push strategy. Interestingly, we find that AWE-J48 performs better than BXGB, which comes last in this test. This is noteworthy considering that the base learner in AWE-J48 (a single decision tree) is simpler than the one in BXGB (an ensemble of trees generated using XGB).

These tests provide insights into the different versions of AXGB. We see that, in the push-strategy versions ($AXGB_{[p]}$, $AXGB_{A[p]}$), tracking performance to detect concept drift ($AXGB_{A[p]}$) provides a consistent advantage over the drift-unaware approach ($AXGB_{[p]}$). The reason for this is that, as expected, $AXGB_{A[p]}$ reacts faster to changes in performance: When a drift is detected, the window size is reset and new models are quickly added to the ensemble, flushing out older models. This is not the case for methods using the replace-strategy ($AXGB_{[r]}$, $AXGB_{A[r]}$). In this case, we find that performance gains between methods are marginal ($\approx 0.1\%$). Notice that $AXGB_{[r]}$ is actually slightly better than $AXGB_{A[r]}$ for most datasets. These results are significant given the compromise between the computational overhead of tracking concept drift and the gains in performance. We analyze this trade-off when discussing results of the timing tests.

The $AGR_a$ and $AGR_g$ data sets are interesting examples for analysis, given that the type (abrupt, gradual) and position (25%, 50% and 75%) of concept drifts are known. The impact of concept drift on all the models considered in our tests can be observed in Figure 6.3 for both data sets. We see a drop in performance after a concept change. It is worth noting that this drop is larger when change is

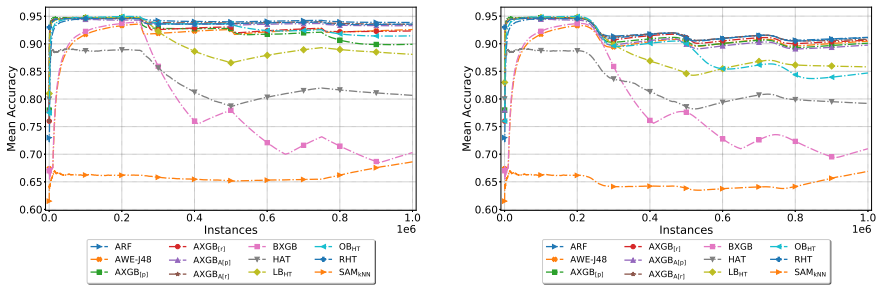
**Table 6.3:** Parameters used for batch-incremental methods.

| Parameter | AXGB * | BXGB | AWE-J48 |
|---|---|---|---|
| ensemble size | 30 | 30 | 30 |
| ensemble size (base learner) | - | 30 | - |
| max window size | 1000 | 1000 | 1000 |
| min window size | 1 | - | - |
| max depth | 6 | 6 | - |
| learning rate | 0.3 | 0.3 | - |

* The same parameter configuration is used for all variations: $AXGB_{[p]}$, $AXGB_{A[p]}$, $AXGB_{[r]}$ and $AXGB_{A[r]}$.

**Table 6.4:** Comparing performance of AXGB vs batch-incremental methods.

| Data set | $AXGB_{[p]}$ | $AXGB_{[r]}$ | $AXGB_{A[p]}$ | $AXGB_{A[r]}$ | BXGB | AWE-J48 |
|---|---|---|---|---|---|---|
| $AGR_a$ | 0.919 | **0.931** | 0.927 | 0.927 | 0.703 | 0.926 |
| $AGR_g$ | 0.896 | **0.907** | 0.897 | 0.901 | 0.710 | 0.905 |
| AIRL | 0.604 | 0.613 | 0.611 | 0.609 | **0.641** | 0.599 |
| ELEC | 0.718 | 0.722 | 0.740 | **0.752** | 0.702 | 0.614 |
| $HYPER_f$ | 0.822 | **0.848** | 0.825 | **0.848** | 0.756 | 0.777 |
| $SEA_a$ | 0.865 | **0.875** | 0.866 | 0.874 | 0.856 | 0.860 |
| $SEA_g$ | 0.863 | **0.873** | 0.863 | 0.872 | 0.857 | 0.860 |
| WEATHER | 0.765 | 0.765 | **0.767** | 0.761 | 0.737 | 0.712 |
| **average** | 0.807 | 0.817 | 0.812 | **0.818** | 0.745 | 0.782 |



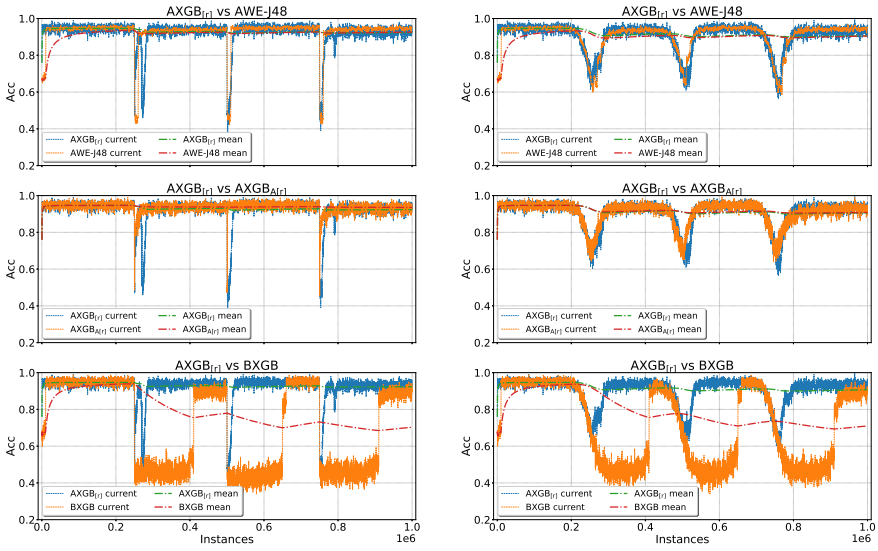

**(a)** $AGR_a$ data set.

**(b)** $AGR_g$ data set.

**Figure 6.3:** Example of the impact of concept drift on the mean performance of models compared in our tests. The $AGR_a$, Figure 6.3a, and $AGR_g$, Figure 6.3b, data sets contain drifts at the 25%, 50% and 75% of the stream.

gradual Figure 6.3b, this can be attributed to the transition phase between concepts. During this period, old and new concepts are present, thus recovering takes longer and the model's performance is negatively impacted.

In Figure 6.4, we observe the behavior of $AXGB_{[r]}$ vs AWE-J48, $AXGB_{A[r]}$ and BXGB. Consider the abrupt changes in $AGR_a$, Figure 6.4a. We see that $AXGB_{[r]}$ recovers quickly after the three drifts. Although there is a sudden drop during the recovery after the first drift (which is quickly corrected), this is not observed after the other two drifts. $AXGB_{A[r]}$ is the faster to adapt to the drift, which is expected considering that ADWIN detects the drift and the ensemble is updated accordingly. AWE-J48 behaves in a similar fashion to $AXGB_{[r]}$. Worst batch-incremental performer for $AGR_a$ is BXGB whose performance drops after the drifts and can not adapt quickly. Additionally, recovery is only efficient for the third concept, whereas recovery is sub-optimal for the second and fourth concepts.




**(a)** $AGR_a$ data set. **(b)** $AGR_g$ data set.

**Figure 6.4:** Performance of $AXGB_{[r]}$ vs AWE-J48, $AXGB_{A[r]}$ and BXGB. Values marked as *current* correspond to accuracy on a sliding window of 200 samples.

In the case of gradual change, Figure 6.4b, the transition phase between concepts provides evidence on how models adapt in the presence of two concepts. We see that $AXGB_{[r]}$ and AWE-J48 show similar performance and adaptability during and after concept changes. In the case of $AXGB_{A[r]}$, we notice that it is still faster to react to concept drift, but $AXGB_{[r]}$ is close behind during the transition period. This is translated into the marginal gains observed for this test. Finally, BXGB is still the worst performer. This can be attributed to the combination of concept-drift-unawareness and the *slow* ensemble update which is related to the configuration used, Table 6.3. This is further analyzed in the hyper-parameter relevance test.

Next, we compare AXGB against instance-incremental methods. Results are shown in Table 6.5. For AXGB, we only show results of $AXGB_{[r]}$ and $AXGB_{A[r]}$. We see that the top performer in this test is ARF, closely followed by RHT. AXGB's performance is not on par with that of the top performers, but it is fundamental to note that (i) these results are consistent with those in [104], where instance-incremental methods outperform batch-incremental methods, and (ii) both $AXGB_{[r]}$ and $AXGB_{A[r]}$ are placed in the middle tier between $LB_{HT}$ and $OB_{HT}$. HAT and $SAM_{kNN}$ are the worst performers in our tests. We believe that these findings serve to indicate the potential of extreme gradient boosting for data streams.

**Table 6.5:** Comparing AXGB vs instance-incremental methods.

| Dataset | AXGB | $AXGB_A$ | ARF | HAT | $LB_{HT}$ | $OB_{HT}$ | $SAM_{kNN}$ | RHT |
|---|---|---|---|---|---|---|---|---|
| $AGR_a$ | 0.931 | 0.927 | **0.939** | 0.807 | 0.881 | 0.915 | 0.686 | 0.936 |
| $AGR_g$ | 0.907 | 0.901 | **0.912** | 0.792 | 0.858 | 0.847 | 0.669 | 0.911 |
| AIRL | 0.613 | 0.609 | **0.680** | 0.608 | 0.670 | 0.658 | 0.605 | 0.648 |
| ELEC | 0.722 | 0.752 | 0.855 | **0.874** | 0.836 | 0.794 | 0.799 | 0.873 |
| $HYPER_f$ | 0.848 | 0.848 | 0.849 | 0.869 | 0.814 | 0.806 | 0.870 | **0.896** |
| $SEA_a$ | 0.875 | 0.874 | **0.897** | 0.827 | 0.891 | 0.869 | 0.876 | 0.889 |
| $SEA_g$ | 0.873 | 0.872 | **0.893** | 0.825 | 0.889 | 0.869 | 0.873 | 0.885 |
| WEATHER | 0.765 | 0.761 | **0.791** | 0.693 | 0.783 | 0.749 | 0.781 | 0.758 |
| **average** | 0.817 | 0.818 | **0.852** | 0.787 | 0.828 | 0.813 | 0.770 | 0.850 |

### 6.3.2 Hyper-parameter Relevance

As previously mentioned, hyper-parameters play a key role in the performance of XGB. Thus, we also need to consider their impact in AXGB. In order to do so, we present results obtained by running multiple tests on different combinations of key parameters: the maximum depth of the trees, the learning rate (eta), the ensemble size, and the maximum and minimum window size. To cover a wide range of values for each parameter, we use a grid search based on the grid parameters specified in Table 6.6. The parameter grid corresponds to a total of $4 \times 4 \times 5 \times 5 \times 3 = 1200$ combinations. For this test, we compare the following XGB-based methods: $AXGB_{[p]}$, $AXGB_{A[p]}$ and BXGB.

For establishing the effect of parameter tuning, the test is split into two phases: training and optimization on the first 30% of the stream—using this validation data to evaluate all parameter combinations in the grid and choosing the best one using prequential evaluation of classification accuracy—and performance evaluation on the remaining 70% of the stream to establish accuracy of the parameter-optimized algorithm by evaluating the algorithm with the identified parameter settings using prequential evaluation on this remaining data. The

**Table 6.6:** Parameter grid used to evaluate hyper-parameters relevance.

| Parameter | Values |
|---|---|
| max depth | 1, 5, 10, 15 |
| learning rate | 0.01, 0.05, 0.1, 0.5 |
| ensemble size | 5, 10, 25, 50, 100 |
| max window size | 512, 1024, 2048, 4096, 8192 |
| min window size | 4, 8, 16 |

**Table 6.7:** Parameter tuning results.

| Dataset | *Ref* $\text{AXGB}_{[p]}$ | *Tuning* $\text{AXGB}_{[p]}$ | *Ref* $\text{AXGB}_{A[p]}$ | *Tuning* $\text{AXGB}_{A[p]}$ | *Ref* BXGB | *Tuning* BXGB |
|---|---|---|---|---|---|---|
| $\text{AGR}_a$ | 0.881 | 0.927 | **0.933** | 0.931 | 0.727 | 0.930 |
| $\text{AGR}_g$ | 0.898 | 0.906 | 0.902 | 0.905 | 0.728 | **0.909** |
| AIRL | 0.616 | 0.627 | 0.588 | 0.628 | 0.632 | **0.639** |
| ELEC | 0.713 | 0.736 | 0.658 | 0.739 | 0.631 | **0.742** |
| $\text{HYPER}_f$ | 0.816 | 0.873 | 0.833 | 0.876 | 0.754 | **0.904** |
| $\text{SEA}_a$ | 0.879 | 0.889 | 0.881 | **0.892** | 0.854 | 0.890 |
| $\text{SEA}_g$ | 0.877 | 0.888 | 0.878 | **0.889** | 0.855 | 0.888 |
| WEATHER | 0.755 | 0.767 | 0.758 | 0.765 | 0.703 | **0.782** |
| average | 0.804 | 0.827 | 0.804 | 0.828 | 0.736 | **0.835** |

ensemble model is trained from scratch in this second phase. This strategy is limited in the sense that the nature of the validation data, including concepts drifts, is assumed to be similar to that of the remaining data, but it provides insight into the importance of hyper-parameters.

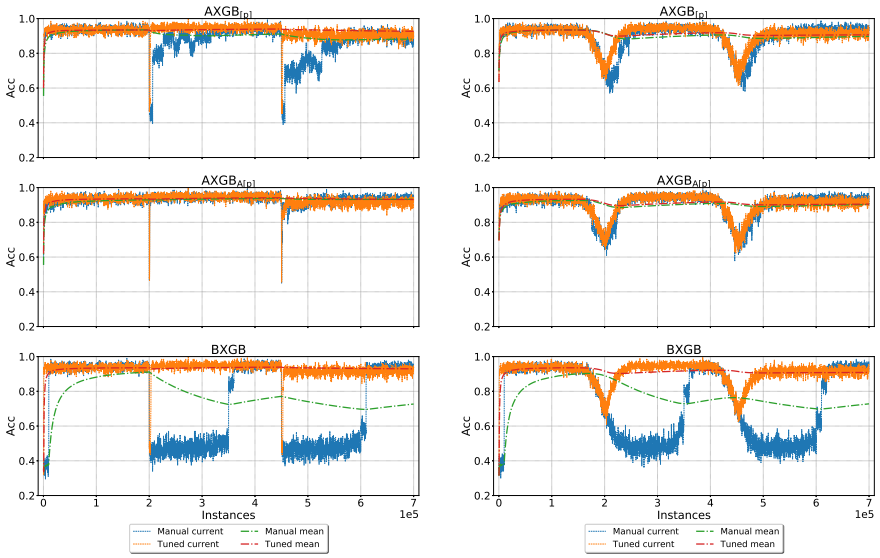
Results from this experiment are available in Table 6.7. Reported results correspond to measurements obtained with parameter tuning (*Tuning*) vs reference results (*Ref*) obtained using the hand picked parameters in Table 6.3, building an ensemble from scratch on the same 70% portion of the stream. The optimized parameter configurations are listed in Table 6.8.

