Université Libre de Bruxelles
Computer Science Department
Machine Learning Group

Adaptive Machine Learning for Credit Card Fraud Detection

A thesis submitted in fulfillment of the requirements for the Ph.D. degree in Computer Science

Author:
Andrea Dal Pozzolo

Supervisor:
Prof. Gianluca Bontempi

December 2015

Declaration of Authorship

I, Andrea Dal Pozzolo, declare that this thesis titled "Adaptive Machine Learning for Credit Card Fraud Detection" has been composed by myself and contains original work of my own execution. Some of the reported work has been done in collaboration with a number of co-authors whose contributions are acknowledged in the relevant sections.

This thesis has been written under the supervision of Prof. Gianluca Bontempi.
The members of the jury are:

- Prof. Bart Baesens (Katholieke Universiteit Leuven, Belgium)
- Prof. Cesare Alippi (Politecnico di Milano, Italy)
- Prof. Gianluca Bontempi (Université Libre de Bruxelles, Belgium)
- Prof. Tom Lenaerts (Université Libre de Bruxelles, Belgium)
- Prof. Yves De Smet (Université Libre de Bruxelles, Belgium)
- Dr. Olivier Caelen (Worldline S.A., Belgium)

"The cure for boredom is curiosity. There is no cure for curiosity."

Dorothy Parker

UNIVERSITÉ LIBRE DE BRUXELLES
Computer Science Department
Machine Learning Group

Abstract

Adaptive Machine Learning for Credit Card Fraud Detection

by Andrea Dal Pozzolo

Billions of dollars of loss are caused every year by fraudulent credit card transactions. The design of efficient fraud detection algorithms is key for reducing these losses, and more and more algorithms rely on advanced machine learning techniques to assist fraud investigators. The design of fraud detection algorithms is however particularly challenging due to the non-stationary distribution of the data, the highly unbalanced classes distributions and the availability of few transactions labeled by fraud investigators. At the same time public data are scarcely available for confidentiality issues, leaving unanswered many questions about what is the best strategy. In this thesis we aim to provide some answers by focusing on crucial issues such as: i) why and how undersampling is useful in the presence of class imbalance (i.e. frauds are a small percentage of the transactions), ii) how to deal with unbalanced and evolving data streams (non-stationarity due to fraud evolution and change of spending behavior), iii) how to assess performances in a way which is relevant for detection and iv) how to use feedbacks provided by investigators on the fraud alerts generated. Finally, we design and assess a prototype of a Fraud Detection System able to meet real-world working conditions and that is able to integrate investigators' feedback to generate accurate alerts.

UNIVERSITÉ LIBRE DE BRUXELLES
Département d'Informatique
Machine Learning Group

Résumé

Adaptive Machine Learning for Credit Card Fraud Detection

par Andrea Dal Pozzolo

Des milliards de dollars de pertes sont réalisés chaque année en raison de transactions frauduleuses sur les cartes de crédit. La clé pour réduire ces pertes est le développement d'algorithmes de détection de fraude efficaces. De plus en plus d'algorithmes se basent sur des techniques de machine learning avancées pour assister les détecteurs de fraude. La conception d'algorithmes de détection de la fraude est toutefois particulièrement difficile en raison de la distribution non-stationnaire des données, de la distribution trés déséquilibrée des classes et la disponibilité de seulement quelques transactions cataloguées par les détecteurs de fraude. Dans le même temps, des données publiques ne sont guère disponibles pour des raisons de confidentialité, laissant de nombreuses questions sans réponse à propos de ce qui est la meilleure stratégie pour traiter avec eux. Dans cette thèse, nous proposons quelques réponses en se concentrant sur des questions cruciales telles que: i) pourquoi et comment undersampling est utile en présence de classes déséquilibrées (c-à-d les fraudes sont un petit pourcentage des transactions), ii) comment traiter l'évolution du flux de données non-balancées (non-stationnaire en raison de l'évolution et le changement de comportement de dépense fraude), iii) les évaluations de la performance qui sont pertinentes pour la détection et iv) l'utilisation des évaluations fournies par les détecteurs sur les alertes de fraude générées. Enfin, nous concevons un système de détection de la fraude en mesure de satisfaire les conditions de travail dans le monde réel et qui est capable d'intégrer les détecteurs avec rétroaction pour générer des alertes de fraude précises.

Acknowledgements

Foremost, I would like to express my deep and sincere gratitude to my supervisor Prof. Gianluca Bontempi for his inspiration, patience and encouragement. He introduced me to the field of Machine Learning and provided lots of good ideas and advice. I would have never completed this thesis without his help. Besides Gianluca, I am deeply grateful to Dr. Olivier Caelen for his suggestions and guidance throughout my PhD study. His great ideas and unfailing support were essential to finish this thesis.

I would like to thank also Prof. Giacomo Boracchi and Prof. Cesare Alippi for their supervision and fruitful discussions on the problem of learning in non-stationary environments during my research visits at Politecnico di Milano. A great thanks also to Prof. Nitesh V. Chawla and Dr. Reid A. Johnson for sharing their precious knowledge and expertise on the topic of unbalanced classification during my research visits at Notre Dame University. My sincere thanks also go to the PhD committee members and the examiners of this thesis for the insightful comments and hard questions throughout my PhD study.

A well-known quote says that behind every great man there's a great woman. While I'm not a great man, there's a great woman behind me and her name is Katia. All my achievements also depends on her, so thank you my love for having always supported me during these four years.

If I was able to arrive to the PhD studies it is because I had two great parents, Francesco and Francesca, that always believed in what I was doing. I wish to convey special thanks to my family for their endless support and understanding.

Last but not least, I would like to thank my friends (from Italy and Belgium) and colleagues both from the Computer Science Department of ULB and Worldline S.A. (former and current) for their continuous encouragement in these 4 years. A big thanks also to Serge Waterschoot (Fraud Risk Manager at Worldline S.A.) that made possible this PhD and Innoviris[1] (Brussels Region) that funded the *Doctiris* project.

[1] Innoviris is the Brussels institute for the encouragement of scientific research and innovation.

Contents

List of Figures

List of Tables

List of Acronyms

To my future wife Katia

Part I

Overview

Chapter 1

Introduction

1.1 The problem of Fraud Detection

Fraud is as old as humanity itself and can take an unlimited variety of different forms. Moreover, the development of new technologies provides additional ways in which criminals may commit fraud [2], for instance in e-commerce the information about the card is sufficient to perpetrate a fraud. The use of credit cards is prevalent in modern day society and credit card fraud has kept on growing in recent years. Financial losses due to fraud affect not only merchants and banks (e.g. reimbursements), but also individual clients. If the bank loses money, customers eventually pay as well through higher interest rates, higher membership fees, etc. Fraud may also affect the reputation and image of a merchant causing non-financial losses that, though difficult to quantify in the short term, may become visible in the long period. For example, if a cardholder is victim of fraud with a certain company, he may no longer trust their business and choose a competitor.

The actions taken against fraud can be divided into fraud *prevention*, which attempts to block fraudulent transactions at source, and fraud *detection*, where successful fraud transactions are identified a posteriori. Technologies that have been used in order to prevent fraud are Address Verification Systems (AVS), Card Verification Method (CVM) and Personal Identification Number (PIN). AVS involves verification of the address with zip code of the customer while CVM and PIN involve checking of the numeric code that is keyed in by the customer. For prevention purposes, financial institutions challenge all transactions with rule based filters and data mining methods as neural networks [3].

Fraud detection is, given a set of credit card transactions, the process of identifying if a new authorized transaction belongs to the class of fraudulent or genuine transactions [4]. A Fraud Detection System (FDS) should not only detect fraud cases efficiently,

but also be cost-effective in the sense that the cost invested in transaction screening should not be higher than the loss due to frauds [5]. Bhatla [6] shows that screening only 2% of transactions can result in reducing fraud losses accounting for 1% of the total value of transactions. However, a review of 30% of transactions could reduce the fraud losses drastically to 0.06%, but increase the costs exorbitantly. In order to minimize costs of detection it is important to use expert rules and statistical based models (e.g. Machine Learning) to make a first screen between genuine and potential fraud and ask the investigators to review only the cases with high risk.

Typically, transactions are first filtered by checking some essential conditions (e.g. sufficient balance) and then scored by a predictive model (see Figure 1.1). The predictive model scores each transaction with high or low risk of fraud and those with high risk generate alerts. Investigators check these alerts and provide a feedback for each alert, i.e. true positive (fraud) or false positive (genuine). These feedbacks can then be used to improve the model. A predictive model can be built upon experts' rules, i.e. rules based on knowledge from fraud experts, but these require manual tuning and human supervision. Alternatively, with Machine Learning (ML) techniques [7] we can efficiently discover fraudulent patterns and predict transactions that are most likely to be fraudulent. ML techniques consist in inferring a prediction model on the basis of a set of examples. The model is in most cases a parametric function, which allows predicting the likelihood of a transaction to be fraud, given a set of features describing the transaction. In the domain of fraud detection, the use of learning techniques is attractive for a number of reasons. First, they allow to discovery patterns in high dimensional data streams, i.e. transactions arrive as a continuous stream and each transaction is defined by many variables. Second, fraudulent transactions are often correlated both over time and space. For examples, fraudsters typically try to commit frauds in the same shop with different cards within a short time period. Third, learning techniques can be used to detect and model existing fraudulent strategies as well as identify new strategies associated to unusual behavior of the cardholders. Predictive models based on ML techniques are also able to automatically integrate investigators' feedbacks to improve the accuracy of the detection, while in the case of expert system, including investigators feedbacks requires rules revision that can be tedious and time consuming.