We can see that optimizing hyper-parameters clearly benefits all methods. As previously discussed, hyper-parameters can provide an advantage over other methods. In this case, under-performers are now on par or above $\text{LB}_{\text{HT}}$. Surprisingly, BXGB obtains the largest boost in performance and is now the method that performs best. When analyzing the parameter configurations, we see that BXGB favors smaller values for max window size, learning rate and max depth. The observed increase in performance can be attributed to the impact of the hyper-parameters on the base learner in BXGB (batch XGB models): BXGB is an ensemble of ensembles. Another factor to consider is the small window sizes. In practice, having smaller windows means that models are replaced faster as the stream progresses and this can ameliorate the lack of drift awareness to some degree. It is reasonable that the same applies to the reduction in the performance gap between $\text{AXGB}_{[p]}$ and $\text{AXGB}_{A[p]}$. In the case of AXGB, we notice that the learning rate has a consistent impact on performance (lower is better), followed by max window size and max depth. Finally, these tests reveal the contrast in the impact of the ensemble size on the two versions of AXGB. While $\text{AXGB}_{[p]}$ benefits from a smaller ensemble size, the

**Table 6.8:** Selected parameters in tuning tests.

| Dataset | ensemble size | max window size | min window size | learning rate | max depth |
|---|---|---|---|---|---|
| | | $AXGB_{[p]}$ | | | |
| $AGR_a$ | 10 | 512 | 16 | 0.1 | 15 |
| $AGR_g$ | 25 | 512 | 16 | 0.1 | 15 |
| AIRL | 5 | 512 | 4 | 0.1 | 5 |
| ELEC | 5 | 1024 | 4 | 0.1 | 5 |
| $HYPER_f$ | 25 | 512 | 8 | 0.1 | 5 |
| $SEA_a$ | 25 | 2048 | 4 | 0.05 | 10 |
| $SEA_g$ | 25 | 2048 | 4 | 0.05 | 10 |
| WEATHER | 50 | 512 | 16 | 0.1 | 10 |
| | | $AXGB_{A[p]}$ | | | |
| $AGR_a$ | 25 | 512 | 8 | 0.1 | 15 |
| $AGR_g$ | 50 | 512 | 4 | 0.1 | 15 |
| AIRL | 10 | 512 | 8 | 0.1 | 5 |
| ELEC | 100 | 512 | 16 | 0.1 | 10 |
| $HYPER_f$ | 50 | 512 | 8 | 0.1 | 5 |
| $SEA_a$ | 25 | 8192 | 4 | 0.05 | 15 |
| $SEA_g$ | 25 | 8192 | 8 | 0.05 | 10 |
| WEATHER | 50 | 1024 | 16 | 0.1 | 15 |
| | | BXGB | | | |
| $AGR_a$ | 10 | 512 | - | 0.1 | 10 |
| $AGR_g$ | 25 | 512 | - | 0.05 | 10 |
| AIRL | 5 | 512 | - | 0.05 | 5 |
| ELEC | 10 | 512 | - | 0.1 | 1 |
| $HYPER_f$ | 10 | 512 | - | 0.5 | 1 |
| $SEA_a$ | 25 | 1024 | - | 0.05 | 10 |
| $SEA_g$ | 25 | 1024 | - | 0.05 | 15 |
| WEATHER | 50 | 512 | - | 0.1 | 5 |

contrary applies to $AXGB_{A[p]}$. This supports the intuition that drift-aware methods can benefit from larger ensembles (to build complex models) which adapt faster in the presence of drift by triggering the corresponding ensemble update mechanism. On the other hand, batch-incremental methods without explicit drift detection mechanisms rely on their natural ability to adapt, which can be counterproductive with large ensemble sizes. It is important to note that although BXGB is the top performer in this test, it is not efficient in terms of resources (time and memory), which represents a compromise in stream learning applications where resources are limited.




**(a)** $AGR_a$ data set. **(b)** $AGR_g$ data set.

**Figure 6.5:** Manual selection vs hyper-parameter tuning. Values marked as *current* correspond to accuracy measured on a sliding window of 200 samples.

Again, we focus on $AGR_a$ and $AGR_g$ to further analyze the impact of hyper-parameters on $AXGB_{[p]}$, $AXGB_{A[p]}$ and BXGB. Results shown in Figure 6.5 correspond to the performance evaluation on the latter 70% of the stream. On both data sets, two concept drifts are available on this sub-section. In the case of abrupt drift, Figure 6.5a, we see that the three models adapt quickly and effectively. As discussed, the improvement is more evident in the case of BXGB which is now on par with the rest. Similar results are observed for gradual drift, Figure 6.5b. In this case we see that $AXGB_{[p]}$ is able to adapt faster during the concept transition phase. Again, BXGB is the one showing the biggest improvement.

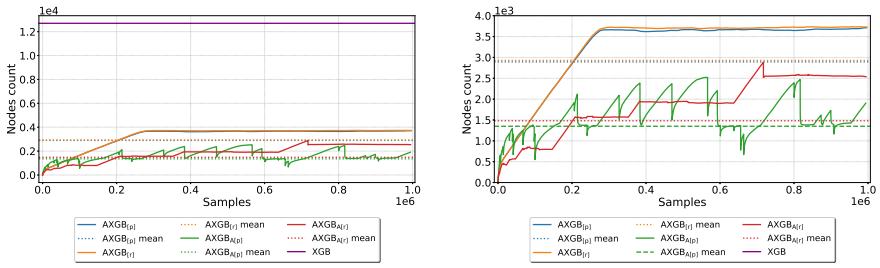
### 6.3.3 Memory and Model Complexity

In this section, we analyze memory usage by the proposed methods during the learning process. For this purpose, we count the number of nodes in the ensemble, including both leaf nodes and internal nodes of each tree. This approach also provides some intuition regarding the model's complexity. We perform this test on a synthetic data set with 40 features (only 30 informative) and 5% noise, corresponding to the *Madelon* data set as described in [56]. We use 1 million samples for training and calculate the number of nodes in the ensemble to get an estimate of the model size. Models are trained using the following configuration: ensemble size = 30, max window size = 10K, min window size = 1, learning rate = 0.05, max depth = 6. We measure the

number of nodes as new members of the ensemble are introduced. Results for this test are available in Figure 6.6. We use the number of nodes generated by the XGB algorithm as a reference. It is important to note that this number is constant since it represents the size of the model trained on all the data. In Figure 6.6a, we can compare the number of nodes in the batch model vs the stream models. The same information is displayed in Figure 6.6a on a different scale to better visualize the behaviour of stream classifiers across the stream.

In Figure 6.6a, we see that AXGB$_{[p]}$ and AXGB$_{[r]}$ have similar behaviour. As the stream progresses, the number of nodes added to the ensemble increases until reaching a plateau. This is expected since new models are trained on larger windows of data. The plateau corresponds to the region where the ensemble is complete and old members of the ensemble are replaced by new members trained on equally large windows. On the other hand, AXGB$_{A[p]}$ and AXGB$_{A[r]}$ also exhibit an incremental increase in the number nodes over the stream—at a lower rate than the AXGB versions—but with some interesting differences. In AXGB$_{A[p]}$, we see multiple drop points in the nodes count, which is attributed to the ensemble update mechanism. When drift is detected, the window size is reset and new models are pushed into the ensemble, in other words, simpler models are quickly introduced into the ensemble. In contrast, the number of nodes in AXGB$_{A[r]}$ increases steadily. It is important to remember that, despite the difference in number of nodes between AXGB$_{[r]}$ and AXGB$_{A[r]}$, the difference in performance is marginal, as discussed in Section 6.3.1.

We also analyze AXGB by counting the number of models in the ensemble across the stream, shown in Figure 6.7. Notice that the number of models reach the maximum value when the ensemble is full; from that point on, new models replace old ones. As anticipated, we see that AXGB$_{A[p]}$ fills the ensemble quickly at the beginning of the stream because concept drift detection triggers the reset of the window size and speeds up the introduction of new models. AXGB$_{[p]}$ and AXGB$_{[r]}$ fill the ensemble at a slower rate and finish filling the




**(a)** Total nodes in ensemble. **(b)** Total nodes in ensemble, detail.

**Figure 6.6:** Insight into ensemble complexity by number of nodes over the stream.

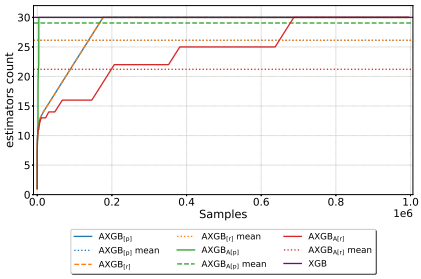


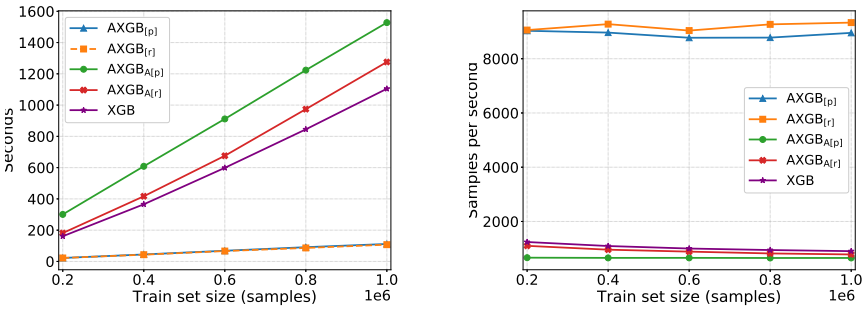

**Figure 6.7:** Number of ensemble members as the stream is processed.

ensemble before the 200K mark. This is in line with our expectations given the introduction of new models trained on increasing window sizes as defined in Eq. 6.5. Finally, AXGB$_{A[r]}$ is the slowest to fill the ensemble at around the 700K mark. This is expected given that upon drift detection, AXGB$_{A[r]}$ starts replacing older models at the beginning of the ensemble.

It is important to mention that additional memory resources are used by the different AXGB variants given their batch-incremental nature: mini-batches are accumulated in memory before they are used to fit a tree. In this sense, other things being equal, instance-incremental methods are more memory efficient. However, these results show that all versions of AXGB keep the size of the model under control, a critical feature when facing theoretically infinite data streams.

### 6.3.4 Training Time

Finally, we measure training time for the different versions of AXGB. We use as reference the time required to train an XGB model using the same test configuration as in Section 6.3.3 with the following data set sizes: 200K, 400K, 600K, 800K and 1M. Results correspond to the average time after running the experiments 10 times for each data set




**(a)** Training time. **(b)** Throughput.

**Figure 6.8:** Time/throughput test results.

size and for each classifier. Measurements are shown in Figure 6.8 in terms of time (seconds) and in Figure 6.8b in terms of throughput (samples per second). These tests show that the fastest learners are $\text{AXGB}_{[p]}$ and $\text{AXGB}_{[r]}$, both showing small change in training time as the number of instances increases. This is an important feature given that, along with memory usage, training time plays a key role in stream learning applications. On the other hand, the drift-aware versions ($\text{AXGB}_{A[p]}$ and $\text{AXGB}_{A[r]}$) have similar behaviour in terms of time compared to the batch model (XGB) while being slightly slower. This can be attributed to the overhead from the drift-detection process, which implies getting predictions for each instance and keeping the drift detector statistics. Additionally, we see that $\text{AXGB}_{A[p]}$ is the slowest classifier, which might be related to the overhead incurred by predicting using a larger ensemble, given that the ensemble is quickly filled (as seen in Figure 6.7).

# 7 SCIKIT-MULTIFLOW

Recent years have witnessed the proliferation of Free and Open Source Software (FOSS) in the research community. Specifically, in the field of machine learning, researchers have benefited from the availability of different frameworks that provide tools for faster development, allow replicability and reproducibility of results and foster collaboration. Following the FOSS principles, we introduce scikit-multiflow, a Python framework to implement algorithms and perform experiments in the field of machine learning on evolving data streams. scikit-multiflow is inspired in the popular frameworks scikit-learn, MOA and MEKA.

scikit-learn [95] is the most popular open source software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forest, gradient boosting, k-means and DBSCAN, and is designed to inter-operate with the Python numerical and scientific packages NumPy and SciPy.

MOA [16] is the most popular open source framework for data stream mining, with a very active growing community. It includes a collection of machine learning algorithms (classification, regression, clustering, outlier detection, concept drift detection and recommender systems) and tools for evaluation. Related to the WEKA project [57], MOA is also written in Java, while scaling to more demanding problems.

The MEKA project [106] provides an open source implementation of methods for multi-label learning and evaluation. In multi-label classification, the aim is to predict multiple output variables for each input instance. This different from the 'standard' case (binary, or multi-class classification) which involves only a single target variable.

As a multi-output streaming framework, scikit-multiflow serves as a bridge between research communities that have flourished around the aforementioned frameworks, providing a common ground where they can thrive. scikit-multiflow assists on the democratization of Stream Learning by bringing this research field closer to the machine learning community, given the increasing popularity of the Python programing language. The objective is two-folded: First, fill the void in Python for a stream learning framework which can also interact with available tools such as scikit-learn and extend the set of available state-of-the-art methods on this platform. Second, provide a set of tools to facilitate the development of stream learning research, an example is [85].

It is important to notice that scikit-multiflow complements scikit-learn, whose primary focus is batch learning, expanding the set of free

and open source tools for stream learning. In addition, scikit-multiflow can be used within Jupyter Notebooks, a popular interface in the data science community. Special focus in the design of scikit-multiflow is to make it friendly to new users and familiar to experienced ones.

scikit-multiflow contains stream generators, learning methods (classification and regression), concept drift detectors and evaluation methods. Examples of available stream generators are: Agrawal [4], Hyperplane [64], Led [21], Mixed [48], Random-RBF, Random-RBF with drift, Random Tree, SEA [113], SINE [48], STAGGER [48], Waveform,

**Table 7.1:** Available methods in scikit-multiflow. Methodologies on the left, and frameworks on the right.

| | | | | | | Java | | Python | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Algorithm** | **Classification** | **Regression** | **Single-Output** | **Multi-Output** | **Drift Detection** | MOA | MEKA† | scikit-learn† | scikit-multiflow | **Reference** |
| kNN | ✓ | | ✓ | | | ✓ | ✓ | ✓ | ✓ | [20] |
| kNN + ADWIN | ✓ | | ✓ | | | ✓ | | | ✓ | [15] |
| SAM kNN | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | [79] |
| Hoeffding Tree | ✓ | | ✓ | | | ✓ | | | ✓ | [35] |
| Hoeffding Adaptive Tree | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | [14] |
| FIMT-DD | | ✓ | ✓ | | ✓ | ✓ | | | ✓ | [65] |
| Adaptive Random Forest | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | [54] |
| Oza Bagging | ✓ | | ✓ | | * | ✓ | | | ✓ | [93] |
| Leverage Bagging | ✓ | | ✓ | | ✓ | ✓ | | | ✓ | [18] |
| Multi-output Learner | ✓ | ✓ | ✓ | ✓ | * | ✓ | ✓ | ✓ | ✓ | [20] |
| Classifier Chains | ✓ | | ✓ | ✓ | * | | ✓ | ✓ | ✓ | [105] |
| Regressor Chains | | ✓ | ✓ | ✓ | * | | ✓ | ✓ | ✓ | [105] |
| SGD | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | [20] |
| Naive Bayes | ✓ | | ✓ | | | ✓ | ✓ | ✓ | ✓ | [20] |
| Multi Layer Perceptron | ✓ | ✓ | ✓ | | | | ✓ | ✓ | ✓ | [20] |
| ADWIN | | | | | ✓ | ✓ | | | ✓ | [13] |
| DDM | | | | | ✓ | ✓ | | | ✓ | [48] |
| EDDM | | | | | ✓ | ✓ | | | ✓ | [8] |
| Page Hinkley | | | | | ✓ | ✓ | | | ✓ | [94] |

* Depending on the base learner.

† We have only listed incremental methods for data-streams; MEKA and scikit-learn have many other batch-learning models available. MEKA in particular, has many problem-transformation methods which may be incremental depending on the base learner (it is able to use those from the MOA framework).

Multi-label, Regression and concept-drift. Available evaluators correspond to prequential and hold-out evaluations, both supporting multiple performance metrics for *classification* (accuracy, kappa coefficient, kappa t, kappa m), *multi-output classification* (Hamming score, Hamming loss, exact match, Jaccard index), *regression* (mean squared error, mean absolute error) and *multi-output regression* (average mean squared error, average mean absolute error, average root mean squared error). Learning methods and change detectors are listed in Table 7.1. This table also serves to outline the position of scikit-multiflow with respect to other open source frameworks.

## 7.1 ARCHITECTURE

The base class in scikit-multiflow is `StreamModel` which contains the following abstract methods to be implemented by its subclasses:

- `fit()` – Trains a model in a batch fashion. Works as an interface to batch methods that implement a `fit()` function such as scikit-learn methods.
- `partial_fit()` – Incrementally trains a stream model.
- `predict()` – Predicts the target's value in supervised learning methods.
- `predict_proba()` – Calculates per-class probabilities in classification problems.

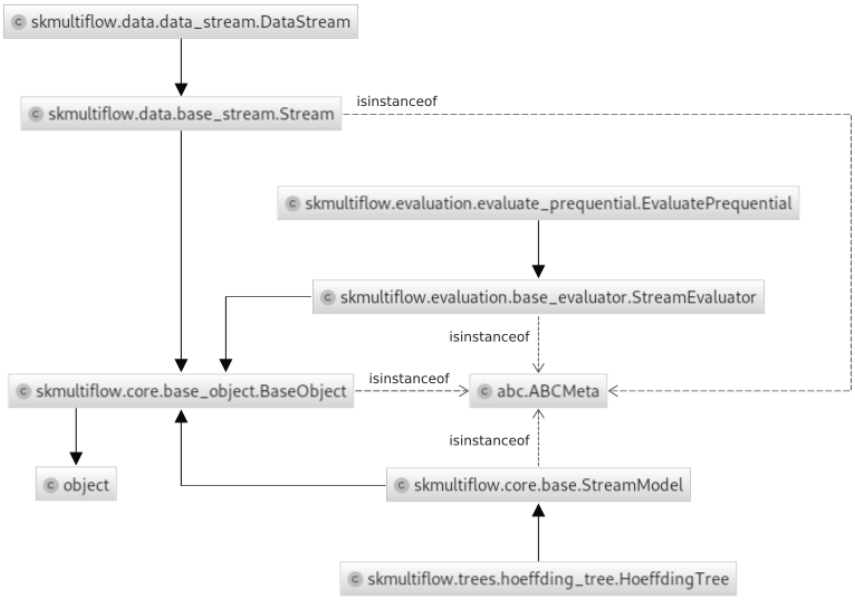



**Figure 7.1**: Class hierarchy within scikit-multiflow. This example shows instances of a `StreamModel` (HoeffdingTree), a `Stream` (DataStream) and a `StreamEvaluator` (EvaluatePrequenctial).

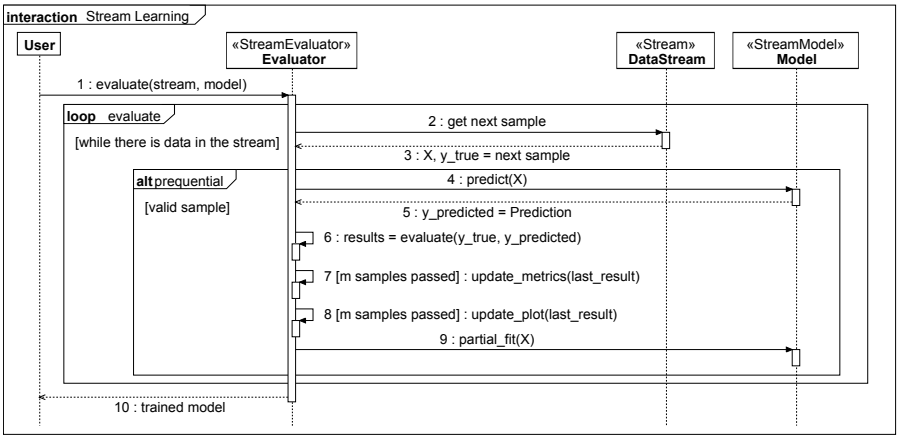



**Figure 7.2:** Training and testing a stream model using scikit-multiflow. This sequence corresponds to prequential evaluation.

A `StreamModel` object interacts with two other objects: a `Stream` object and (optionally) a `StreamEvaluator` object. The diagram in Figure 7.1 shows an example of the class hierarchy within scikit-multiflow. The `Stream` object provides a continuous flow of data on request. The `StreamEvaluator` performs multiple tasks: queries the stream for data, trains and tests the model on the incoming data and continuously tracks the model's performance. The sequence to train a stream model and track its performance using prequential evaluation in scikit-multiflow is outlined in Figure 7.2.

## 7.2 IN A NUTSHELL

Here we provide a quick overview of different elements of scikit-multiflow and how they can be used in the field of stream learning research. In agreement with the scope of this thesis, we only focus on stream learning classification and concept drift detection.

### 7.2.1 Data Streams

Within scikit-multiflow, data is represented by the `Stream` class. The most important functionality of the `Stream` class is to provide new samples of data on demand. Stream *generators* are a cheap source of data, where samples are generated on demand to avoid storing data physically. There are multiple stream generators in scikit-multiflow and all of them work in a similar way. The following code snippet shows how to use a generator within scikit-multiflow.

```
from skmultiflow.data import AGRAWALGenerator

# 1. Instantiate the stream generator
generator = AGRAWALGenerator()
```

```
generator.prepare_for_use()

# 2. Get data from the stream
X, y = generator.next_sample()
print(X.shape, y.shape)
>>> (1, 9) (1,)

X, y = generator.next_sample(1000)
# Inspect the shape of X and y
print("X: {}, y:{}".format(X.shape, y.shape))
>>> X: (1000, 9), y: (1000,)

# 3. Check if the stream has more data
generator.has_more_samples()
>>> True

# 4. Restart the stream
generator.restart()
```

### 7.2.2 Stream Learning Experiments

In this example, we show how to train a classifier (Hoeffding Tree [35]) on a stream (Waveform) and will measure its performance using prequential evaluation:

1. Instantiate a stream object. The WaveformGenerator generates by default samples with 21 numeric attributes and 3 classes, based on a random differentiation of some base waveforms.
2. Instantiate the Hoeffding Tree classifier. We will use the default parameters.
3. Setup the evaluator, we will use the EvaluatePrequential class. Since we are using a generator, we need to specify the max number of samples to evaluate, `max_samples=2000`.
4. Run the evaluation. By calling `evaluate()` method, we pass control to the evaluator, which will perform the following sub-tasks:
   a) Check if there are samples in the stream.
   b) Pass the next sample to the classifier.
   c) Test the classifier (using `predict()`)
   d) Update the classifier (using `partial_fit()`)

The corresponding code snippet to train and test a stream model is shown bellow:

```
from skmultiflow.data import WaveformGenerator
from skmultiflow.trees import HoeffdingTree
from skmultiflow.evaluation import EvaluatePrequential
```

```
    # 1. Create a stream
    stream = WaveformGenerator()
    stream.prepare_for_use()

    # 2. Instantiate the HoeffdingTree classifier
    ht = HoeffdingTree()

    # 3. Setup the evaluator
    evaluator = EvaluatePrequential(max_samples=20000)

    # 4. Run evaluation
    evaluator.evaluate(stream=stream,
                       model=ht,
                       model_names=['HT'])
```

scikit-multiflow will report the progress of the evaluation and will return the evaluation summary:

```
Prequential Evaluation
Evaluating 1 target(s).
Pre-training on 200 sample(s).
Evaluating...
 ################### [100%] [20.80s]
Processed samples: 20000
Mean performance:
HT - Accuracy     : 0.7941
HT - Kappa        : 0.6915
```

Additionally, the evaluator objects provides methods to interact with the metrics from the evaluation when it is finished.

The evaluators in scikit-multiflow support multiple models per evaluation, streamlining the comparison of methods, a common task in research and real-world applications. The following snippet shows this functionality where the goal is to compare a Hoeffding Tree model vs a Hoeffding Adaptive Tree [14] model.

```
from skmultiflow.data import FileStream
from skmultiflow.evaluation import EvaluatePrequential
from skmultiflow.trees import HoeffdingTree
from skmultiflow.trees import HAT

# 1. Load stream data from a file
stream = FileStream("path/elec.csv")
stream.prepare_for_use()

# 2. Create a list of classifiers to compare, in this case
    Hoeffding Tree and Hoeffding Adaptive Tree
classifiers = [HoeffdingTree(), HAT()]

# 3. Setup the evaluator, we will use prequential evaluation
eval = EvaluatePrequential(show_plot=True,
                           metrics=['accuracy',
```

```
                                    'kappa',
                                    'running_time',
                                    'model_size'])

# 4. Run the evaluation
eval.evaluate(stream=stream,
              model=classifiers,
              model_names=['HT', 'HAT'])
```

In this case, since we use the `show_plot=True` option, in addition to the evaluation summary we get a dynamic plot of the evaluation to visualize the behavior of both models over time. The plot for this example is shown in Figure 7.3.

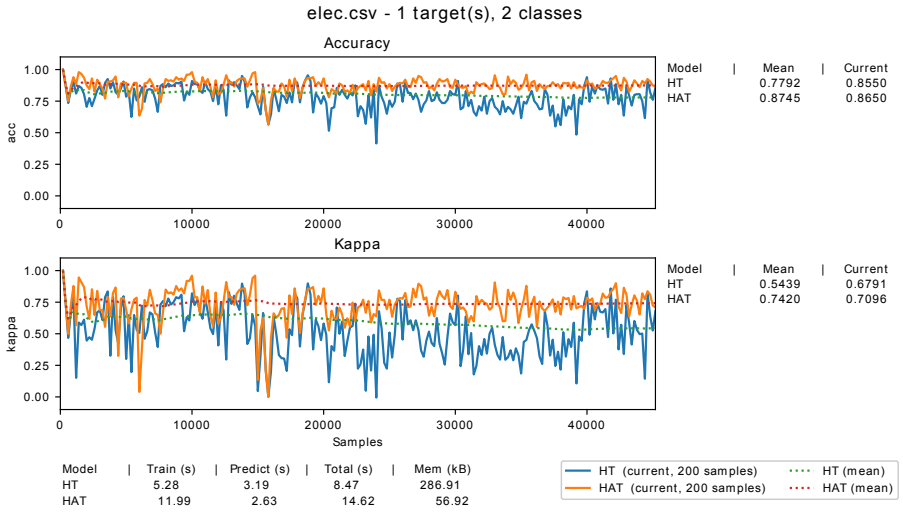



**Figure 7.3:** Comparing multiple models on scikit-multiflow.

### 7.2.3 Concept Drift Detection

scikit-multiflow provides implementations of multiple state-of-the-art drift detectors, Table 7.1. The drift detection methods in scikit-multiflow provide similar functionality, an `add_element()` method to add new observations and a `detected_change()` method to query the flag that indicates if a drift has been detected or not. Here we show how to use ADWIN [13]. For this example we use a stream (`drift_stream`) where the first half contains a sequence corresponding to a normal distribution of integers from 0 to 1. From index 999 to 1999 the sequence is a normal distribution of integers from 0 to 7.

```
from skmultiflow.drift_detection import ADWIN

# 1. Instantiate the drift detector, in this case ADWIN
adwin = ADWIN()

# 2. Run the detection test
for i in range(drift_stream.size):
```

```
    adwin.add_element(drift_stream[i])
    if adwin.detected_change():
        print('Change detected at index {}'.format(i))
>>> Change detected at index 1023
>>> Change detected at index 1055
>>> Change detected at index 1087
>>> Change detected at index 1151
```

For this example, we see that the detector sets the drift flag to `True` in the transition between the two distributions. During this transition period the ADWIN drift detector flags 4 changes until the data distribution is stable.

## 7.3 DEVELOPMENT

The scikit-multiflow package is distributed under the BSD License. Development follows the FOSS principles and encompasses:

- A webpage, https://scikit-multiflow.github.io/, including documentation and user guide. Both, documentation and user guide, are living documents that are continuously updated to reflect the current stage of scikit-multiflow.
- Version control via git. The source code of the package is publicly available on Github at https://github.com/scikit-multiflow/scikit-multiflow
- Package deployment and software quality are enforced via continuous integration and functional testing, https://travis-ci.org/scikit-multiflow/scikit-multiflow

# Part III

# CONCLUSIONS

# 8 FUTURE WORK AND CONCLUSIONS

Big data is undoubtedly changing the way in which humanity generates, processes and understands data. The application of big data across multiple fields is already impacting millions of lives. At the front line of the digital revolution we find the field of machine learning. In this thesis we investigated the overlapping region between two learning paradigms: batch and stream. In particular, we focused on the classification problem, an instance of supervised learning. Given that research is a never ending task in the advancement of humanity, we list further research directions that stem from our work. Subsequently, we present the final remarks of this thesis.