When a fraud cannot be prevented, it is desirable to detect it as rapidly as possible. In both cases, prevention and detection, the problem is magnified by a number of domain constraints and characteristics. First, care must be taken not to prevent too many legitimate transactions or incorrectly block genuine cards. Customer irritation is to be avoided. Second, most banks process vast numbers of transactions, of which only a small fraction is fraudulent, often less than 0.1% [3]. Third, only a limited number of transactions can be checked by fraud investigators, i.e. we cannot ask a human person

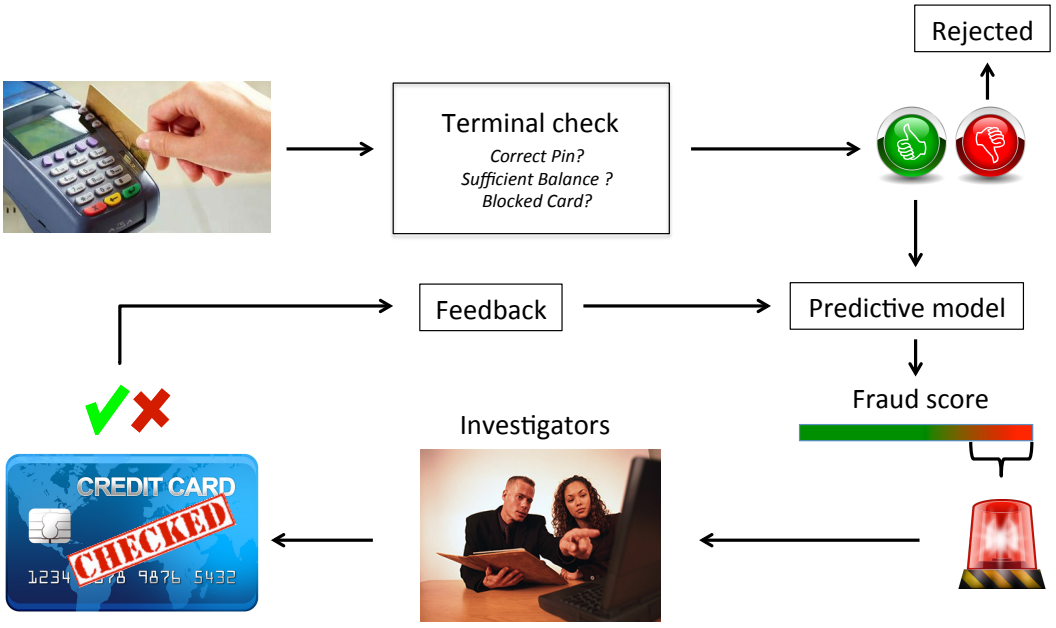


FIGURE 1.1: The Credit Card Fraud Detection process. Credit cards payments need to pass a first terminal check and then if not rejected are scored by a predictive model that raises alerts for the most suspicious transactions. Investigators provide feedbacks on the alerts (transaction labeled as fraud or genuine) that can be used to improve the accuracy of the predictive model.

to check all transactions one by one if it is fraudulent or not. In other words companies and public institutions need automatic systems able to support fraud detection [8].

Credit card frauds may occur in various ways [9]: just to mention some, we can have stolen card fraud, cardholder-not-present fraud and application fraud:

- *Stolen card fraud* is the most common type of fraud where the fraudster usually tries to spend as much as possible and as quickly as possible. The detection of such a fraud typically relies on the discovery of an unexpected usage pattern of the credit card (generally unexpectedly important) with respect to the common practice.

- *Cardholder-not-present fraud* is often observed in e-business. Here the fraudster needs the information about a credit card but not the card itself. This fraud demands a prompt detection since, unlike the previous case, the official card owner is not aware that his own data have been stolen.

- *Application fraud* corresponds to the application for a credit card with false personal information. This kind of fraud occurs more rarely since it could be detected during the application by checking the information of the applier, contrary to other frauds that can not be anticipated.

The previous classification however is not exhaustive, since frauds are continuously evolving. For a more general discussion of different fraud types we refer the reader to [10]. Whenever a fraud method is detected, criminals adapt their strategies and try others. At the same time, everyday there are new criminals taking part in the game and trying new and old strategies. In this setting it is important to update the detection tools, but keeping old ones as well [2]. Exchange of ideas for detection tools is difficult as fraudsters could benefit from it by testing their strategies. For the same reason datasets are typically not publicly available to the research community.

1.2 The impact of frauds

With this extensive use of credit cards, fraud appears as a major issue in the credit card business. It is hard to have some figures on the impact of fraud, since companies and banks do not like to disclose the amount of losses due to frauds. Another problem in credit-card fraud loss estimation is that we can measure the loss of only those frauds that have been detected, and it is not possible to assess the size of unreported/undetected frauds. Other frauds are reported long after the criminal has completed the crime [11].

However, the Association for Payment Clearing Services (APACS) has estimated that total losses through credit card fraud in the United Kingdom have been growing rapidly from £122 million in 1997 to £440.3 million in 2010 [8]. According to The Nilson Report [12], Global Credit, Debit, and Prepaid Card Fraud losses reached $11.27 Billion in 2012 - Up 14.6% Over 2011. Gross fraud losses accounted for 5.22% total volume, up from 5.07% in 2011. In 2012, only in the USA fraud losses reached $5.33 billion. According to the Lexis Nexis [13], in 2014 fraudulent card transactions worldwide have reached around $11 billion a year, and the USA may account for about half of that.

The European Central Bank (ECB) reports [14] that, in 2012, €1 in every €2'635 spent on credit and debit cards issued within SEPA (the European Union, Iceland, Liechtenstein, Monaco, Norway and Switzerland) was lost to fraud. The total value of fraud was estimated reaching €1.33 billion in 2012, registering an increase of 14.8% compared with 2011. In particular 60% of these frauds came from card-not-present (CNP) payments (i.e. payments via post, telephone or the internet), 23% from point-of-sale (POS) terminals and 17% from ATMs. The introduction of EMV security standard (chip on cards) have reduced fraud share (0.048% in 2008 and 0.038% in 2012) on the total number of transactions.[1] However, from 2011 to 2012, CNP frauds have increased by 21%, following the growing of CNP payments, which rose by around 15% to 20% a year between 2008 and 2012, while all the other transactions rose by 4%. The ECB report

[1] With EMV cards the cardholder is asked to enter a PIN.

shows also that credit cards are more affected by fraud than debit card, estimating that for every €1000 we have €1 of loss due to fraud in credit while €1 for every €5'400 in debit card. Another interesting fact is that CNP fraud is usually more frequent in mature card markets, whereas POS fraud is more common in less developed markets.

The 2015 Cybersource report [15] shows that businesses are reluctant towards the adoption of 3-D Secure methods [16] (online authentications based on a three-domain model: acquirer, issuer and interoperability), because it may cause poor customer experience and the risk of customers abandoning their purchases.

1.3 Credit Card Fraud Detection

Credit card fraud detection relies on the analysis of recorded transactions. Transaction data are mainly composed of a number of attributes (e.g. credit card identifier, transaction date, recipient, amount of the transaction). Automatic systems are essential since it is not always possible or easy for a human analyst to detect fraudulent patterns in transaction datasets, often characterized by a large number of samples, many dimensions and online updates. Also, the cardholder is not reliable in reporting the theft, loss or fraudulent use of a card [17]. In the following we will now discuss advantages and disadvantages of Expert Driven and Data Driven approaches to fraud detection.

The Expert Driven approach uses domain knowledge from fraud investigators to define rules that are used to predict the probability of a new transaction to be fraudulent. Let us imagine that the investigators know from experience that a transaction done on a betting website with an amount greater than $10000 is almost certain to be fraudulent. Then we can automatize the detection by mean of a rule as "IF transaction amount $>$ $10000 & Betting website THEN fraud probability $= 0.99$". In the same spirit we can define a set of rules for different scenarios. Typically, expert rules can be distinguished between scoring rules and blocking rules. The former assigns a score to a transaction based on the risk the investigators associate to a certain pattern; the latter can block the transaction because the risk of fraud is too high. The advantages of expert rules are: i) they are easy to develop and to understand, ii) they explain why an alert was generated and iii) they exploit domain expert knowledge. However, they have a number of drawbacks: i) they are subjective (if you ask 7 experts you may get 7 different opinions), ii) they detect only easy correlations between variables and frauds (it is hard for a human analyst to think in more the three dimension and explore all possible pattern combinations), iii) they are able to detect only known fraudulent strategies, iv) they require human monitoring/supervision (update in case of performance drop) and v) they can become obsolete soon due to fraud evolution.

A different way to automatize the detection is by means of Data Driven methods, i.e. setting up a FDS based on Machine Learning able to learn from data in a supervised or unsupervised manner which patterns are the most probably related to a fraudulent behavior. With Machine Learning we let the computers to discover fraudulent patterns in the data. Data Driven approaches have also advantages and disadvantages, for instance with Machine Learning algorithms we can: i) learn complex fraudulent configurations (use all features available), ii) ingest large volumes of data, iii) model complex distributions, iv) predict new types of fraud (anomalies from *genuine* patterns) and v) adapt to changing distribution in the case of fraud evolution. However, they have also some drawbacks: i) they need enough samples, ii) some models are black box, i.e. they are not easily interpretable by investigators and they do not provide an understanding of the reason why an alert is generated.