## 8.1 FUTURE DIRECTIONS

In the following, we highlight promising research directions and provide recommendations for future studies.

### 8.1.1 Over-indebtedness Prediction

In the case of over-indebtedness prediction, it is plausible that increasing the warning time from 6 to 9∼12 months would improve the chances of recovery. Additionally, quantifying the associated cost to correct/incorrect predictions would open the path for cost aware learning techniques. A promising direction is to handle recurrent trends in financial data which could improve the robustness of the over-indebtedness warning mechanism.

### 8.1.2 Missing Data Imputation

A natural extension of our imputation method is its application to data streams. One possibility is keeping concurrent models for each feature. However, an important issue are the constraints in stream learning in terms of resources. Thus, we suggest that further research should focus on the selection of training data within the cascade. For example, accounting for the correlation between attributes could reduce the computational burden from unrelated attributes while maintaining data interpretability. Similarly, adding an active strategy to control the number of samples used for training within the cascade could result in a more efficient approach without sacrificing imputation performance.

### 8.1.3 Fast and Slow Learning

We believe that our findings on the feasibility of a unified batch-stream framework outline an area of opportunity to further investigate ways to exploit the collaboration between different type of learners. Using an ensemble instead of single batch models to improve robustness and a pool of models to keep old models which can be useful in the presence of recurrent concepts. What's more, investigating the integration of batch and stream models rises interesting research questions, e. g. *can we use the predictions from a batch model to boost the performance of a stream model and vice versa?*

### 8.1.4 Adaptive XGBoost

Finally, regarding our adaptation of XGBoost for evolving data streams. Further studies which focus on measuring the contribution of individual weak learners, could enable the improvement of the strategy to update (or not) the ensemble. More broadly, research onto automatic parameter tuning could enable research on long-term performance of stream models. Finally, an interesting topic for future work is an instance-incremental strategy to update individual weak learners and consequently the ensemble. We believe that our work will serve as base for future studies on boosting for mining evolving data streams.

## 8.2 FINAL REMARKS

We presented OVI1, a systematic approach to over-indebtedness risk prediction, a real-world problem with significant implications at personal, social end economic levels. OVI1 is a data-driven warning mechanism for *inter-bank* application within the *Groupe BPCE*, the second largest banking institution in France. We propose a multi-metric criteria to account for the costs associated to correct/incorrect classifications given the extreme class-imbalance in the data. Two versions of the framework are defined: OVI1-BATCH and OVI1-STREAM. To the extent of our knowledge, our work is the first to cast over-indebtedness prediction as a stream learning problem, overcoming the limitations of batch models which become obsolete over time as financial markets evolve. Tests results show that OVI1-BATCH and OVI1-STREAM outperform the current mechanism used by *Groupe BPCE* for both single and multi-metric criteria.

To address the problem of missing data in classification, we proposed CIM, an effective and scalable imputation method. CIM imputes both numerical and nominal data and mitigates the impact of 'missing at random' and 'missing completely at random' data on binary

and multi-class classification. Test results show that CIM performs well over a wide range of missing values ratios and does not require parameter tuning to achieve optimal performance. CIM is scalable to large data sets, including highly dimensional data sets, a limitation of established methods such as EMI and kNNI. Additionally, CIM always succeeds in generating an imputed data set, something that is not guaranteed by EMI. We also presented a multi-label learning version of CIM able to impute multiple features at a time. Our works is the first to approach hybrid (nominal and numerical) data types in multi-label learning.

We also introduced Fast and Slow Learning, a unified framework that uses fast (stream) and slow (batch) models concurrently. Our research shows that considering learning as a continuous task, and contrary to popular belief, a *symbiosis* of batch and stream learning is feasible and can be effective. We showed the applicability of the proposed framework for classification of streams with concept drift. Test results on synthetic and real-world data confirm that the Fast and Slow Classifier leverages stream and batch methods by combining their strengths: a Fast model adapts to changes in the data and provides predictions based on current concepts, whereas a Slow model generates complex models considering wider data distributions. Additionally, a drift detection mechanism serves to trigger automatically the training of new Slow models that are consistent with the new concept. Furthermore, the Fast and Slow Classifier allows the introduction of batch methods, other than learning algorithms, into the stream learning setting. For example, if the stream has missing data, CIM can be added in a straightforward manner by creating a batch pipeline [buffering + CIM + training].

Similarly, we proposed AXGB, an adaptation of the extreme gradient boosting algorithm for evolving data streams. The core idea is the incremental creation and update of the ensemble, i.e. weak learners are trained on mini-batches of data and then added to the ensemble. We study variations of the proposed method by considering two main factors: concept drift awareness and the strategy to update the ensemble. We test AXGB against instance-incremental and batch-incremental methods on synthetic and real-world data. Additionally, we consider a simple batch-incremental approach (BXGB) where members of the ensemble are full XGB batch models trained on consecutive mini-batches. From our tests, we conclude that $\text{AXGB}_{[r]}$ (no concept drift awareness and model replacement within the ensemble) represents the best compromise in terms of performance, training time and memory usage. Another noteworthy finding from our experiments is the good predictive performance of BXGB after parameter tuning. If resource consumption is a secondary consideration, this approach may be a worthwhile candidate in practical data stream mining scenarios, particularly considering that our parameter tuning

experiments did not investigate optimizing the size of the boosted ensemble for each mini-batch in BXGB. Overall, despite the limitations of mini-batch-based data stream mining, and its drawbacks compared to instance-incremental methods, it appears that XGB-based techniques are promising candidates for data stream applications. In a similar way, we believe AXGB might be an interesting alternative to XGB for certain batch applications given its efficient management of resources and adaptability.

Finally, we presented scikit-multiflow, a data stream framework in Python. scikit-multiflow contributes to the democratization of stream learning by extending the array of open source software available to researchers and public in general. scikit-multiflow is primarily a platform for stream learning research, the Fast and Slow Classifier and AXGB are implemented and evaluated on scikit-multiflow. We are positive that this open source tool will benefit the research community and will push forward the field of stream learning.

Part IV

# APPENDICES

# A CASCADE IMPUTATION

Plots in Figure A.1 provide insights on the distribution of missing values per missingness mechanism.

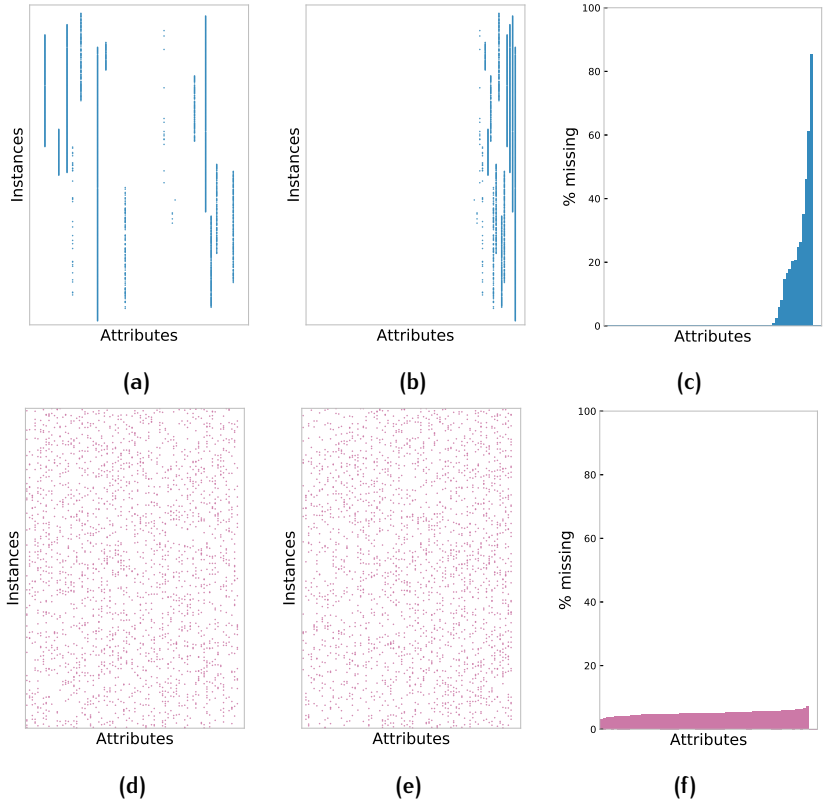



**Figure A.1:** Difference between MAR (top) and MCAR (bottom) data distributions. Plots correspond to the same data set (Music) with 5% missing values. Figures A.1a and A.1d show original position of missing values (dots). Figures A.1b and A.1e show the updated positions after initial column-wise sorting performed by CIM. Figures A.1c and A.1f are missing values histograms after sorting.

Tests results for Imputation-Classification tests, Chapter 4 Section 4.3, are shown in Figure A.2 for Logistic Regression and in Figure A.3 for Random Forest. Plots show performance based on AUROC for binary data sets and F1-Score for multi-class data sets. Detailed information is available in Tables A.1, A.2, A.3 and A.4.