Data Driven and Expert Driven methods usually work in parallel and their combination is often the best solution. Typically, real-world FDSs like the one of our project partner (Worldline S.A. from Brussels, Belgium) use Data Driven methods to obtain precise alerts (reduce False Positive (FP)), while Expert Driven ones are mostly used to make sure that all the frauds are detected (reduce False Negative (FN)) at the cost of having few false alerts.[2] Data Driven solutions should not be seen as a tool to replace expert-based system, but rather as a support to obtain a more accurate detection. A detailed description of how these two types of methods work and how they are combined will be provided in Section 2.2.1.

This thesis will concern automatic data driven methods based on Machine Learning techniques. The design of a FDS based on Data Driven Models (DDMs) is not an easy task, it requires the practitioners to decide which feature to use, strategy (e.g. supervised or unsupervised), algorithm (e.g. decision trees, neural network, support vector machine), frequency of update of the model (once a year, monthly or every time new data is available), etc. We hope that, by the end of the thesis, the reader will have a better understanding of how to design and implement an effective data driven fraud detection solution.

1.4 Challenges in Data Driven Fraud Detection Systems

The design of a FDSs employing DDMs based on Machine Learning algorithms is particularly challenging for the following reasons:

[2]From personal communication with the project collaborators it has emerged that Expert Driven methods are more recall oriented, while Data Driven ones are optimized for precision, i.e. the first aim to reduce missed frauds and the second to reduce false alerts.

1. Frauds represent a small fraction of all the daily transactions [18].
2. Frauds distribution evolves over time because of seasonality and new attack strategies [19].
3. The true nature (class) of the majority of transactions is typically known only several days after the transaction took place, since only few transactions are timely checked by investigators [20].

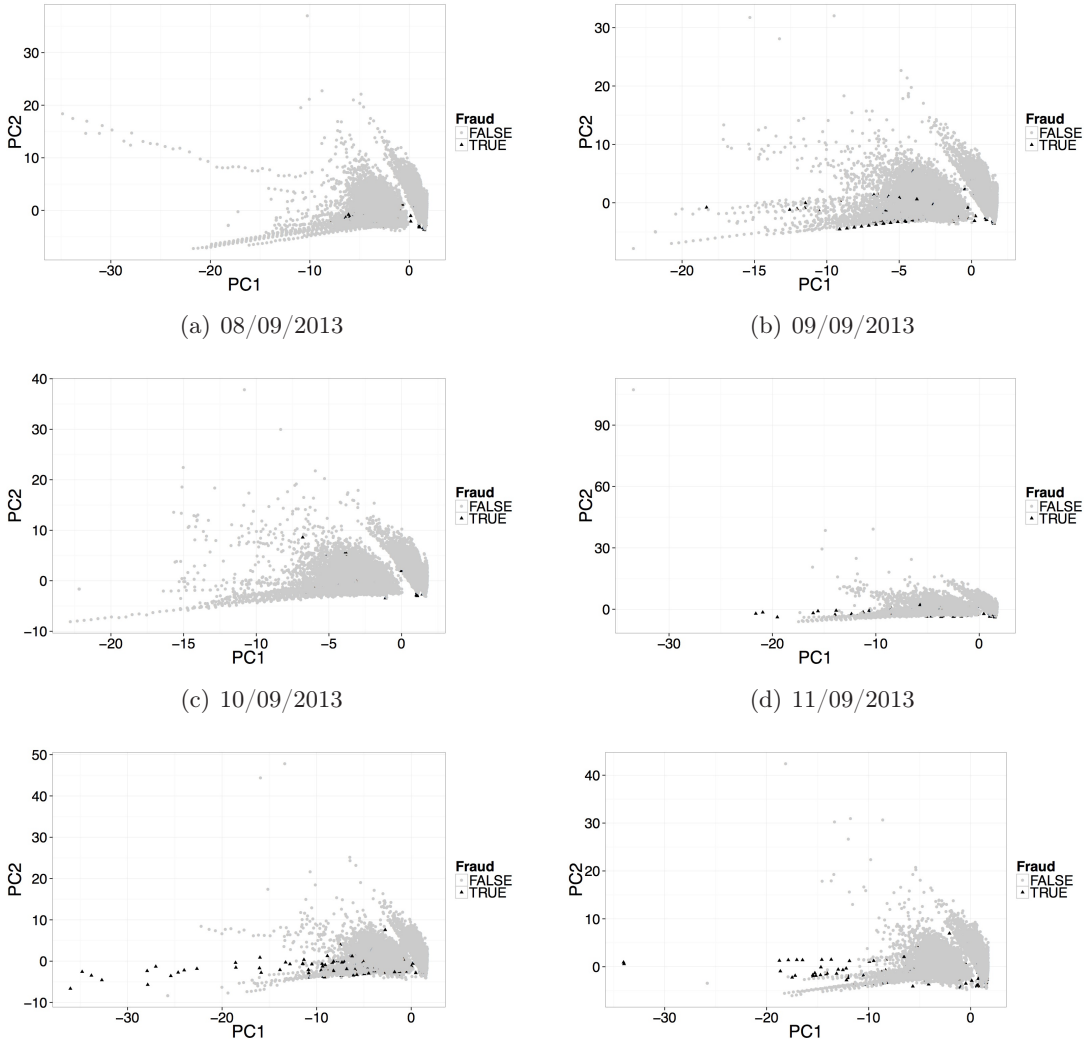
The first challenge is also known as the *unbalanced problem* [21], since the distribution of the transactions is skewed towards the genuine class. The distributions of genuine and fraud samples are not only unbalanced, but also overlapping (see the plot over the first two principal components in Figure 1.2). Most Machine Learning algorithms are not designed to cope with a both unbalanced and overlapped class distributions [22].

The change in fraudulent activities and costumer behavior is the main responsible of non-stationarity in the stream of transactions. This situation is typically referred to as *concept drift* [23] and is of extreme relevance for FDSs which have to be constantly updated either by exploiting the most recent supervised samples or by forgetting outdated information that might be no more useful whereas not misleading. FDS strategies that are not updated or revisited frequently are often losing their predictive accuracy in the long term [18].

The third challenge is related to the fact that, in a real-world setting, it is impossible to check all transactions. The cost of human labour seriously constrains the number of alerts, returned by the FDS, which can be validated by investigators. Investigators check FDS alerts by calling the cardholders, and then provide the FDS with *feedbacks* indicating whether the alerts were related to fraudulent or genuine transactions. These feedbacks, which refer to a tiny fraction of the daily transactions amount, are the only real-time information that can be provided to train or update classifiers. The class (fraudulent / non-fraudulent) of the rest of transactions is known only several days later. Classes can be automatically assigned when a certain time period has passed, e.g. by assuming a certain reaction time for customers to discover and then report frauds. Standard FDSs ignoring feedbacks from investigators often provide less accurate alerts than FDSs able to use efficiently both feedbacks and the other supervised samples available [20].

1.5 Contributions

The contributions of the thesis address the challenges discussed in the previous section.



(a) 08/09/2013 (b) 09/09/2013

(c) 10/09/2013 (d) 11/09/2013

(e) 12/09/2013 (f) 13/09/2013

FIGURE 1.2: Plot of genuine and fraudulent transactions between the 8^{th} and 13^{th} of September 2013 over the first two principal components. Frauds represent a small part of all observations available. The distributions of fraud and genuine samples are highly overlapping and changes over the time.

1.5.1 Understanding sampling methods

The first main contribution of this thesis is the formalization of sampling methods adopted in unbalanced classification tasks. In particular we focus on undersampling which is a standard technique for balancing skewed distributions. Despite its popularity in the machine learning community, there was no detailed analysis about the impact of undersampling on the accuracy of the final classifier. Chapter 4 reveals the condition under which undersampling will improve the ranking of the posterior probability returned by a classifier. Then we propose a method to calibrate the posterior probability of a classifier applied after undersampling. However, there is no guarantee that undersampling

is the best method to adopt for a given classifier and dataset, so we propose to use a racing algorithm to adaptively select the best unbalanced strategy.

1.5.2 Learning from evolving and unbalanced data streams

The second main contribution is related to the non-stationary nature of transaction distributions in credit cards. The continuous evolution in the way fraudsters attempt and commit illegal activities requires a continuous adaptation of the learning algorithm used to detect frauds. At the same time the distribution is highly skewed towards the genuine class, making standard techniques for evolving data streams not suitable. In Chapter 5 we investigate multiple solutions for unbalanced data stream with the goal of showing in practice which strategy is the best to adopt for fraud detection. We show that retaining historical transactions is useful, but it is also important to forget outdated samples for the model to be precise. Also, propagation of fraudulent instances along the streams is another effective technique to deal with such unbalanced data streams. When propagation of old transactions in the stream is not desired (e.g. for computational reason), we suggest to use an ensemble of Hellinger Distance Decision Tree (HDDT) as they showed better performances than approaches based on instance propagations.

1.5.3 Formalization of a real-world Fraud Detection System

In Chapter 6 we present a prototype of a FDS able to meet real-world working conditions, which is the third main contribution of the thesis. Most FDSs monitor streams of credit card transactions by means of classifiers returning alerts for the riskiest payments. Fraud Detection differs from conventional classification tasks because, in a first phase, only a small set of supervised samples is provided by human investigators. Labels of the vast majority of transactions are made available only several days later, when customers have possibly reported unauthorized transactions. The delay in obtaining accurate labels and the interaction between alerts and supervised information has to be carefully taken into consideration when learning in a concept-drifting environment. We show that investigator's feedbacks and delayed labels have to be handled separately and require different classification strategies. Finally, based on our results, we argue that the best detection solution consists in training two separate classifiers (on feedbacks and delayed labels, respectively), and then aggregating the outcomes.

1.5.4 Software and Credit Card Fraud Detection Dataset

Besides these main contributions, during this thesis we also developed a software package called `unbalanced` [24] available for the R language [25]. This package is presented in Appendix A, it implements some of most well-known techniques for unbalanced classification and proposes a racing algorithm [26] to select adaptively the most appropriate strategy for a given unbalanced task [27].

Another contribution of this thesis is the release to the public of a dataset containing information about credit card transaction with examples of fraudulent samples. The dataset was first used in the experiments of our paper [28] and then made available at: `http://www.ulb.ac.be/di/map/adalpozz/data/creditcard.Rdata`. It contains 31 numerical input variables. Feature "Time" denotes the seconds elapsed between each transaction and the first transaction in the dataset. The feature "Amount" is the transaction amount, which can be used for example-dependent cost-sensitive learning. Feature "Class" is the response variable and it takes value 1 in case of fraud and 0 otherwise. The dataset is highly unbalanced; frauds represent 0.172% of all transactions (492 frauds out of 284807 transactions). For confidentiality reason, the meaning of most variables is not revealed and the features have been transformed by means of principal components. The cardholder identifier is also not available so each transaction can be considered independent from the others. This is one of the rare datasets on fraud detection available to the community.

1.6 Publications and research activities

The list of works published during this thesis is summarized below, by category and chronological order.

Peer reviewed international journal articles

- Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. *Credit Card Fraud Detection with Alert-Feedback Interaction.* Submitted to IEEE Transactions on Neural Networks and Learning Systems.

- Andrea Dal Pozzolo, Olivier Caelen, Yann-Ael Le Borgne, Serge Waterschoot, and Gianluca Bontempi. *Learned lessons in credit card fraud detection from a practitioner perspective.* Expert Systems with Applications, 41(10):4915-4928, 2014.

Peer reviewed international conference papers

- Andrea Dal Pozzolo, Olivier Caelen, and Gianluca Bontempi. *When is undersampling effective in unbalanced classification tasks?*. In European Conference on Machine Learning. ECML-KDD, 2015.
- Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. *Calibrating Probability with Undersampling for Unbalanced Classification.* In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015.
- Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. *Credit Card Fraud Detection and Concept-Drift Adaptation with Delayed Supervised Information.* In Neural Networks (IJCNN), The 2015 International Joint Conference on. IEEE, 2015.
- Andrea Dal Pozzolo, Reid A. Johnson, Olivier Caelen, Serge Waterschoot, Nitesh V Chawla, and Gianluca Bontempi. *Using HDDT to avoid instances propagation in unbalanced and evolving data streams.* In Neural Networks (IJCNN), The 2014 International Joint Conference on. IEEE, 2014.
- Andrea Dal Pozzolo, Olivier Caelen, Serge Waterschoot, Gianluca Bontempi, *Racing for unbalanced methods selection.* Proceedings of the 14th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL), IEEE, 2013

Peer reviewed international conference poster

- Andrea Dal Pozzolo, Olivier Caelen, and Gianluca Bontempi. *Comparison of balancing techniques for unbalanced datasets.* PhD Student Forum of the International Conference on Data Mining (ICDM), IEEE, 2012.

Research visits and grants

During my PhD I've visited the following research groups:

- Data, Inference Analytics, and Learning (DIAL) Lab at the University of Notre Dame, Indiana, USA in November 2013 and May 2014 (hosted by Prof. Nitesh V Chawla)

- Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) at Politecnico di Milano, Italy in November 2014 and from April to May 2015 (hosted by Prof. Giacomo Boracchi)

The first visit at DIAL led to the conference paper at IJCNN 2014 [19] and the visit at DEIB to the paper at IJCNN 2015 [20]. At the beginning of the PhD studies, I took also part to the organization of the International Conference in Data Mining (ICDM) 2012.

In the last year of the PhD I received two travel grants: i) ULB CCCI travel grant for the second research visit at DEIB (€1500) and ii) INNS travel grant for the IJCNN 2015 conference ($800).

1.7 Financial support and project objective

The work presented in this thesis was funded by the *Doctiris* project titled "Adaptive real time machine learning for credit card fraud detection", supported by Innoviris which is the Brussels institute for the encouragement of scientific research and innovation. The objective of the 4 years project was to design, assess and validate a machine learning framework able to calibrate in an automatic, real-time and adaptive manner the fraud detection strategy of the industrial partner. The academic partner is the Machine Learning Group, from the Computer Science Department of the Université Libre de Bruxelles. The business partner is Worldline S.A., which is a company based in Brussels and leader in e-payment services.

FIGURE 1.3: Partners of the *Doctiris* project "Adaptive real time machine learning for credit card fraud detection".

1.8 Outline

The thesis is organized as follows: Chapter 2 introduces the notions and tools that will be considered in the thesis. The chapter is divided in two main parts, the first (Section 2.1) provides the reader preliminary knowledge about Machine Learning and the problem of classification. The second (Section 2.2) is devoted to the problem of Fraud Detection (FD) and describes the main layers of a FDS.

Chapter 3 reviews the different approaches which have been investigated in the literature to address the challenges introduced in Section 1.4. Each section presents relevant works that have been proposed in the literature for solving different problems typical of Fraud Detection. Section 3.1 is dedicated to methods for unbalanced classification tasks and Section 3.2 presents solutions to the problem of learning in non-stationary environments. Section 3.3 reviews works on unbalanced data streams and Section 3.4 introduces algorithmic solution that have been proposed for Fraud Detection.

Chapter 4, 5 and 6 represent the main contribution of the thesis. In Chapter 4 we tackle the problem of learning in the presence of unbalanced class distribution with a focus on undersampling, how it works, how it can improve the performances of a classifier and to choose the best strategy for a given dataset. Then Chapter 5 provides the reader with an experimental comparison of several methods for unbalanced data streams. The first part (Section 5.1) investigates which solution works best on a credit card data stream, while the second (Section 5.2) illustrates an algorithm for avoiding propagation of instances along the stream. Chapter 6 presents a prototype of a realistic FDS, where the interaction between the classifier generating the alerts and investigators providing feedbacks is modeled and exploited to improve the detection.

Finally, Chapter 7 provides a summary of the results presented in the thesis and proposes future research directions. The remainder of the present chapter provides the notations that will be used throughout the following chapters.

1.9 Notation

Throughout the thesis, random variables are written with boldface letters ($\mathbf{y}$), their realizations are in normal font (y) and estimations wear a hat ($\hat{y}$). $\mathcal{P}(\mathbf{y})$ denotes the probability distribution of the random variable $\mathbf{y}$. Uppercase letters (Y) are used for sets and uppercase calligraphic font ($\mathcal{L}$) for learning algorithms.

Table 1.1: Probability Notation

$\mathcal{P}(y)$	probability that the discrete random variable $\mathbf{y}$ takes the value y, i.e., $\mathcal{P}(\mathbf{y} = y)$
$\mathcal{P}(\mathbf{y})$	probability mass distribution of the random variable $\mathbf{y}$
$\mathcal{P}(y\|x)$	the conditional probability that $\mathbf{y} = y$ knowing that $\mathbf{x} = x$
$\mathcal{P}(\mathbf{y}\|x)$	the conditional distribution function of the target $\mathbf{y}$ knowing that $\mathbf{x} = x$
$E[\mathbf{y}]$	expected value of $\mathbf{y}$.
$Var[\mathbf{y}]$	variance of $\mathbf{y}$.
$H(\mathbf{y})$	entropy of $\mathbf{y}$
$H(\mathbf{y}\|\mathbf{x})$	conditional entropy of $\mathbf{y}$ given $\mathbf{x}$
$I(\mathbf{y};\mathbf{x})$	mutual information of $\mathbf{y}$ and $\mathbf{x}$
$\|\mathrm{X}\|$	the number of elements of the set X
$\hat{y}$	statistical estimation of y
$\hat{\mathcal{P}}(y\|x)$	estimation of $\mathcal{P}(y\|x)$

Table 1.2: Learning Theory Notation

$\mathcal{L}$	Learning algorithm
$\mathbf{y} \in \mathrm{Y}$	unidimensional discrete random variable representing the output of $\mathcal{L}$
$\mathrm{Y} \subset \mathbb{R}$	the domain of the random variable $\mathbf{y}$
y	a realization of the random variable $\mathbf{y}$
$\mathbf{x} \in \mathrm{X}$	a multidimensional discrete random variable representing the inputs of $\mathcal{L}$
$\mathrm{X} \subset \mathbb{R}^n$	the domain of the random variable $\mathbf{x}$
n	dimension of the input space X
x	a realization of $\mathbf{x}$
T_N	the training set, composed of N realizations $(x_i, y_i), i \in \{1, 2, ..., N\}$
N	number of samples in the training set
x_i	the i^{th} realization of the variable $\mathbf{x}$ in T_N
$\Lambda \subset \mathbb{R}^d$	Model parameter space.
$\theta \in \Lambda$	Model parameter vector.
Λ^θ	Class of models with parameters θ.
$h(x, \theta) \in \Lambda^\theta$	Prediction model with inputs x and parameters θ.
$L(y, h(x, \theta))$	Loss function.
$R_{\mathrm{emp}}(\theta)$	Empirical risk.
G_N	Generalization error.