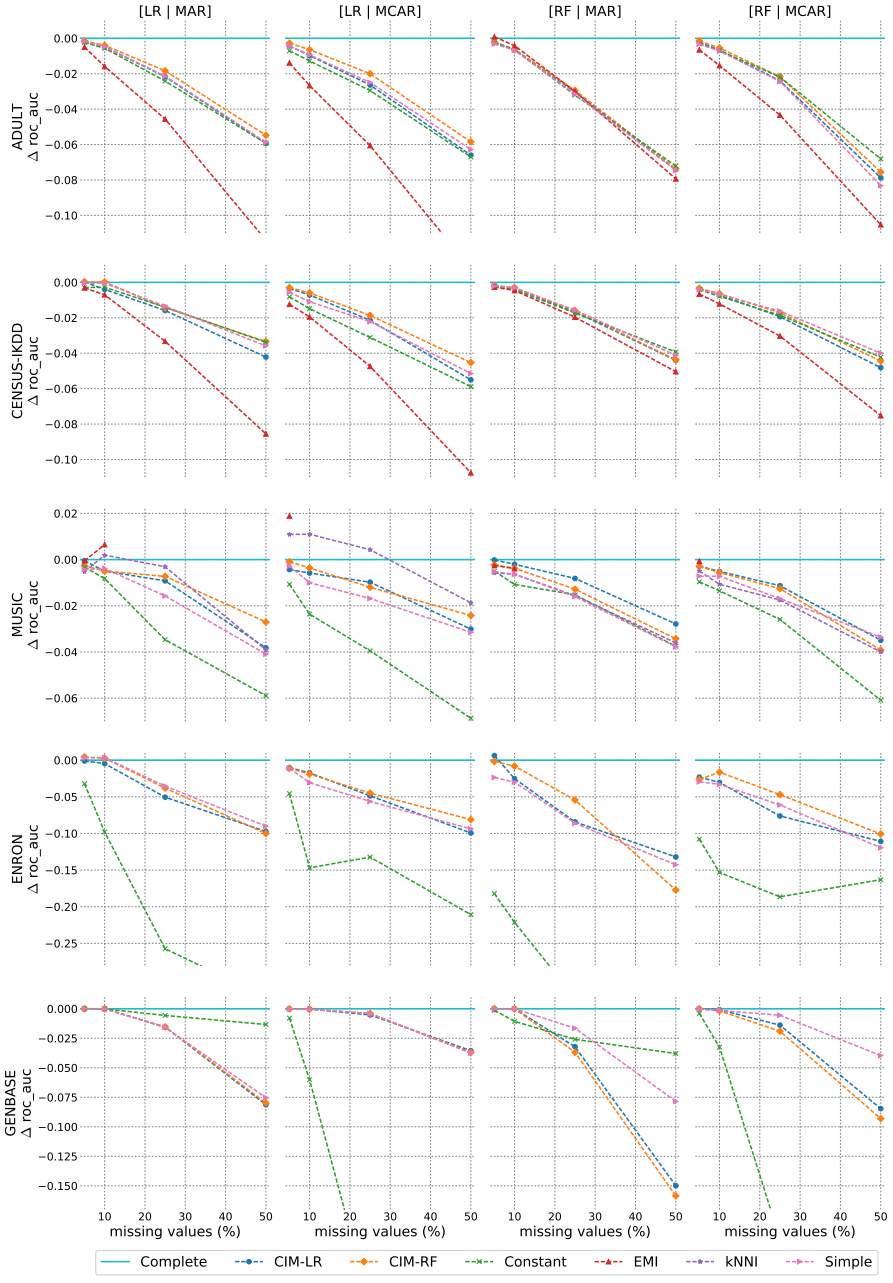



**Figure A.2:** Performance of classification models based on imputed data vs baseline. [1/2]

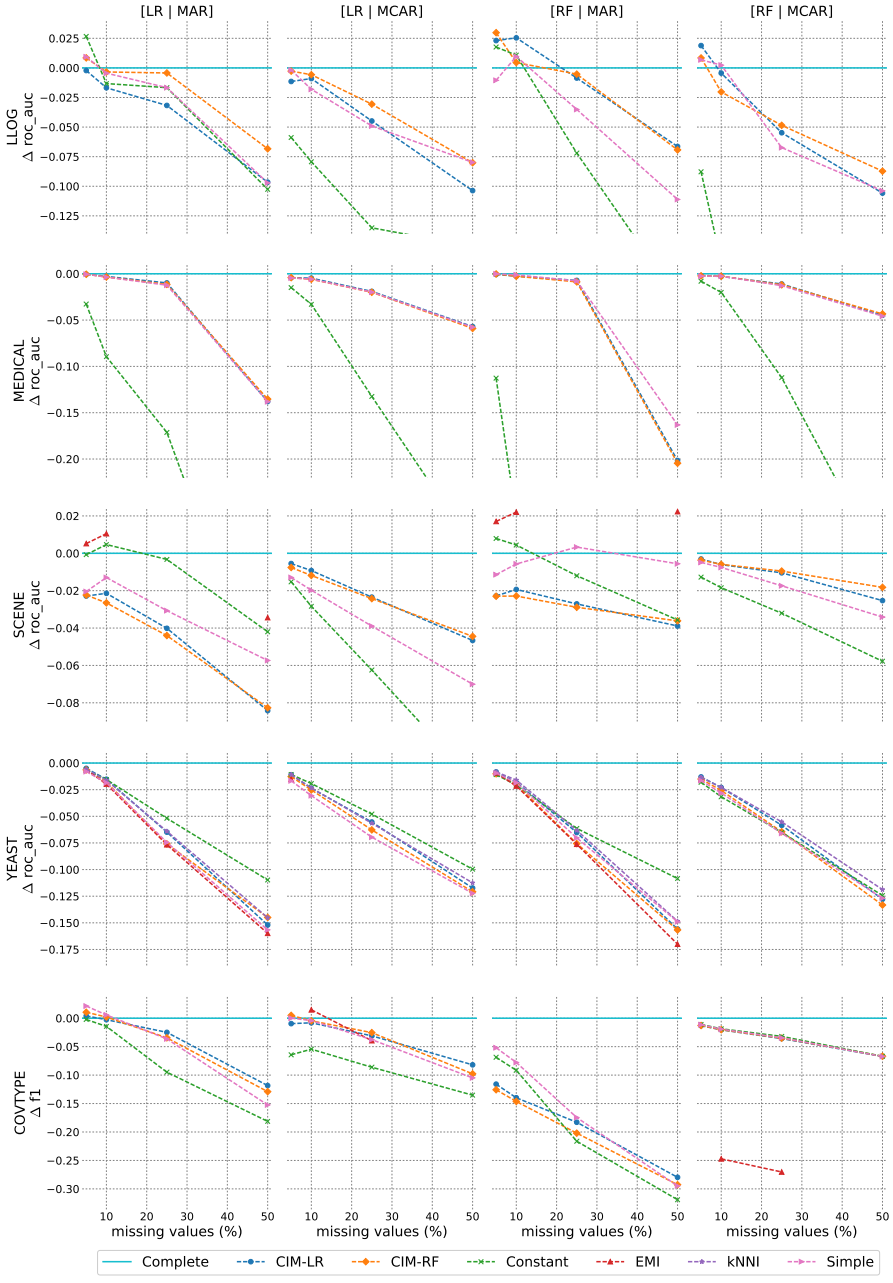



**Figure A.3:** Performance of classification models based on imputed data vs baseline. [2/2]

**Table A.1**: Classification results. Logistic Regression on MAR data.

| Data set | Missing | CIM-LR | CIM-RF | Constant | EMI | kNNI | Simple |
|---|---|---|---|---|---|---|---|
| Adult | 5 % | 90.45 | **90.48** | 90.39 | 90.13 | - | 90.46 |
| | 10 % | 90.14 | **90.23** | 90.07 | 89.04 | - | 90.15 |
| | 25 % | 88.46 | **88.79** | 88.22 | 86.07 | - | 88.49 |
| | 50 % | 84.70 | **85.15** | 84.69 | 79.23 | - | 84.76 |
| Census-IKDD | 5 % | 91.67 | **91.68** | 91.35 | 91.33 | - | 91.61 |
| | 10 % | 91.24 | **91.67** | 91.37 | 90.94 | - | 91.59 |
| | 25 % | 90.05 | 90.20 | 90.24 | 88.31 | - | **90.29** |
| | 50 % | 87.42 | **88.30** | 88.26 | 83.08 | - | 88.05 |
| Music | 5 % | 84.87 | 84.65 | 84.60 | **84.87** | 84.40 | 84.49 |
| | 10 % | 84.42 | 84.42 | 84.10 | **85.57** | 85.11 | 84.54 |
| | 25 % | 84.00 | 84.19 | 81.45 | - | **84.61** | 83.35 |
| | 50 % | 81.10 | **82.22** | 79.03 | - | 81.01 | 80.83 |
| Enron | 5 % | 69.02 | **69.56** | 65.92 | - | - | 69.46 |
| | 10 % | 68.66 | 69.39 | 59.31 | - | - | **69.46** |
| | 25 % | 64.07 | 65.31 | 43.39 | - | - | **65.59** |
| | 50 % | 59.40 | 59.12 | 36.77 | - | - | **60.12** |
| Genbase | 5 % | **100.00** | **100.00** | **100.00** | - | - | **100.00** |
| | 10 % | **100.00** | **100.00** | 99.99 | - | - | **100.00** |
| | 25 % | 98.45 | 98.46 | **99.43** | - | - | 98.42 |
| | 50 % | 91.87 | 92.02 | **98.66** | - | - | 92.46 |
| Llog | 5 % | 81.90 | 82.97 | **84.79** | - | - | 83.06 |
| | 10 % | 80.44 | **81.78** | 80.80 | - | - | 81.68 |
| | 25 % | 78.95 | **81.70** | 80.45 | - | - | 80.45 |
| | 50 % | 72.47 | **75.29** | 71.86 | - | - | 72.41 |
| Medical | 5 % | 99.69 | **99.71** | 96.49 | - | - | **99.71** |
| | 10 % | **99.47** | 99.41 | 90.81 | - | - | 99.40 |
| | 25 % | **98.78** | 98.62 | 82.64 | - | - | 98.53 |
| | 50 % | 85.98 | **86.23** | 52.92 | - | - | 85.95 |
| Scene | 5 % | 95.73 | 95.79 | 97.95 | **98.54** | - | 95.97 |
| | 10 % | 95.88 | 95.37 | 98.49 | **99.07** | - | 96.72 |
| | 25 % | 94.01 | 93.62 | **97.69** | - | - | 94.95 |
| | 50 % | 89.60 | 89.75 | 93.82 | **94.58** | - | 92.29 |
| Yeast | 5 % | **94.00** | 93.86 | 93.85 | 93.87 | 93.93 | 93.69 |
| | 10 % | **92.97** | 92.71 | 92.95 | 92.53 | 92.89 | 92.65 |
| | 25 % | 88.01 | 86.90 | **89.31** | 86.83 | 88.10 | 87.05 |
| | 50 % | 79.29 | 80.00 | **83.52** | 78.51 | 79.98 | 78.84 |
| Covtype | 5 % | 44.95 | 45.59 | 44.32 | - | - | **46.69** |
| | 10 % | 44.28 | 44.75 | 43.11 | - | - | **45.11** |
| | 25 % | **42.06** | 41.10 | 35.06 | - | - | 40.91 |
| | 50 % | **32.70** | 31.64 | 26.41 | - | - | 29.30 |

**Note:** Binary classification results using ROC AUC except for Covtype which corresponds to multi-class classification using F1-Score.

**Table A.2:** Classification results. Logistic Regression on MCAR data.

| Data set | Missing | CIM-LR | CIM-RF | Constant | EMI | kNNI | Simple |
|---|---|---|---|---|---|---|---|
| Adult | 5 % | 90.20 | **90.35** | 89.93 | 89.24 | - | 90.19 |
| | 10 % | 89.66 | **89.99** | 89.35 | 87.96 | - | 89.69 |
| | 25 % | 87.99 | **88.63** | 87.68 | 84.57 | - | 88.13 |
| | 50 % | 84.03 | **84.78** | 83.93 | 77.63 | - | 84.35 |
| Census-IKDD | 5 % | 91.29 | **91.35** | 90.83 | 90.40 | - | 91.08 |
| | 10 % | 90.93 | **91.05** | 90.16 | 89.69 | - | 90.55 |
| | 25 % | 89.50 | **89.77** | 88.53 | 86.90 | - | 89.43 |
| | 50 % | 86.14 | **87.11** | 85.76 | 80.89 | - | 86.49 |
| Music | 5 % | 84.49 | 84.83 | 83.85 | **86.81** | 86.01 | 84.64 |
| | 10 % | 84.34 | 84.56 | 82.56 | - | **86.02** | 83.92 |
| | 25 % | 83.94 | 83.72 | 80.97 | - | **85.35** | 83.24 |
| | 50 % | 81.91 | 82.50 | 78.05 | - | **83.05** | 81.77 |
| Enron | 5 % | **68.11** | 68.01 | 64.56 | - | - | 68.01 |
| | 10 % | **67.39** | 67.22 | 54.43 | - | - | 66.06 |
| | 25 % | 64.26 | **64.63** | 55.89 | - | - | 63.49 |
| | 50 % | 59.19 | **61.01** | 48.05 | - | - | 59.75 |
| Genbase | 5 % | **99.99** | **99.99** | 99.21 | - | - | **99.99** |
| | 10 % | **99.97** | 99.96 | 93.99 | - | - | **99.97** |
| | 25 % | 99.48 | **99.61** | 74.89 | - | - | 99.59 |
| | 50 % | **96.45** | 96.31 | 58.48 | - | - | 96.24 |
| Llog | 5 % | 80.98 | 81.85 | 76.24 | - | - | **81.94** |
| | 10 % | 81.22 | **81.55** | 74.19 | - | - | 80.32 |
| | 25 % | 77.64 | **79.06** | 68.61 | - | - | 77.22 |
| | 50 % | 71.76 | 74.11 | 67.03 | - | - | **74.14** |
| Medical | 5 % | **99.34** | **99.34** | 98.27 | - | - | 99.31 |
| | 10 % | **99.29** | 99.17 | 96.46 | - | - | 99.19 |
| | 25 % | **97.86** | 97.77 | 86.52 | - | - | 97.78 |
| | 50 % | **94.06** | 93.86 | 70.36 | - | - | 94.00 |
| Scene | 5 % | **97.47** | 97.26 | 96.50 | - | - | 96.73 |
| | 10 % | **97.10** | 96.83 | 95.18 | - | - | 96.04 |
| | 25 % | **95.67** | 95.59 | 91.77 | - | - | 94.12 |
| | 50 % | 93.36 | **93.57** | 85.91 | - | - | 91.02 |
| Yeast | 5 % | 93.40 | 93.22 | **93.42** | 92.99 | 93.41 | 92.83 |
| | 10 % | 92.13 | 91.97 | **92.57** | - | 92.16 | 91.44 |
| | 25 % | 88.96 | 88.22 | **89.70** | - | 88.85 | 87.54 |
| | 50 % | 82.76 | 82.41 | **84.53** | - | 83.25 | 82.27 |
| Covtype | 5 % | 43.59 | **45.04** | 38.12 | - | - | 44.55 |
| | 10 % | 43.74 | 44.08 | 39.10 | **46.02** | - | 44.13 |
| | 25 % | 41.45 | **42.01** | 35.93 | 40.60 | - | 40.77 |
| | 50 % | **36.35** | 34.77 | 31.02 | - | - | 34.05 |

**Note:** Binary classification results using ROC AUC except for Covtype which corresponds to multi-class classification using F1-Score.

**Table A.3**: Classification results. Random Forest on MAR data.

| Data set | Missing | CIM-LR | CIM-RF | Constant | EMI | kNNI | Simple |
|---|---|---|---|---|---|---|---|
| Adult | 5 % | 94.05 | **94.06** | 94.02 | 93.97 | - | 94.03 |
| | 10 % | 93.91 | **93.95** | 93.83 | 93.78 | - | 93.92 |
| | 25 % | 92.58 | 92.64 | 92.51 | 92.27 | - | **92.68** |
| | 50 % | 89.84 | 89.87 | **90.31** | 89.20 | - | 90.12 |
| Census-IKDD | 5 % | **87.93** | 87.72 | 87.45 | 87.70 | 87.37 | 87.42 |
| | 10 % | **87.75** | 87.57 | 86.87 | 87.56 | 87.32 | 87.32 |
| | 25 % | **87.13** | 86.67 | 86.42 | - | 86.39 | 86.35 |
| | 50 % | **85.16** | 84.52 | 84.20 | - | 84.34 | 84.17 |
| Music | 5 % | **68.07** | 67.26 | 49.23 | - | - | 65.08 |
| | 10 % | 64.92 | **66.63** | 45.31 | - | - | 64.44 |
| | 25 % | 59.02 | **62.01** | 35.02 | - | - | 58.84 |
| | 50 % | **54.23** | 49.71 | 28.95 | - | - | 53.17 |
| Enron | 5 % | **68.07** | 67.26 | 49.23 | - | - | 65.08 |
| | 10 % | 64.92 | **66.63** | 45.31 | - | - | 64.44 |
| | 25 % | 59.02 | **62.01** | 35.02 | - | - | 58.84 |
| | 50 % | **54.23** | 49.71 | 28.95 | - | - | 53.17 |
| Genbase | 5 % | **100.00** | **100.00** | 99.88 | - | - | **100.00** |
| | 10 % | **100.00** | **100.00** | 98.93 | - | - | **100.00** |
| | 25 % | 96.78 | 96.31 | 97.41 | - | - | **98.35** |
| | 50 % | 85.01 | 84.15 | **96.21** | - | - | 92.16 |
| Llog | 5 % | 85.23 | **85.89** | 84.68 | - | - | 81.87 |
| | 10 % | **85.46** | 83.37 | 84.00 | - | - | 83.89 |
| | 25 % | 82.06 | **82.39** | 75.72 | - | - | 79.41 |
| | 50 % | **76.28** | 75.99 | 63.43 | - | - | 71.81 |
| Medical | 5 % | 99.54 | 99.59 | 88.34 | - | - | **99.60** |
| | 10 % | 99.35 | 99.34 | 73.14 | - | - | **99.49** |
| | 25 % | **98.90** | 98.74 | 58.48 | - | - | 98.84 |
| | 50 % | 79.45 | 79.16 | 38.64 | - | - | **83.30** |
| Scene | 5 % | 93.54 | 93.52 | 96.61 | **97.52** | - | 94.68 |
| | 10 % | 93.88 | 93.53 | 96.26 | **98.02** | - | 95.25 |
| | 25 % | 93.11 | 92.93 | 94.62 | - | - | **96.15** |
| | 50 % | 91.93 | 92.20 | 92.25 | **98.05** | - | 95.26 |
| Yeast | 5 % | **94.98** | 94.80 | 94.72 | 94.90 | **94.98** | 94.87 |
| | 10 % | 94.00 | 93.82 | 93.75 | 93.66 | **94.22** | 93.96 |
| | 25 % | 89.29 | 88.25 | **89.68** | 88.19 | 89.50 | 88.79 |
| | 50 % | 80.23 | 80.15 | **84.97** | 78.81 | 80.97 | 80.91 |
| Covtype | 5 % | 51.13 | 50.15 | 55.85 | - | - | **57.53** |
| | 10 % | 48.74 | 48.10 | 53.57 | - | - | **54.95** |
| | 25 % | 44.43 | 42.52 | 41.09 | - | - | **45.27** |
| | 50 % | **34.76** | 33.46 | 30.82 | - | - | 33.24 |

**Note:** Binary classification results using ROC AUC except for Covtype which corresponds to multi-class classification using F1-Score.