Table 1.3: Fraud Detection Notation.

k	Number of alerts that investigators are able to validate everyday.
A_t	Alerts raised at day t, where $\|\mathrm{A}_t\| = k$.
$\mathrm{P}_k(t)$	Precision on A_t.
$\mathrm{D}_{t-\delta}$	Delayed supervised couples available at day t and occurred at $t - \delta$.
F_t	Feedbacks at t: recent supervised couples available at day t.
B_t	Batch of transactions available at day t.
δ	Number of days for which feedbacks are provided.
M	Number of days for which delayed samples are provided.

Chapter 2

Preliminaries

Machine Learning plays an important role in Data Driven Fraud Detection System, so in this chapter we will first introduce the reader to the problem of learning from data before moving to the specific application domain. In particular we will focus on the problem of learning with supervised data, a.k.a. supervised learning, where the learning algorithm is trained on labeled examples. Section 2.1 introduces the basics of learning from annotated data that we will later build upon in future chapters. Section 2.2 is devoted to the specificities of the Fraud Detection application.

2.1 Machine Learning

Machine Learning plays a key role in many scientific disciplines and its applications are part of our daily life. It is used for example to filter spam email, for weather prediction, in medical diagnosis, product recommendation, face detection, fraud detection, etc. Machine Learning (ML) studies the problem of learning, which can be defined as the problem of acquiring knowledge through experience. This process typically involves observing a phenomenon and constructing a hypothesis on that phenomenon that will allow one to make predictions or, more in general, to take rational actions. For computers, the experience or the phenomenon to learn is given by the data, hence we can define ML as the process of extracting knowledge from data [29].

Machine learning is closely related to the fields of Statistics, Pattern Recognition and Data Mining [7]. At the same time, it emerges as a subfield of computer science and gives special attention to the algorithmic part of the knowledge extraction process. In summary, the focus of ML is on algorithms that are able to learn automatically the patterns hidden in the data.

This thesis is about supervised learning, where ML algorithms are trained on some annotated data (the training set) to build predictive models, or *learners*, which will enable us to predict the output of new unseen observations. It is called *supervised* because the learning process is done under the supervision of an output variable, in contrast with *unsupervised learning* where the response variable is not available.

Supervised learning assumes the availability of labeled samples, i.e. observations annotated with their output, which can be used to train a learner. In the training set we can distinguish between input features and an output variable that is assumed to be dependent on the inputs. The output, or response variable, defines the class of observations and the input features are the set of variables that have some influence on the output and are used to predict the value of the response variable. Depending on the type of output variable we can distinguish between two types of supervised task: i) classification and ii) regression. The first assumes a categorical output, while the latter a continuous one. Fraud detection belongs to the first type since observations are transactions that can be either genuine or fraudulent, while in other problems such as stock price predictions the response is a continuous variable. On the other hand, in both classification and regression tasks, input features can include both quantitative and qualitative variables. In this thesis we will also refer to qualitative variables as categorical, discrete and factor variables.

2.1.1 Formalization of supervised learning

Let x be the realization of a input random vector $\mathbf{x}$ defined in the input domain X, and y denote the output value of the response variable $\mathbf{y}$, taking its values in the output domain Y. In supervised learning we have four main players [30]:

- G, the *generator* of input vectors $x \in \mathrm{X} \subset \mathbb{R}^n$ from an unknown probability distribution $F_{\mathbf{x}}(x)$.
- S, the *supervisor* which assigns to every input x an output value $y \in \mathrm{Y} \subset \mathbb{R}$ according to the conditional probability $F_{\mathbf{y}}(y|x)$.
- T, the *training samples* which are Independent and Identically Distributed (IID) samples (x, y) drawn from the joint distribution $F_{\mathbf{x},\mathbf{y}}(x, y)$.
- LM, the *learning machine* that, given some training samples T, returns a class of parametric function (or hypothesis) $h(x, \theta), \theta \in \Lambda$, estimating S for an input x and defining a mapping between the input and output domains, where Λ defines the domain for the parameters θ.

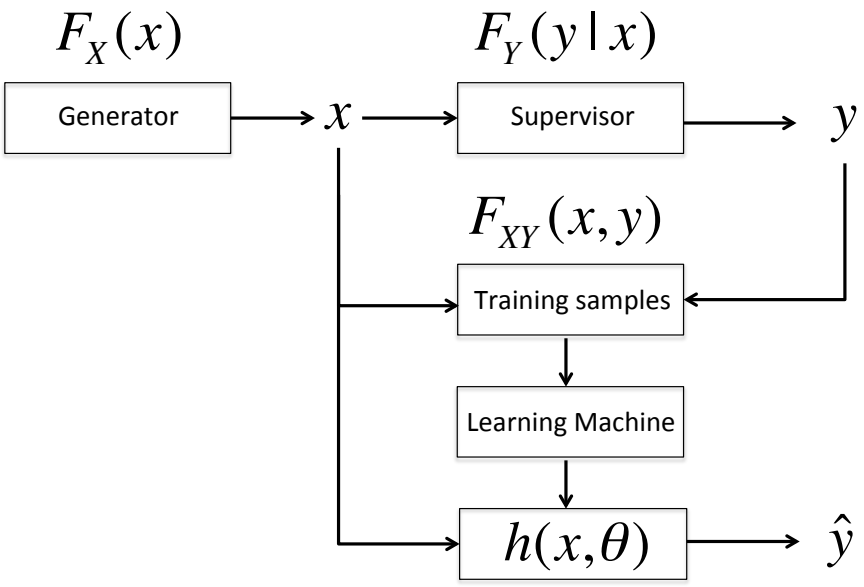


FIGURE 2.1: The players of supervised learning. A generator returns an input vector x according to $F_{\mathbf{x}}(x)$ and a supervisor associates an output value y to x on the basis of $F_{\mathbf{y}}(y|x)$. A collection of input and output values defines a training set according to $F_{\mathbf{x},\mathbf{y}}(x,y)$. The learning machine produces an hypothesis $h(x,\theta)$ on the basis of a training set and predicts the output $\hat{y}$ for a new observation.

The relationship that links the inputs x to the outputs y, denoted as $F_{\mathbf{y}}(y|x)$, is defined by S and it can be expressed as a function of the inputs, namely $f(x)$ and known as *target operator*, where $y = f(x) + \epsilon$ with ϵ being some noise. In general $F_{\mathbf{x}}(x)$, $F_{\mathbf{y}}(y|x)$ and $F_{\mathbf{x},\mathbf{y}}(x,y)$ are unknown, but some training samples T are available for LM. On the basis of T it is possible to *learn* a LM that approximate $F_{\mathbf{y}}(y|x)$. A model or learner $\mathcal{L}$ is defined as one of the many hypotheses $h(x,\theta)$ in the parameter space $\Lambda \subset \mathbb{R}^d$. It is generated by LM, and associates to each input vector x an output value or prediction $\hat{y} = h(x,\theta)$, with $\hat{y} \in \mathrm{Y}$. When $\mathrm{Y} \in \mathbb{R}$ we have a regression task, while for classification Y is a set of classes or categories. In particular, this thesis will be concerned with binary classification tasks in which $\mathrm{Y} \in \{+,-\}$, where the positive output will be also coded as 1 and the negative as 0. With respect to fraud detection, positive observations will denote fraudulent transactions and negative genuine (legitimate) ones.

LM explores multiple nested classes of hypotheses $\Lambda^1 \subset \Lambda^2 \ldots \subset \Lambda^q \subset \Lambda$ for a given T and generates different hypotheses by setting distinct parameters θ in $h(x,\theta)$. In order to find the optimal hypothesis $h(x,\theta^*)$ we have first to define a *loss function* $L(y,\hat{y})$ that measures the discrepancy between $\hat{y}$ and y, where $\hat{y} = h(x,\theta)$. Given a loss function we can then define a *risk function* $R(\theta)$ that measure how well a hypothesis $h(x,\theta)$ approximates $f(x)$ over the $\mathrm{X} \times \mathrm{Y}$ domain.

$$R(\theta) = \int_{\mathrm{X,Y}} L(y,\hat{y}) dF_{\mathbf{y}}(y|x) dF_{\mathbf{x}}(x) \tag{2.1}$$

The optimal model $h(x, \theta^*)$ is defined as the hypothesis having the parameters θ^* minimizing $R(\theta)$:

$$\theta^* = \arg\min_{\theta \in \Lambda} R(\theta) \tag{2.2}$$

The mapping $f^* : \mathrm{X} \to \mathrm{Y}$ defined by $h(x, \theta^*)$ is also known as *target function*. A training set $\mathrm{T}_N = \{(x_i, y_i), i = 1, \ldots, N\}$ is defined as a collection of N samples from T. For a given training set T_N we can compute only what is called the *empirical risk* (R_{emp}) and the hypothesis minimizing this quantity may be different from $h(x, \theta^*)$. In the Empirical Risk Minimization procedure [30] the parameters $\hat{\theta}$ are identified by minimizing $R_{emp}(\theta)$:

$$R_{emp}(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, h(x_i, \theta)) \tag{2.3}$$

$$\hat{\theta} = \arg\min_{\theta \in \Lambda} R_{emp}(\theta) \tag{2.4}$$

It is important to stress that the same hypothesis $h(x, \hat{\theta})$ achieves different $R_{emp}(\hat{\theta})$ when the training set changes ($\hat{\theta}$ depends on T_N), and the training set itself can be seen as a realization of a random variable $\mathbf{T}_N$. For this reason the accuracy of a learning algorithm is often measured using the *generalization error* (G_N), which is the mean of $R_{emp}(\hat{\theta})$ over all possible training sets:

$$G_N = E_{\mathbf{T}_N}[R_{emp}(\hat{\theta})] \tag{2.5}$$

The choice of class of hypothesis has a great impact on the error G_N that a learning algorithm $\mathcal{L}$ is able to achieve. When the model is too simple $h(x, \hat{\theta})$ could poorly approximate the target operator $f(x)$ on the training set (high $R_{emp}(\hat{\theta})$), but being general enough to fit well multiple datasets, while a too complex model could fit perfectly the training set (low $R_{emp}(\hat{\theta})$), but having poor generalization on new datasets. This interaction can lead to the problem of *underfitting* or *overfitting* and it is often referred to as the Bias-Variance trade-off of a model [31].