**Table A.4**: Classification results. Random Forest on MCAR data.

| Data set | Missing | CIM-LR | CIM-RF | Constant | EMI | kNNI | Simple |
|---|---|---|---|---|---|---|---|
| Adult | 5 % | 90.51 | **90.59** | 90.42 | 90.08 | - | 90.40 |
| | 10 % | 90.07 | **90.19** | 90.04 | 89.20 | - | 90.02 |
| | 25 % | 88.34 | **88.57** | 88.55 | 86.40 | - | 88.30 |
| | 50 % | 82.86 | 83.18 | **83.93** | 80.21 | - | 82.42 |
| Census-IKDD | 5 % | 93.89 | **93.91** | 93.83 | 93.58 | - | 93.84 |
| | 10 % | 93.54 | **93.61** | 93.45 | 93.03 | - | 93.56 |
| | 25 % | 92.30 | 92.52 | 92.38 | 91.21 | - | **92.62** |
| | 50 % | 89.42 | 89.80 | 90.02 | 86.72 | - | **90.24** |
| Music | 5 % | 87.67 | 87.66 | 86.99 | **87.88** | 87.46 | 87.23 |
| | 10 % | **87.44** | 87.39 | 86.60 | - | 86.88 | 87.23 |
| | 25 % | **86.82** | 86.69 | 85.36 | - | 86.20 | 86.26 |
| | 50 % | 84.46 | 84.04 | 81.86 | - | 83.95 | **84.59** |
| Enron | 5 % | **65.13** | 64.81 | 56.66 | - | - | 64.48 |
| | 10 % | 64.40 | **65.80** | 52.12 | - | - | 64.15 |
| | 25 % | 59.82 | **62.74** | 48.79 | - | - | 61.34 |
| | 50 % | 56.36 | **57.36** | 51.13 | - | - | 55.50 |
| Genbase | 5 % | **99.99** | **99.99** | 99.60 | - | - | 99.98 |
| | 10 % | **99.92** | 99.83 | 96.75 | - | - | 99.86 |
| | 25 % | 98.61 | 98.10 | 80.49 | - | - | **99.45** |
| | 50 % | 91.53 | 90.69 | 61.19 | - | - | **96.03** |
| Llog | 5 % | **84.81** | 83.76 | 74.15 | - | - | 83.60 |
| | 10 % | 82.48 | 80.89 | 66.50 | - | - | **83.16** |
| | 25 % | 77.44 | **78.07** | 60.14 | - | - | 76.18 |
| | 50 % | 72.34 | **74.20** | 57.88 | - | - | 72.53 |
| Medical | 5 % | **99.41** | 99.39 | 98.80 | - | - | 99.36 |
| | 10 % | 99.32 | **99.35** | 97.60 | - | - | **99.35** |
| | 25 % | **98.51** | 98.42 | 88.39 | - | - | 98.33 |
| | 50 % | 95.13 | **95.28** | 68.05 | - | - | 95.04 |
| Scene | 5 % | **95.51** | 95.45 | 94.54 | - | - | 95.33 |
| | 10 % | 95.21 | **95.23** | 93.98 | - | - | 95.06 |
| | 25 % | 94.77 | **94.87** | 92.61 | - | - | 94.08 |
| | 50 % | 93.29 | **94.00** | 90.04 | - | - | 92.41 |
| Yeast | 5 % | **94.51** | 94.31 | 94.00 | 94.28 | 94.47 | 94.17 |
| | 10 % | 93.51 | 93.24 | 92.65 | - | **93.52** | 93.00 |
| | 25 % | 89.97 | 89.36 | 89.30 | - | **90.29** | 89.22 |
| | 50 % | 83.03 | 82.48 | 83.38 | - | **83.93** | 83.13 |
| Covtype | 5 % | 61.41 | 61.44 | **61.63** | - | - | 61.58 |
| | 10 % | 60.68 | 60.70 | **60.86** | 38.00 | - | 60.80 |
| | 25 % | 59.14 | 59.22 | **59.53** | 35.69 | - | 59.28 |
| | 50 % | 56.00 | **56.04** | 56.03 | - | - | 56.00 |

**Note:** Binary classification results using ROC AUC except for Covtype which corresponds to multi-class classification using F1-Score.

# B RÉSUMÉ EN FRANÇAIS

Ces dernières années, nous avons assisté à une révolution numérique qui a radicalement changé la façon dont nous générons et consommons des données. En 2016, 90% des données mondiales ont été créées au cours des deux dernières années et on prévoit que d'ici à 2020 l'univers numérique atteindra 44 zettabytes[1] (44 000 milliards de gigabytes). Ce nouveau paradigme de données omniprésentes a eu une impact sur différents secteurs de la société, notamment les administrations publiques, les soins de santé, les banques et les loisirs, pour n'en nommer que quelques-uns. En raison de l'ampleur de son potentiel, les données ont été appelées "le pétrole de l'ère numérique"[2].

## B.1 DÉFIS ET OPPORTUNITÉS

L'apprentissage automatique n'est pas nouveau, il s'agit en réalité d'un domaine d'étude bien établi, doté d'un solide bagage scientifique et technique. Cependant, l'impact perturbateur du Big Data et les défis qu'il pose ont revigoré le monde de la recherche. En outre, cela a contribué à faire de l'apprentissage automatique un sujet d'intérêt majeur pour le grand public. Actuellement, l'une des axes de recherche les plus actives à l'ère des données omniprésentes est de faciliter la mise en place de mécanismes permettant l'apprentissage automatique à grande échelle. Dans ce contexte, il est essentiel que les méthodes d'apprentissage puissent suivre le rythme des données, non seulement en termes de volume, mais aussi de la vitesse à laquelle elles sont générées et traitées, afin d'être utiles à l'homme.

Traditionnellement, le pipeline minimal (collecte de données $\rightarrow$ estimation (entraînement) de modèle $\rightarrow$ prédiction) est appliqué à des lots ou batchs de données. Lorsque de nouvelles données deviennent disponibles, la même séquence est répétée et un nouveau modèle est généré. Cette approche, connue sous le nom de *apprentissage en batch*, s'est révélée efficace dans de nombreuses applications du monde réel. Cependant, cela représente des compromis importants lors de la résolution de problèmes liés au Big Data. Par exemple, si les données arrivent à grande vitesse, nous devons exécuter le pipeline complet encore et encore pour conserver une séquence de modèles cohérente avec les données récentes. Comme (en général) aucune connaissance n'est conservée, nous devons relancer l'ensemble du pipeline, ce qui peut

1 *The digital universe in 2020*, John Gantz et David Reinsel, IDC, février 2013.
2 *The world's most valuable resource is no longer oil, but data*, The Economist, mai 2017.

représenter un gaspillage considérable de ressources si les données ne présentent que de petites variations dans le temps. Il est également difficile de définir la quantité de données à utiliser ou le temps que nous sommes prêts à attendre tout en maximisant les chances de générer un modèle optimal.

L'équilibre entre l'investissement en ressources (temps, mémoire, coûts) et la qualité du modèle est un élément essentiel pour la viabilité d'un algorithme d'apprentissage dans des applications Big Data réelles. L'*apprentissage par flux* est le domaine de recherche émergent qui se concentre sur l'apprentissage en continu à partir de flux de données infinis. Un algorithme d'apprentissage par flux ne voit qu'une fois les données, modifie son état interne (modèle), puis traite le prochain échantillon de données en partant du principe que les anciens échantillons ne sont plus vus. L'apprentissage en continu met l'accent sur une utilisation efficace des ressources sans compromettre l'apprentissage. Dans cette thèse, nous nous concentrons sur la région de chevauchement entre l'apprentissage par batch et par flux.

## B.2 OPEN DATA SCIENCE

La science des données est un domaine interdisciplinaire émergent à l'ère numérique qui unit les statistiques, l'analyse des données et l'apprentissage automatique pour extraire des connaissances et des informations à partir des données. L'un des principaux contributeurs à l'adoption rapide de la science des données est le mouvement *Open Source*. La désignation "open source" est attribuée à quelque chose que des personnes, autres que l'auteur, peuvent modifier et partager parce que sa conception est accessible au public. Dans le contexte du développement de logiciel, il s'agit du méthode utilisé pour développer des programmes informatiques. La communauté de l'apprentissage automatique a été bénéficié d'un grand nombre de frameworks open source centrés sur de multiples sujets et plateformes (système d'exploitation et langage de programmation). Comme exemples des avantages de la recherche open source, nous pouvons identifier:

- **Recherche reproductible**, une partie essentielle du processus scientifique.
- **Développement plus rapide**, car les chercheurs peuvent se concentrer sur les éléments essentiels de leur travail sans se laisser détourner des détails techniques.
- **Favorise la collaboration**, en fournissant une plate-forme commune sur laquelle une communauté peut prospérer.

- **Démocratisation** de l'apprentissage automatique en réduisant l'écart technique des individus non experts.
- **Maintenabilité** basée sur la communauté et non sur des individus ou des groupes isolés.

## B.3 CONTRIBUTIONS

Nous résumons ici les contributions de cette thèse.

Dans le contexte de la **prévision de surendettement**(Chapter 3):

- Contrairement aux autres approches, nous considérons la prévision de surendettement comme un problème de classification multidimensionnelle et fournissons une solution complète pilotée par les données.
- Nous nous concentrons sur le processus de sélection des caractéristiques plutôt que sur des caractéristiques spécifiques sélectionnées à la main. Dans la littérature, plusieurs solutions reposent sur des ensembles fixes de caractéristiques, en supposant que: i) Les données de différentes institutions bancaires ont des distributions similaires. ii) Les caractéristiques de données sont réplicables. Des suppositions difficiles à réaliser dans des applications réelles.
- Nous étudions l'impact du déséquilibre extrême de classe. Bien que l'objectif principal soit d'identifier les personnes à risque, l'impact de la classification erronée sur les deux classes est pris en compte. i) Nous appliquons un critère multi-métrique pour trouver le meilleur compromis entre la performance et l'équité d'un modèle. ii) Nous discutons des coûts associés à la classification erronée des instances.
- Pour autant que nous sachions, cette étude est la première à considérer la prédiction du surendettement comme un problème d'apprentissage par flux de données. C'est une solution attrayante, car les modèles de flux s'adaptent aux changements des données, ce qui constitue un inconvénient de l'apprentissage par batch traditionnel, car de nouveaux modèles doivent être générés au fil du temps pour remplacer les modèles obsolètes.

Concernant l'**imputation de données manquantes** (Chapter 4):

- Nous proposons une nouvelle méthode d'imputation model-based, efficace et évolutive, qui jette le processus d'imputation comme un ensemble de tâches de classification ou de régression.

- Contrairement aux techniques d'imputation bien établies, la méthode proposée n'est pas restrictive en ce qui concerne le type de données manquantes, en soutenant: les mécanismes *manquant au hasard* et *manquant complètement au hasard*, données numériques et nominales, données petites à grande échelle, y compris les données de grande dimension.

- Nous fournissons une solution pour imputer plusieurs caractéristiques de données à la fois via un apprentissage multi-label. Dans la mesure de nos connaissances, notre approche est la première à aborder l'apprentissage multi-étiquettes pour les types de données hybrides, c'est-à-dire qu'il impute les données numériques et nominales simultanément.

Les contributions de l'**apprentissage rapide et lent** (Chapter 5) sont les suivantes:

- Nous proposons une stratégie unifiée d'apprentissage automatique basée sur les apprenants *fast* (par flux) et *slow* (par batch). En considérant l'apprentissage comme une tâche continue, nous montrons que les méthodes d'apprentissage par batch peuvent être efficacement adaptées à le cadre du flux dans des conditions spécifiques.

- Nous montrons l'applicabilité de ce nouveau paradigme en classification. Les résultats des tests sur des données synthétiques et réelles confirment que Fast and Slow Classifier exploite les méthodes de flux et de traitement par batch en combinant leurs forces: un modèle rapide s'adapte aux modifications des données et fournit des prévisions basées sur les concepts actuels, tandis qu'un modèle lent génère des modèles complexes construits sur des distributions de données plus larges.

- Nous pensons que notre recherche donne un nouveau regard sur la possibilité de solutions hybrides dans lesquelles l'apprentissage par batch et par flux interagit positivement. Cela revêt une importance particulière dans les applications réelles liées aux solutions par batch et dont le passage au cadre de flux représente un gros compromis.

Pour notre travail sur **boosting pour l'apprentissage en flux** (Chapter 6), nous listons les contributions suivantes:

- Nous proposons une adaptation de l'algorithme eXtreme Gradient Boosting (XGBoost) pour l'évolution des flux de données. L'idée centrale est la création et mise-à-jour progressif de l'ensemble, c'est-à-dire que les apprenants faibles sont formés sur des mini-batches de données, puis ajoutés à l'ensemble.

- De plus, nous considérons une approche simple, incrémentielle par batches, dans laquelle les membres de l'ensemble sont des modèles XGBoost complets formés sur des mini-batches consécutifs. Si la consommation de ressources est une considération secondaire, cette approche (après le réglage des paramètres) peut constituer un candidat intéressant pour une application dans des scénarios pratiques d'extraction de flux de données.

- Nous effectuons une évaluation approfondie en termes de performances, de pertinence des hyper-paramètres, de mémoire et de temps d'entraînement. Bien que l'objectif principal soit d'apprendre des flux de données, nous pensons que notre méthode de flux constitue une alternative intéressante à la version batch pour certaines applications, en raison de sa gestion efficace des ressources et de son adaptabilité. De plus, notre évaluation sert de mise à jour des études trouvées dans la littérature qui comparent les méthodes instance-incrémentielles et batch-incrémentales.

Finalement, les contributions de notre **framework de apprentissage par flux** open source (Chapter 7):

- Nous présentons scikit-multiflow, un framework open source pour l'apprentissage par flux de données et multi-output en Python. Le source-code est disponible publiquement et distribué sous la license BSD 3-Clause.

- scikit-multiflow fournit plusieurs méthodes d'apprentissage, générateurs de données et évaluateurs à la pointe de la technologie, pour différents problèmes d'apprentissage par flux, notamment single-output, multi-output et multi-label.

- scikit-multiflow est conçu comme une plate-forme pour encourager la démocratisation de la recherche sur l'apprentissage par flux. Par exemple, les méthodes proposées dans Chapter 5 et Chapter 6 sont implémentées sur scikit-multiflow et l'évaluation correspondante est (dans la plupart des cas) effectuée sur cette plateforme.

# BIBLIOGRAPHY


[1] Anja Achtziger, Marco Hubert, Peter Kenning, Gerhard Raab, and Lucia Reisch. "Debt out of control: The links between self-control, compulsive buying, and real debts." In: *Journal of Economic Psychology* 49 (2015), pp. 141–149.

[2] Edgar Acuña and Caroline Rodriguez. "The Treatment of Missing Values and its Effect on Classifier Accuracy." In: *Classification, Clustering, and Data Mining Applications* 1995 (2004), pp. 639–647. DOI: 10.1007/978-3-642-17103-1_60.

[3] Charu C Aggarwal, S Yu Philip, Jiawei Han, and Jianyong Wang. "-A Framework for Clustering Evolving Data Streams." In: *Proceedings 2003 VLDB Conference*. Elsevier. 2003, pp. 81–92.

[4] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Database mining: A performance perspective." In: *IEEE transactions on knowledge and data engineering* 5.6 (1993), pp. 914–925.

[5] Amir F. Atiya. "Bankruptcy prediction for credit risk using neural networks: a survey and new results." In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 12.4 (2001), pp. 929–935.

[6] Richard C Atkinson and Richard M Shiffrin. "Human memory: A proposed system and its control processes1." In: *Psychology of learning and motivation*. Vol. 2. Elsevier, 1968, pp. 89–195.

[7] Alan D Baddeley and Graham Hitch. "Working memory." In: *Psychology of learning and motivation*. Vol. 8. Elsevier, 1974, pp. 47–89.

[8] Manuel Baena-Garcia, Jose Del Campo-Avila, Raul Fidalgo, Albert Bifet, Ricard Gavalda, and Rafael Morales-bueno. "Early Drift Detection Method." In: *4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams* 6 (2006), pp. 77–86. DOI: 10.1.1.61.6101.

[9] Gustavo E A P A Batista and Maria Carolina Monard. "A study of k-nearest neighbour as an imputation method." In: *Frontiers in Artificial Intelligence and Applications* 87 (2002), pp. 251–260.

[10] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data." In: *ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets* 6.1 (2004), pp. 20–29.

[11] Alina Beygelzimer, Satyen Kale, and Haipeng Luo. "Optimal and Adaptive Algorithms for Online Boosting." In: *Sustainable Energy, Grids and Networks* 7 (Feb. 2015), pp. 70–79. DOI: 10.1016/j.segan.2016.06.001. arXiv: 1502.02651. URL: https://linkinghub.elsevier.com/retrieve/pii/S2352467716300285%20http://arxiv.org/abs/1502.02651.

[12] Albert Bifet, Eibe Frank, Geoff Holmes, and Bernhard Pfahringer. "Ensembles of Restricted Hoeffding Trees." In: *ACM Transactions on Intelligent Systems and Technology* 3.2 (Feb. 2012), pp. 1–20. DOI: 10.1145/2089094.2089106. arXiv: arXiv:1311.6355v1. URL: http://dl.acm.org/citation.cfm?doid=2089094.2089106.

[13] Albert Bifet and Ricard Gavaldà. "Learning from Time-Changing Data with Adaptive Windowing." In: *Proceedings of the 2007 SIAM International Conference on Data Mining* (2007), pp. 443–448. DOI: 10.1137/1.9781611972771.42.

[14] Albert Bifet and Ricard Gavaldà. "Adaptive Learning from Evolving Data Streams." In: *8th International Symposium on Intelligent Data Analysis* (2009), pp. 249–260. DOI: 10.1007/978-3-642-03915-7_22.

[15] Albert Bifet, Ricard Gavalda, Geoff Holmes, and Bernhard Pfahringer. *Machine Learning for Data Streams with Practical Examples in MOA*. https://moa.cms.waikato.ac.nz/book/. MIT Press, 2018.

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

[17] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. *DATA STREAM MINING A Practical Approach*. Tech. rep. 2011.

[18] 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*. 1. 2010, pp. 135–150.

[19] Albert Bifet, Bernhard Pfahringer, Jesse Read, and Geoff Holmes. "Efficient data stream classification via probabilistic adaptive windows." In: *Proceedings of the 28th annual ACM symposium on applied computing*. ACM. 2013, pp. 801–806.

[20] Christopher M Bishop. "Pattern Recognition and Machine Learning." In: (2006).

[21] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[22] Leo Breiman, David Wolpert, Philip Chan, and Salvatore Stolfo. "Pasting Small Votes for Classification in Large Databases and On-Line." In: *Machine Learning* 36 (1999), pp. 85–103.

[23] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Lujan. "Conditional Likelihood Maximisation: A Unifying Framework for Mutual Information Feature Selection." In: *Journal of Machine Learning Research* 13 (2012), pp. 27–66.

[24] Serpil Canbas, Altan Cabuk, and Suleyman Bilgin Kilic. "Prediction of commercial bank failure via multivariate statistical analysis of financial structures: The Turkish case." In: *European Journal of Operational Research* 166.2 (2005), pp. 528–546.

[25] Rich Caruana and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms." In: *Proceedings of the 23rd international conference on Machine learning* C.1 (2006), pp. 161–168.

[26] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: Synthetic minority oversampling technique." In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357.

[27] CL Philip Chen and Chun-Yang Zhang. "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data." In: *Information Sciences* 275 (2014), pp. 314–347.

[28] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System." In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794.

[29] Ales Chmelar. "Household Debt and the European Crisis." In: *European Credit Research Institute* 13 (2013).

[30] Civic Consulting. *The over-indebtedness of European households: updated mapping of the situation, nature and causes, effects and initiatives for alleviating its impact*. Tech. rep. DG SANCO, European Commission, 2013.

[31] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. "Online Passive-Aggressive Algorithms." In: *Journal of Machine Learning Research* 7.Mar (2006), pp. 551–585. DOI: 10.1.1.9.3429.

[32] 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)* (1984), pp. 278–292.

[33] A.P. Dempster, N.M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." In: *Journal of the Royal Statistical Society Series B Methodological* 39.1 (1977), pp. 1–38. DOI: http://dx.doi.org/10.2307/2984875. arXiv: 0710.5696v2.

[34] Thomas G. Dietterich. "Ensemble Methods in Machine Learning." In: (2000), pp. 1–15. DOI: 10.1007/3-540-45014-9_1. URL: http://link.springer.com/10.1007/3-540-45014-9%7B%5C_%7D1.

[35] Pedro Domingos and Geoff Hulten. "Mining High-speed Data Streams." In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '00. Boston, Massachusetts, USA: ACM, 2000, pp. 71–80.

[36] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[37] Pavlos S. Efraimidis and Paul G. Spirakis. "Weighted random sampling with a reservoir." In: *Information Processing Letters* 97.5 (2006), pp. 181–185. DOI: 10.1016/j.ipl.2005.11.003.

[38] Tom Fawcett. "An introduction to ROC analysis." In: *Pattern Recognition Letters* 27.8 (June 2006), pp. 861–874. ISSN: 01678655. DOI: 10.1016/j.patrec.2005.10.010.

[39] Françoise Fessant and Sophie Midenet. *Self-organising map for data imputation and correction in surveys*. 2002. DOI: 10.1007/s005210200002.

[40] Francois Fleuret. "Fast Binary Feature Selection with Conditional Mutual Information." In: *Journal of Machine Learning Research* 5 (2004), pp. 1531–1555.

[41] Nicole Fondeville, Erhan Őzdemir, and Terry Ward. *Over-indebtedness New evidence from the EU-SILC special module*. Tech. rep. European Comission, 2010.

[42] Eibe Frank, Geoffrey Holmes, and Richard Kirkby. "Racing committees for large datasets." In: *Discovery Science* (2002), pp. 153–164.

[43] Yoav Freund. "Boosting a weak learning algorithm by majority." In: *Information and computation* 121.2 (1995), pp. 256–285.

[44] Yoav Freund and Robert E. Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." In: *Journal of Computer and System Sciences* (1997), pp. 119–139.

[45] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer-Verlag New York, 2009.

[46] Drew Fudenberg and Eric Maskin. "The Folk Theorem in Repeated Games with Discounting or with Incomplete Information." In: *Econometrica* 54.3 (May 1986), p. 533. ISSN: 00129682. DOI: 10.2307/1911307.

[47] João Gama and Petr Kosina. "Recurrent concepts in data streams classification." In: *Knowledge and Information Systems* 40.3 (2014), pp. 489–507. DOI: 10.1007/s10115-013-0654-6.

[48] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. "Learning with Drift Detection." In: (2004), pp. 286–295. DOI: 10.1007/978-3-540-28645-5_29.

[49] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. "On evaluating stream learning algorithms." In: *Machine Learning* 90.3 (2013), pp. 317–346. DOI: 10.1007/s10994-012-5320-9.

[50] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. "A survey on concept drift adaptation." In: *ACM Computing Surveys* 46.4 (2014), pp. 1–37. DOI: 10.1145/2523813. arXiv: 1010.4784.

[51] John Gathergood. "Self-control, financial literacy and consumer over-indebtedness." In: *Journal of Economic Psychology* 33 (2012), pp. 590–602.

[52] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees." In: *Machine Learning* 63.1 (2006), pp. 3–42.

[53] Betti Gianni, Neil Dourmashkin, Mariacristina Rossi, and Ya Ping Yin. "Consumer Over-indebtedness in the EU: Measurement and Characteristics." In: *Journal of Economic Studies* 34.2 (2007).

[54] Heitor M. Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. "Adaptive random forests for evolving data stream classification." In: *Machine Learning* 106.9-10 (2017), pp. 1469–1495. ISSN: 15730565. DOI: 10.1007/s10994-017-5642-8.

[55] Isabelle Guyon and André Elisseeff. "An Introduction to Variable and Feature Selection." In: *Journal of Machine Learning Research (JMLR)* 3.3 (2003), pp. 1157–1182.

[56] Isabelle Guyon, Jiwen Li, Theodor Mader, Patrick A. Pletscher, Georg Schneider, and Markus Uhr. "Competitive baseline methods set new standards for the NIPS 2003 feature selection benchmark." In: *Pattern Recognition Letters* 28.12 (2007), pp. 1438–1444. ISSN: 01678655. DOI: 10.1016/j.patrec.2007.02.014.

[57] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. "The WEKA Data Mining Software: An Update." In: *SIGKDD Explor. Newsl.* 11.1 (Nov. 2009), pp. 10–18. ISSN: 1931-0145. DOI: 10.1145/1656274.1656278.

[58] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. "Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning." In: *Advances in intelligent computing* 17.12 (2005), pp. 878–887.

[59] Haibo He and Edwardo a. Garcia. "Learning from imbalanced data." In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (2009), pp. 1263–1284.

[60] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[61] James Honaker, Gary King, and Matthew Blackwell. "Amelia II: A Program for Missing Data." In: *Journal of Statistical Software* 45.1 (2011), pp. 1–47. ISSN: 1548-7660. DOI: 10.18637/jss.v045.i07.

[62] Csilla Horváth, Oliver B. Büttner, Nina Belei, and Feray Adıgüzel. "Balancing the balance: Self-control mechanisms and compulsive buying." In: *Journal of Economic Psychology* 49 (2015), pp. 120–132.

[63] Zan Huang, Hsinchun Chen, Chia-Jung Hsu, Wun-Hwa Chen, and Soushan Wu. "Credit rating analysis with support vector machines and neural networks: a market comparative study." In: *Decision Support Systems* 37.4 (2004), pp. 543–558.

[64] Geoff Hulten, Laurie Spencer, and Pedro Domingos. "Mining time-changing data streams." In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 97–106.

[65] Elena Ikonomovska, João Gama, and Sašo Džeroski. "Learning model trees from evolving data streams." In: *Data mining and knowledge discovery* 23.1 (2011), pp. 128–168.

[66] Young Hun Jung, Jack Goetz, and Ambuj Tewari. "Online Multiclass Boosting." In: *Advances in Neural Information Processing Systems*. 2017, pp. 919–928. DOI: 10.1016/j.steroids.2009.01.008. arXiv: 1702.07305. URL: http://arxiv.org/abs/1702.07305.

[67] Daniel Kahneman. *Thinking, fast and slow*. New York: Farrar, Straus and Giroux, 2011.

[68] Bernadette Kamleitner, Erik Hoelzl, and Erich Kirchler. "Credit use: Psychological perspectives on a multifaceted phenomenon." In: 47.1 (2012), pp. 1–27.

[69] Pilsung Kang. "Locally linear reconstruction based missing value imputation for supervised learning." In: *Neurocomputing* 118 (2013), pp. 65–78. DOI: 10.1016/j.neucom.2013.02.016.

[70] Gary King, James Honaker, Anne Joseph, and Kenneth Scheve. "Analyzing Incomplete Political Science Data." In: *American Political Science Review* 85.1269 (2001), pp. 49–69. DOI: 10.2307/3117628.

[71] Nicolas Kourtellis, Gianmarco De Francisci Morales, Albert Bifet, and Arinto Murdopo. "VHT: Vertical hoeffding tree." In: *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE. 2016, pp. 915–922.

[72] Alexandros Ladas, Eamonn Ferguson, Jon Garibaldi, and Uwe Aickelin. "A Data Mining framework to model Consumer Indebtedness with Psychological Factors." In: *IEEE International Conference of Data Mining: The Seventh International Workshop on Domain Driven Data Mining 2014 (DDDM 2014)* (2014). DOI: 10.1109/ICDMW.2014.148. arXiv: 1502.05911.

[73] Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. "Building High-level Features Using Large Scale Unsupervised Learning." In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. Edinburgh, Scotland: Omnipress, 2012, pp. 507–514.

[74] Mahnhoon Lee and Witold Pedrycz. "The Fuzzy C-means Algorithm with Fuzzy P-mode Prototypes for Clustering Objects Having Mixed Features." In: *Fuzzy Sets Syst.* 160.24 (Dec. 2009), pp. 3590–3600. ISSN: 0165-0114. DOI: 10.1016/j.fss.2009.06.015.

[75] Dan Li, Jitender Deogun, William Spaulding, and Bill Shuart. "Towards Missing Data Imputation: A Study of Fuzzy K-means Clustering Method." In: *Rough Sets and Current Trends in Computing: 4th International Conference, RSCTC 2004, Uppsala, Sweden, June 1-5, 2004. Proceedings*. Ed. by Shusaku Tsumoto, Roman Słowiński, Jan Komorowski, and Jerzy W. Grzymała-Busse. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 573–579. DOI: 10.1007/978-3-540-25929-9_70.

[76] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. "Feature Selection: A Data Perspective." In: *Journal of Machine Learning Research* (2016), pp. 1–73.

[77] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*. John Wiley & Sons, 2002.

[78] Xu Ying Liu, Jianxin Wu, and Zhi Hua Zhou. "Exploratory under-sampling for class-imbalance learning." In: *Proceedings - IEEE International Conference on Data Mining, ICDM* (2006), pp. 965–969.

[79] Viktor Losing, Barbara Hammer, and Heiko Wersing. "KNN classifier with self adjusting memory for heterogeneous concept drift." In: *Proceedings - IEEE International Conference on Data Mining, ICDM* 1 (2017), pp. 291–300. ISSN: 15504786. DOI: 10.1109/ICDM.2016.141.