The bias of a model is the error from erroneous assumptions in the learning algorithm, and the variance is the error from sensitivity to small fluctuations in the training set (formal definition of bias and variance available in Appendix C). Section 2.1.3 will illustrate how G_N has both a bias and a variance component. It is commonly said that a hypothesis underfits the data when it has large bias but low variance, while it overfits the data when it has low bias but large variance. In both cases, the hypothesis gives a poor representation of the target and a reasonable trade-off needs to be found. On the basis of the available training set, the goal of the practitioner is then to search for the optimal trade-off between the variance and the bias terms.

Typically, the supervised learning procedure minimizes the error G_N by means of a two-step nested search process [32]. The inner process is known as *parametric identification*, and the outer as *structural identification*. The parametric identification step considers a single class of hypotheses Λ^j, with $j = 1, \ldots, q$, and selects a hypothesis $h(\cdot, \theta_N^j)$ with $\theta_N^j \in \Lambda^j$ by means of a learning algorithm $\mathcal{L}$. The role of the learning algorithm is to find the model that minimizes the empirical error on a given training set. Then $h(\cdot, \theta_N^j)$ is selected as the one minimizing an estimation of the generalization error $\hat{G}_N^j$ by means of a validation technique (e.g. cross validation or bootstrap).[1]

$$\hat{\theta}_N^j = \arg\min_{\hat{\theta}_N \in \Lambda^j} \hat{G}_N^j \tag{2.6}$$

The structural identification step ranges over different classes of hypotheses Λ^j, $1 \leq j \leq q$ and calls for each of them the parametric routine which returns the vector $\hat{\theta}_N^j$. The final model to be used for prediction is selected in the set $\{\hat{\theta}_N^j, j = 1, \ldots, q\}$, as the one having the smallest generalization error across all the classes of hypotheses. This last process is called *model selection*.

2.1.2 The problem of classification

A classification task can be explained as the problem of defining the association between a categorical dependent variable and independent variables that can take either continuous or discrete values. As mentioned in the previous section, the dependent variable $\mathbf{y} \in \mathrm{Y}$ is allowed to take values among a small set of K possible classes $\mathrm{Y} = \{c_1, \ldots, c_K\}$ and the input/output values of a sample (x, y) are drawn from the joint distribution $F_{\mathbf{x},\mathbf{y}}$. A classifier $\mathcal{K}$ is a hypothesis $h(x, \theta)$ that returns $\hat{y} = \hat{c} \in \mathrm{Y}$.

In general we can identify three alternative approaches to solve the classification problem:

- Estimate the posterior probabilities $\mathcal{P}(\mathbf{y} = c_k | \mathbf{x} = x)$, and then subsequently use decision theory to assign a class $\hat{y}$ to an input value x.
- Estimate the class-conditional probabilities $\mathcal{P}(\mathbf{x} = x | \mathbf{y} = c_k)$ and prior probabilities $\mathcal{P}(\mathbf{y} = c_k)$ separately and compute posterior probabilities using Bayes' theorem:

$$\mathcal{P}(\mathbf{y} = c_k | \mathbf{x} = x) = \frac{\mathcal{P}(\mathbf{x} = x | \mathbf{y} = c_k)\mathcal{P}(\mathbf{y} = c_k)}{\sum_{k=1}^{K} \mathcal{P}(\mathbf{x} = x | \mathbf{y} = c_k)\mathcal{P}(\mathbf{y} = c_k)}$$

  Then use decision theory to assign each new x to one of the classes.

[1] Note that we can compute only an estimation of G_N, because typically we have access only to a sample of all possible datasets of size N.

- Find a function $g(x)$, known as *discriminant function* [7], which maps each input x directly onto a class label $\hat{y}$ (no probability estimates needed).

Approaches that model the posterior probabilities directly are called *discriminative* models, while a *generative* approach models the class-conditional and prior probabilities for each class, and then calculates posterior probabilities using Bayes' theorem.

In the case of binary classification $K = 2$ and a popular loss function is the zero-one loss $L_{0/1}$ which assigns loss equal to one in case of wrong prediction and zero otherwise:

$$\begin{aligned} L_{0/1} : \mathrm{Y} \times \mathrm{Y} &\rightarrow \{0, 1\} \\ y, \hat{y} &\mapsto \begin{cases} 0 & \text{if } y \neq \hat{y} \\ 1 & \text{if } y = \hat{y} \end{cases} \end{aligned} \tag{2.7}$$

However, in many applications the cost of misclassification is class dependent. For example in cancer treatment, the cost of not predicting correctly a sick patient (a False Negative) is much higher than making a wrong prediction when the patient is healthy (a False Positive). Assume $\mathrm{Y} = \{1, 0\}$, where 1 denotes positive instances and 0 negative ones. Let $l_{i,j}$ be the loss (cost) incurred in deciding i when the true class is j and $p = \mathcal{P}(\mathbf{y} = 1 | \mathbf{x} = x)$. We can define the risk of predicting an instance as positive or negative as follows:

$$r^+ = (1 - p) l_{1,0} + p l_{1,1}$$

$$r^- = (1 - p) l_{0,0} + p l_{0,1}$$

Standard decision-making process based on decision theory [7, 33, 34] defines the optimal class of a sample as the one minimizing the risk (expected value of the loss function). Bayes decision rule for minimizing the risk can be stated as follows: assign the positive class to samples with $r^+ \leq r^-$, and the negative class otherwise.

$$\hat{y} = \begin{cases} 1 & \text{if } r^+ \leq r^- \\ 0 & \text{if } r^+ > r^- \end{cases} \tag{2.8}$$

As shown by Elkan [35], (2.8) is equivalent to predict a sample as positive when $p > \tau$ and the threshold τ is:

$$\tau = \frac{l_{1,0} - l_{0,0}}{l_{1,0} - l_{0,0} + l_{0,1} - l_{1,1}} \tag{2.9}$$

Typically, the cost of a correct prediction is zero, hence $l_{0,0} = 0$ and $l_{1,1} = 0$. If we set $l_{1,0} = l_{0,1} = 1$ we obtain $L_{0/1}$ and $\tau = 0.5$.

2.1.3 Bias-variance decomposition

As shown in the previous section, a binary classification problem can be solved by estimating the posterior probability and then using a threshold τ to define the class. Since the posterior probability is a continuous value, the probability estimation problem can be treated as a regression problem. In regression, the unknown conditional distribution $F_{\mathbf{y}}(y|x)$ is typically estimated using the conditional expectation $E[\mathbf{y}|x]$ [30]. In the case of binary classification with $\mathrm{Y} = \{1, 0\}$ we can write:

$$E[\mathbf{y}|x] = 1 \cdot \mathcal{P}(\mathbf{y} = 1|x) + 0 \cdot \mathcal{P}(\mathbf{y} = 0|x) = \mathcal{P}(\mathbf{y} = 1|x)$$

In this way, the classification problem can be seen as a regression problem where the output takes value in $\{0, 1\}$. Let $p = \mathcal{P}(\mathbf{y} = 1|\mathbf{x} = x)$ and $\hat{p}$ its approximation given by the estimator $\hat{\boldsymbol{p}}$. A common choice of loss function in regression is the quadratic loss given by $L(p, \hat{p}) = (p - \hat{p})^2$. In the case of the quadratic loss function, it can be shown that the generalization error G_N can be decomposed into a bias and a variance term [36].

$$G_N = E_{\mathbf{T}_N}[(p - \hat{\boldsymbol{p}})^2] = Bias[\hat{\boldsymbol{p}}]^2 + Var[\hat{\boldsymbol{p}}] \tag{2.10}$$

where $Bias[\hat{\boldsymbol{p}}] = E_{\mathbf{T}_N}[\hat{\boldsymbol{p}}] - p$ and $Var[\hat{\boldsymbol{p}}] = E_{\mathbf{T}_N}[(\hat{\boldsymbol{p}} - E_{\mathbf{T}_N}[\hat{\boldsymbol{p}}])^2]$ (derivation available in Appendix C). The first term $Bias[\hat{\boldsymbol{p}}]^2$ measures the "bias" of the model, it gives an idea of how far the average of our estimator $\hat{\boldsymbol{p}}$ is from the true value p. The "variance" term ($Var[\hat{\boldsymbol{p}}]$) quantifies how much $\hat{\boldsymbol{p}}$ vary around its mean for different training set T_N. It therefore quantifies the sensitivity of the model prediction for a given training set. The variance is independent of the true probability p, and is null for the classifier that has the same predictions across all training datasets. Domingos [37] provides a more general bias-variance decomposition that works also for the $L_{0/1}$.