[80] Markus Maier, Matthias Hein, and Ulrike Von Luxburg. "Optimal construction of k-nearest-neighbor graphs for identifying noisy clusters." In: *Theoretical Computer Science* 410 (2009), pp. 1749–1764. DOI: 10.1016/j.tcs.2009.01.009.

[81] Jae H. Min and Young-Chan Lee. "Bankruptcy Prediction Using Support Vector Machine with Optimal Choice of Kernel Function Parameters." In: *Expert Syst. Appl.* 28.4 (May 2005), pp. 603–614.

[82] Sung-Hwan Min, Jumin Lee, and Ingoo Han. "Hybrid genetic algorithms and support vector machines for bankruptcy prediction." In: *Expert Systems with Applications* 31.3 (2006), pp. 652–660. DOI: 10.1016/j.eswa.2005.09.070.

[83] Leandro L. Minku and Xin Yao. "DDD: A new ensemble approach for dealing with concept drift." In: *IEEE Transactions on Knowledge and Data Engineering* 24.4 (2012), pp. 619–633.

[84] Jacob Montiel, Albert Bifet, and Talel Abdessalem. "Predicting over-indebtedness on batch and streaming data." In: *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*. 2017, pp. 1504–1513. DOI: 10.1109/BigData.2017.8258084. URL: https://doi.org/10.1109/BigData.2017.8258084.

[85] Jacob Montiel, Albert Bifet, Viktor Losing, Jesse Read, and Talel Abdessalem. "Learning Fast and Slow: A Unified Batch/Stream Framework." In: *2018 IEEE International Conference on Big Data, BigData 2018, Seattle, WA, USA, December 10-13, 2018*. IEEE. 2018, pp. 1065–1072. DOI: 10.1109/BigData.2018.8622222.

[86] Jacob Montiel, Rory Mitchell, Eibe Frank, Bernhard Pfahringer, Talel Abdessalem, and Albert Bifet. "Adaptive XGBoost for Evolving Data Streams." In: (2018). Submited.

[87] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. "A Hybrid Framework for Scalable Model-based Imputation." In: (2018). Submited.

[88] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. "Scalable Model-Based Cascaded Imputation of Missing Data." In: *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part III*. 2018, pp. 64–76. DOI: 10.1007/978-3-319-93040-4_6. URL: https://doi.org/10.1007/978-3-319-93040-4%5C_6.

[89] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. "Scikit-Multiflow: A Multi-output Streaming Framework." In: *Journal of Machine Learning Research* 19.72 (Oct. 2018), pp. 1–5. eprint: abs/1807.04662. URL: https://github.com/scikit-multiflow/scikit-multiflow.

[90] Daniel J Mundfrom and Alan Whitcomb. "Imputing Missing Values: The Effect on the Accuracy of Classification." In: (1998).

[91] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. "Deep learning applications and challenges in big data analytics." In: *Journal of Big Data* 2.1 (2015), p. 1.

[92] José Manuel Otero-López and Estíbaliz Villardefrancos. "Compulsive buying and life aspirations: An analysis of intrinsic and extrinsic goals." In: *Personality and Individual Differences* 76 (2015), pp. 166–170.

[93] Nikunj C. Oza and Stuart Russell. "Online Bagging and Boosting." In: *Eighth International Workshop on Artificial Intelligence and Statistics*. Ed. by Tommi Jaakkola and Thomas Richardson. Key West, Florida. USA: Morgan Kaufmann, 2001, pp. 105–112.

[94] Ewan S Page. "Continuous inspection schemes." In: *Biometrika* 41.1/2 (1954), pp. 100–115.

[95] 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* 12.Oct (2011), pp. 2825–2830.

[96] Robi Polikar. "Ensemble based systems in decision making." In: *IEEE Circuits and Systems Magazine* 6.3 (2006), pp. 21–44. DOI: 10.1109/MCAS.2006.1688199.

[97] Robi Polikar, Lalita Udpa, Satish S. Udpa, and Vasant Honavar. "Learn++: An incremental learning algorithm for supervised neural networks." In: *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 31.4 (2001), pp. 497–508.

[98] Yongsong Qin, Shichao Zhang, Xiaofeng Zhu, Jilian Zhang, and Chengqi Zhang. "{POP} algorithm: Kernel-based imputation to treat missing values in knowledge discovery from databases." In: *Expert Systems with Applications* 36.2, Part 2 (2009), pp. 2794–2804. ISSN: 0957-4174. DOI: http://dx.doi.org/10.1016/j.eswa.2008.01.059.

[99] Jeff Racine and Qi Li. "Nonparametric estimation of regression functions with both categorical and continuous data." In: *Journal of Econometrics* 119.1 (2004), pp. 99–130. ISSN: 0304-4076. DOI: 10.1016/S0304-4076(03)00157-X.

[100] Md Geaur Rahman and Md Zahidul Islam. "Missing value imputation using decision trees and decision forests by splitting and merging records: Two novel techniques." In: *Knowledge-Based Systems* 53 (2013), pp. 51–65. ISSN: 09507051. DOI: 10.1016/j.knosys.2013.08.023.

[101] Md Geaur Rahman and Md Zahidul Islam. "Missing value imputation using a fuzzy clustering-based EM approach." In: *Knowledge and Information Systems* (2015), pp. 389–422. ISSN: 02193116. DOI: 10.1007/s10115-015-0822-y.

[102] Béatrice Raoult-Texier. *Study of Paths Leading to Over-indebtedness*. Tech. rep. Banque de France, Dec. 2014.

[103] P. Ravi Kumar and V. Ravi. "Bankruptcy prediction in banks and firms via statistical and intelligent techniques - A review." In: *European Journal of Operational Research* 180.1 (2007), pp. 1–28.

[104] Jesse Read, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. "Batch-incremental versus instance-incremental learning in dynamic and evolving data." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7619 LNCS (2012), pp. 313–323. DOI: 10.1007/978-3-642-34156-4_29.

[105] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. "Classifier Chains for Multi-label Classification." In: *Ecml/Pkdd*. 2009, pp. 254–269. DOI: 10.1007/978-3-642-04174-7_17.

[106] Jesse Read, Peter Reutemann, Bernhard Pfahringer, and Geoff Holmes. "MEKA: A Multi-label/Multi-target Extension to Weka." In: *Journal of Machine Learning Research* 17.21 (2016).

[107] Michael B Richman, Theodore B Trafalis, and Indra Adrianto. "Missing data imputation through machine learning algorithms." In: *Artificial Intelligence Methods in the Environmental Sciences*. Springer, 2009, pp. 153–169.

[108] Martin Scholz and Ralf Klinkenberg. "Boosting Classifiers for Drifting Concepts." In: *Intelligent Data Analysis* 11.1 (2007), pp. 1–40.

[109] Kyung-Shik Shin, Taik Soo Lee, and Hyun-jung Kim. "An application of support vector machines in bankruptcy prediction model." In: *Expert Systems with Applications* 28.1 (2005), pp. 127–135.

[110] Kyung-Shik Shin and Yong-Joo Lee. "A genetic algorithm application in bankruptcy prediction modeling." In: *Expert Systems with Applications* 23.3 (2002), pp. 321–328.

[111] Aisha Siddiqa, Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Mohsen Marjani, Shahabuddin Shamshirband, Abdullah Gani, and Fariza Nasaruddin. "A survey of big data management: Taxonomy and state-of-the-art." In: *Journal of Network and Computer Applications* 71 (2016), pp. 151–166.

[112] Brice Stone and Rosalinda Vasquez Maury. "Indicators of personal financial debt using a multi-disciplinary behavioral model." In: *Journal of Economic Psychology* 27.4 (2006), pp. 543–556.

[113] W Nick Street and YongSeog Kim. "A streaming ensemble algorithm (SEA) for large-scale classification." In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 377–382.

[114] Xiaoyuan Su, Russell Greiner, Taghi M Khoshgoftaar, and Amri Napolitano. "Using Classifier-Based Nominal Imputation to Improve Machine Learning." In: *Advances in Knowledge Discovery and Data Mining, Pt I: 15th Pacific-Asia Conference, Pakdd 2011* 6634.Mci (2011), pp. 124–135.

[115] Chih-Fong Tsai. "Feature selection in bankruptcy prediction." In: *Knowledge-Based Systems* 22.2 (2009), pp. 120–127.

[116] Chih-Fong Tsai and Jhen-Wei Wu. "Using Neural Network Ensembles for Bankruptcy Prediction and Credit Scoring." In: *Expert Syst. Appl.* 34.4 (May 2008), pp. 2639–2649.

[117] Theodore Vasiloudis, Foteini Beligianni, and Gianmarco De Francisci Morales. "BoostVHT." In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17*. New York, New York, USA: ACM Press, 2017, pp. 899–908. DOI: 10.1145/3132847.3132974. URL: http://dl.acm.org/citation.cfm?doid=3132847.3132974.

[118] Jeffrey S. Vitter. "Random sampling with a reservoir." In: *ACM Transactions on Mathematical Software* 11.1 (1985), pp. 37–57. ISSN: 00983500. DOI: 10.1145/3147.3165.

[119] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. "Mining concept-drifting data streams using ensemble classifiers." In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*. Vol. 42. New

York, New York, USA: ACM Press, 2003, p. 226. DOI: 10.1145/956750.956778.

[120] Ling Wang and Dong Mei Fu. "Estimation of missing values using a weighted k-nearest neighbors algorithm." In: *Proceedings - 2009 International Conference on Environmental Science and Information Application Technology, ESIAT 2009* 3.2 (2009), pp. 660–663. DOI: 10.1109/ESIAT.2009.206.

[121] Howard Hua Yang and John Moody. "Data Visualization and Feature Selection: New Algorithms for Nongaussian Data." In: *Advances in Neural Information Processing Systems* 12.Mi (1999), pp. 687–693.

[122] Chengqi Zhang, Yongsong Qin, Xiaofeng Zhu, Jilian Zhang, and Shichao Zhang. "Clustering-based missing value imputation for data preprocessing." In: *Industrial Informatics, 2006 IEEE International Conference on*. IEEE. 2006, pp. 1081–1086.

[123] Jianping Zhang and Inderjeet Mani. "kNN Approach to Unbalanced Data Distributions: A Case Study involving Information Extraction." In: *Workshop on Learning from Imbalanced Datasets II ICML* (2003), pp. 42–48.

[124] Xiaofeng Zhu, Shichao Zhang, Zhi Jin, Zili Zhang, and Zhuoming Xu. "Missing value estimation for mixed-attribute data sets." In: *IEEE Transactions on Knowledge and Data Engineering* 23.1 (2011), pp. 110–121. ISSN: 10414347. DOI: 10.1109/TKDE.2010.99.

Cover image designed by starline / Freepik.
*Fast and Slow Machine Learning*, prepared by Jacob Montiel López
between August and December, 2018.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

**Titre :** Apprentissage Automatique Rapide et Lent

**Mots clés :** Flux de Données, Classification, Données Manquent, Dérive de Concept

**Résumé :** L'ère du Big Data a révolutionné la manière dont les données sont créées et traitées. Dans ce contexte, de nombreux défis se posent, compte tenu de la quantité énorme de données disponibles qui doivent être efficacement gérées et traitées afin d'extraire des connaissances. Cette thèse explore la symbiose de l'apprentissage en mode batch et en flux, traditionnellement considérés dans la littérature comme antagonistes, sur le problème de la classification à partir de flux de données en évolution.
L'apprentissage en mode batch est une approche bien établie basée sur une séquence finie : d'abord les données sont collectées, puis les modèles prédictifs sont créés, finalement le modèle est appliqué. Par contre, l'apprentissage par flux considère les données comme infinies, rendant le problème d'apprentissage comme une tâche continue (sans fin). De plus, les flux de données peuvent évoluer dans le temps, ce qui signifie que la relation entre les caractéristiques et la réponse correspondante peut changer.
Nous proposons un cadre systématique pour prévoir le surendettement, un problème du monde réel ayant des implications importantes dans la société moderne. Les deux versions du mécanisme d'alerte précoce (batch et flux) surpassent les performances de base de la solution mise en œuvre par le Groupe BPCE, la deuxième institution bancaire en France. De plus, nous introduisons une méthode d'imputation évolutive basée sur un modèle pour les données manquantes dans la classification. Cette méthode présente le problème d'imputation sous la forme d'un ensemble de tâches de classification / régression résolues progressivement.
Nous présentons un cadre unifié qui sert de plate-forme d'apprentissage commune où les méthodes de traitement par batch et par flux peuvent interagir de manière positive. Nous montrons que les méthodes batch peuvent être efficacement formées sur le réglage du flux dans des conditions spécifiques. Nous proposons également une adaptation de l'Extreme Gradient Boosting algorithme aux flux de données en évolution. La méthode adaptative proposée génère et met à jour l'ensemble de manière incrémentielle à l'aide de mini-lots de données. Enfin, nous présentons scikit-multiflow, un framework open source en Python qui comble le vide en Python pour une plate-forme de développement/recherche pour l'apprentissage à partir de flux de données en évolution.

**Title :** Fast and Slow Machine Learning

**Keywords :** Data Stream, Classification, Missing Data, Concept Drift

**Abstract :** The Big Data era has revolutionized the way in which data is created and processed. In this context, multiple challenges arise given the massive amount of data that needs to be efficiently handled and processed in order to extract knowledge. This thesis explores the symbiosis of batch and stream learning, which are traditionally considered in the literature as antagonists. We focus on the problem of classification from evolving data streams.
Batch learning is a well-established approach in machine learning based on a finite sequence : first data is collected, then predictive models are created, then the model is applied. On the other hand, stream learning considers data as infinite, rendering the learning problem as a continuous (never-ending) task. Furthermore, data streams can evolve over time, meaning that the relationship between features and the corresponding response (class in classification) can change. We propose a systematic framework to predict over-indebtedness, a real-world problem with significant implications in modern society. The two versions of the early warning mechanism (batch and stream) outperform the baseline performance of the solution implemented by the Groupe BPCE, the second largest banking institution in France. Additionally, we introduce a scalable model-based imputation method for missing data in classification. This method casts the imputation problem as a set of classification/regression tasks which are solved incrementally.
We present a unified framework that serves as a common learning platform where batch and stream methods can positively interact. We show that batch methods can be efficiently trained on the stream setting under specific conditions. The proposed hybrid solution works under the positive interactions between batch and stream methods. We also propose an adaptation of the Extreme Gradient Boosting (XGBoost) algorithm for evolving data streams. The proposed adaptive method generates and updates the ensemble incrementally using mini-batches of data. Finally, we introduce scikit-multiflow, an open source framework in Python that fills the gap in Python for a development/research platform for learning from evolving data streams.

**Université Paris-Saclay**
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France