Typically, complex models have high variance and a low bias while simple models have low variance and a high bias. This is because complex models can approximate well the target function (low bias), but are highly affected by the variability of the training set (leading to high variance). The opposite occurs with simple models. Simple learners can show better performances than more complex ones (e.g., [38, 39]). This is because both bias and variance influence the predictive error. The optimal trade-off between bias and variance is often domain or application specific. Several works have found that ensembles of models very often outperform a single model (e.g., [40]), because averaging multiple models often (though not always) reduces the variance of single models.

Well-known ensemble methods are bagging [41] and boosting [42]. In bagging multiple hypotheses are constructed by using different bootstrap samples of the original data set. The resulting models are then combined into an ensemble to make predictions.

Breiman [41] showed that bagging allows one to transform an unbiased classifier into a nearly optimal one by reducing the variance term of (2.10). In the case of boosting, weak learners are combined to obtain accurate prediction [42] and the improvement in the generalization error over a single classifier can also be related to the bias-variance decomposition by introducing the notion of a margin [37], which measures the confidence in the prediction of the ensemble.

A powerful ensemble algorithm based on bagging is Random Forests [43]. Random Forest (RF) is an ensemble of decision trees, where each tree is trained on different bootstrap sample of the original training set and uses a random subset of all the features available. This returns a forest of decision trees that are very different from each other. Diversity in the models generating an ensemble is a key factor for variance reduction [44].

2.1.4 Evaluation of a classification problem

In a classification problem an algorithm is assessed on its overall accuracy to predict the correct classes of new unseen observations and usually $\hat{G}_N$ is assessed in terms of Mean Misclassification Error (MME). Let Y_1 be the set of positive instances, Y_0 the set of negative instances, $\widehat{\mathrm{Y}}_1$ the set of instances predicted as positive and $\widehat{\mathrm{Y}}_0$ the ones predicted as negative. For a binary classification problem it is conventional to define a confusion matrix (Table 2.1).

	Y_1	Y_0
$\widehat{\mathrm{Y}}_1$	TP	FP
$\widehat{\mathrm{Y}}_0$	FN	TN

Table 2.1: Confusion Matrix.

In the case of zero-one loss we can then define MME $= \frac{|\mathrm{FP} \wedge \mathrm{FN}|}{N}$, where operator $|\cdot|$ defines the cardinality of a set and N the size of the dataset. In the following we will write FP to denote |FP| and similarly for all the other entries of the confusion matrix. From MME we can then define the accuracy of a model as 1 - MME. However, in some situations, the accuracy is not a good measure of performance, especially in unbalanced classification problem where one class is much more frequent than the other [45, 46]. For example consider the case where we have $N = 10000$ transactions and only 1% are fraudulent (100 positives). Predicting all transactions as genuine (negative) would return to an accuracy of 0.99, but we would miss to detect all fraudulent cases.

Other classification measures based on the confusion matrix are [47, 48]:

- Precision: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$, also called True Positive Rate (TPR), Sensitivity or Hit rate
- True Negative Rate (TNR): $\frac{TN}{FP+TN}$, also called Specificity
- False Positive Rate (FPR): $\frac{FP}{FP+TN}$
- False Negative Rate (FNR): $\frac{FN}{TP+FN}$
- Balanced Error Rate (BER): $0.5(\frac{FP}{TN+FP} + \frac{FN}{FN+TP})$
- G-mean: $\sqrt{\text{Sensitivity} \times \text{Specificity}}$
- F-measure: $2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$, also called F-score or F1.

In an unbalanced classification problem, it is also well known that quantities like TPR and TNR are misleading assessment measures [49]. Imagine a classifier with TPR=99% and TNR=99% (Table 2.2), if we have 1% of positive samples, then Precision is only 0.5. Even worse, if we have only 0.1% of positives, then Precision is 0.09.

(a) Class percentage

	Y_1	Y_0
$\widehat{Y}_1$	99%	1%
$\widehat{Y}_0$	1%	99%

(b) Frequency

	Y_1	Y_0
$\widehat{Y}_1$	99	99
$\widehat{Y}_0$	1	9801
	100	9900

TABLE 2.2: Confusion Matrix of a classifier with TPR=99% and TNR=99% when we have 100 positive instances in 10000 observations.

BER may be inappropriate too because of different costs of misclassification false negatives and false positives. Precision and Recall have opposite behavior, having high Precision means bad Recall and vice versa. F-measure gives equal importance to Precision and Recall into a metric ranging between 0 and 1 (the higher the better). F-measure and G-mean are often considered to be relevant measures in unbalanced problem [21, 50–52]. In general these measures can be computed only once a confusion matrix is available, which means that their values depend on the threshold used for classification defined by (2.9). Changing the threshold corresponds to use different misclassification costs.

The Receiving Operating Characteristic (ROC) curve [53, 54] is a well-know assessment technique that allows evaluating the performance of a classifier over a range of different thresholds. It is obtained by plotting TPR against FPR (see Figure 2.2), where each point of the curve corresponds to a different classification threshold. A classifier $\mathcal{K}$ is said to be more accurate than a classifier $\mathcal{W}$ in the ROC space only if the curve of $\mathcal{K}$ always dominates the curve of $\mathcal{W}$. The best classifier corresponds to the point (0,1) in the ROC

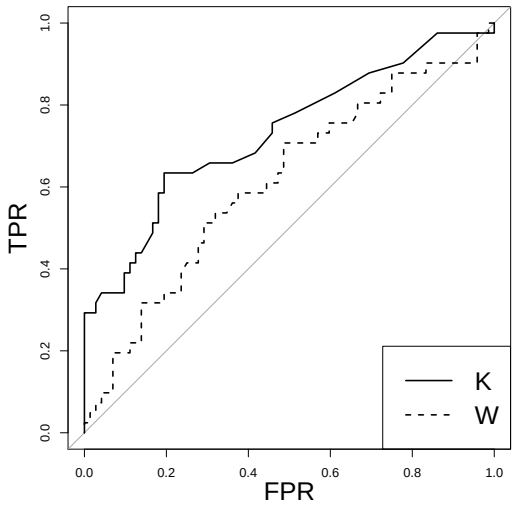


FIGURE 2.2: The Receiving Operating Characteristic (ROC) curve for two classifiers $\mathcal{K}$ and $\mathcal{W}$. The gray line represents the performance of a random model.

space (no false negatives and no false positives), while a classifier predicting at random would have performances along the diagonal connecting the bottom left corner to the top right. When there is not a clear winner (e.g. classifier $\mathcal{K}$ dominates $\mathcal{W}$ only in one part of the ROC space), the comparison is usually done by calculating the Area Under the ROC Curve (AUC). AUC is also a well-accepted measure for unbalanced datasets and it has become the *de facto* standard in classification [19, 51, 55–57].

However, AUC is insensitive to class priors since both TPR and FPR do not change with a different class ratio. Precision-Recall (PR) curves are instead sensitive to changes in the class distribution, but there is there a strong connection between ROC and PR curves. A curve dominates in ROC space if and only if it dominates in PR space, and algorithms that optimize the area under PR curve are guaranteed to optimize the area under ROC curve [58].

When evaluating the output of a classifier it is also important to assess the quality of the estimated probabilities [59]. A well-known measure of quality is Brier Score (BS) [60]. BS is a measure of average squared loss between the estimated probabilities and the actual class value. It allows evaluating how well probabilities are calibrated, the lower the BS the more accurate are the probabilistic predictions of a model. Let $\hat{\mathcal{P}}(y_i|x_i)$ be the probability estimate of sample x_i to have class $y_i \in \{1, 0\}$, BS is defined as:

$$\text{BS} = \frac{1}{N} \sum_{i=1}^{N} [y_i - \hat{\mathcal{P}}(y_i|x_i)]^2 \tag{2.11}$$

2.2 Credit Card Fraud Detection

2.2.1 Fraud Detection System working conditions

In this section we describe the working conditions of a real-world Fraud Detection System (FDS). Figure 2.3 describes the hierarchy of layers in a FDS, each controlling whether the transactions is genuine or should be rather reported as a fraud: i) Terminal, ii) Transaction Blocking Rules, iii) Scoring Rules, iv) Data Driven Model, v) Investigators. The first four elements of the FDS are fully automatized, while the last one requires human intervention and it is the only non-automatic and offline part of the FDS. Automatic tools have to decide whether the transaction request (or the transaction attempt) has to be approved in Real Time (i.e., decision has to be taken immediately) or in Near Real Time (i.e. decisions can be taken in a short time). Blocking and scoring rules are Expert Driven Rules (EDR), i.e. rules designed by investigators based upon their experience. On the contrary, the DDM uses annotated transactions as source of information to extract knowledge about fraudulent and genuine patterns. In the following we will explain the role played by each component of the FDS.

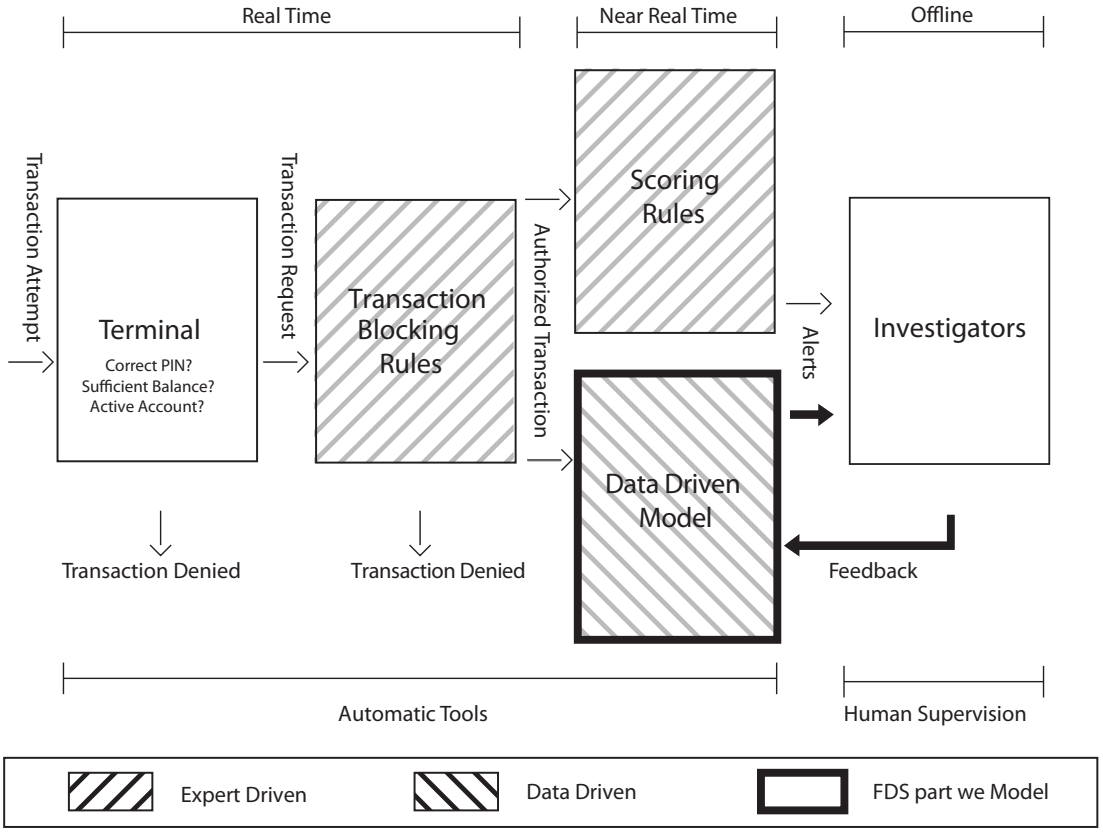


FIGURE 2.3: The layers of a FDS. In this thesis we focus only on the data driven part.

2.2.1.1 FDS Layers:

The *Terminal* represents the first layer of conventional security checks in a FDS. At this stage security checks such as correct PIN code, number of attempts, status of the card (active or blocked), sufficient balance and expenditure limit provide a first filter of the transactions that can get through [61]. These conditions are evaluated in real time and if any of them is not satisfied, the transaction is denied. All transactions passing this first filter raise a transaction request to the second layer controlled by the blocking rules.

Transaction Blocking Rules are designed by experienced investigators and they can block a transaction request before it is authorized. These rules operate in Real Time, and are very precise, they deny a transaction request only if it clearly represents a fraud attempt.[2] Blocking Rules are *if-then (-else)* rules that associate the class fraud or genuine to a transaction when specific conditions are met. An example can be the following: "IF a PIN-based transactions is done in Europe AND in Asia within 5 min from the same cardholder THEN deny the transaction", since the customer cannot physically be in Europe and Asia over a short time interval.[3] Transaction requests passing Blocking Rules are authorized and become transactions (the payment is executed). Before each authorized transaction is sent to the successive security layer of FDS it is enriched with aggregated information (e.g. average expenditure, number of transactions in the same day of the same cardholder, etc.). The resulting *feature vector* describing the transaction is then analyzed by both Scoring Rules and DDMs. These controls are performed in Near Real Time (typically less than 6s [61]) since there is no need to provide an immediate response given that the transaction has been already authorized.

Scoring Rules are EDR defined by investigators based upon their experience that act like a rule-based classifier. They assign a score to each authorized transaction: the larger the score, the more likely the transaction is to be a fraud. In practice scoring rules contain simple, human-understandable conditions (e.g. IF Internet transaction in fiscal paradise and amount $>$ 1000\$, THEN fraud score = 0.9) and as such can be easily designed by investigators or other experts. These rules are easy to understand and fast to deploy, but require manual revision when their performance drops. As a consequence, these rules can be expensive to maintain.

Data Driven Model (DDM) layer relies on a predictive model to assign a fraud score to each transaction. Usually this phase uses ML algorithms that return for each transaction an estimate of the probability to be a fraud. These algorithms can learn complex correlations in the data using large volume of data with high dimensionality. They are

[2] Cards should be blocked only if there is a high risk of frauds, because it prevents the cardholder to make any future payments.

[3] Note that this is just an illustrative example, blocking rules are confidential information.

usually more robust than EDR, but most of them are black box, i.e. it is not possible to convert them into rules that are easy to interpret. DDMs are able to consider all the information associated to a transaction, while EDR returns conditions based on few features of the transactions.

Investigators are fraud experts that check fraud alerts, i.e. transactions that have received a high fraud score by either EDR or DDM. These suspicious transactions are displayed in a Case Management Tool (CMT) where investigators can see the associated fraud score, check where they come from (EDR or DDM) and annotate them as genuine or fraud after verification. In a realistic scenario only few alerts can be checked given a limited number of investigators [62]. For this reason, they typically investigate only transactions with the highest fraud score. The role of investigators is to contact the cardholder, check if the transaction is fraudulent or not and annotate it with its correct label. This process can be long, tedious, and at the same time the number of transactions to check is usually very large, so it is important to minimize false alerts. The labeling process generates annotated transactions that are used by the DDM. In the following we use the term *feedbacks* to refer to these labeled transactions.

2.2.1.2 Supervised Information

Investigators' feedbacks are the most recent supervised information available to the FDS and they represent a tiny part of all transactions processed everyday [20]. Additional labeled transactions are provided by cardholders reporting unauthorized transactions [20, 61]. However, the number of customers reporting frauds not detected by the FDS is usually small and hard to model, since cardholders have different habits when it comes to check the transcript of credit card transactions given by the bank. We do not know the class of all the other transactions that have not been checked, they can be either genuine or frauds missed by the FDS and ignored by the cardholders. However, after a certain time period when no further missed frauds are reported, we can assume that unchecked transactions are genuine. These transactions can then be used to train a new DDM.

2.2.1.3 System Update

Expert driven systems are updated manually, while the data-driven component is updated automatically. Alerts generated today define the feedbacks that will be used to train an algorithm that detects the frauds of tomorrow. This means that Alert-Feedback Interaction (AFI) governs the type of information available to the algorithm that regulates the data driven phase.

Typically, DDMs require a large set of samples to train an accurate model, so the ML algorithms used in the data driven layers are trained at the end of the day, when a sufficient batch of feedbacks are available. Feedbacks of alerted transactions are given by investigators during the day, but only at the end of the day we have all the feedbacks for the alerts generated by the FDS. Therefore, the DDM is usually updated everyday at midnight and the detection model is then applied to all the transactions occurring in the next days.

The focus of the thesis is to improve the DDM part of the FDS and to model the interaction between Data Driven Methods based on ML and investigators. In particular, we will consider a scenario where the DDM is the only element of the FDS responsible for the alerts given to fraud experts and the algorithms are able to learn from the feedback provided. In the remainder of the thesis we will use the term FDS to indicate only the FDS layers we model: DDM and investigators.

2.2.2 Features augmentation

When a transaction is authorized, it is entered in a database containing few variables such as cardholder ID ($CARD\_ID$), amount, datetime, merchant, etc. Starting from these features it is possible to compute new variables that can describe the cardholder behaviour. This process is called feature augmentation, because the ultimate goal is to add to the original feature new informative features. Standard feature augmentation consists into computing variables such as average expenditure of the cardholder in the last week/month, number of transactions in the same shop, number of daily transactions, etc. [18, 62–65]. Van Vlasselaer et al. [61] use also network features to consider the role of the cardholder in the network of transactions.

These new features are also called *aggregated* features as they provide an aggregated view of the cardholder behaviour over a certain time period. It is important to include aggregated features for the DDM to learn the behavior of the customer over time and detect anomalous transactions w.r.t. typical customer usage of the credit card. Transaction aggregation is computationally expensive; therefore aggregated features are often computed for each cardholder offline using historical transactions (e.g. transaction of the same cardholder in the previous month). Once aggregates variable are available they are merged with the original feature to define a feature vector describing the transaction that is then used to train the DDM. After transaction authorization and features augmentation, each feature vector is scored by both DDMs and Scoring Rules which operate in Near-Real Time.

In a testing environment, we should remove *CARD_ID* from the feature vector. A model that receives as input the variable *CARD_ID* may have performances too optimistic on future frauds from the same cardholder if not removed from the dataset. On the contrary, in a real working environment, a card is blocked after the first fraud is found, so we cannot see future transactions from the same card. To replicate the real setting we could remove all the transactions of the *compromised* card, but this would reduce the number of frauds available for testing our algorithm, worsening the class unbalance problem in our dataset. Another option is to remove the *CARD_ID* variable from the feature vector. In this case, each fraud can be treated independently and the FDS cannot leverage *CARD_ID* for detecting frauds. Note that *CARD_ID* is used to compute aggregated variables, so we can safely remove *CARD_ID* only once the new variables are included in the feature vectors. These new features contain information about the cardholder, so there is not loss of information in removing *CARD_ID* after features augmentation.

2.2.3 Accuracy measure of a Fraud Detection System

Choosing a good performance measure is not a trivial task in the case of fraud detection. Among others, fraud detection must deal with the following challenges: i) unbalanced class sizes, ii) cost structure of the problem (the cost of a fraud is not easy to define), iii) time to detection (a card should be blocked as soon as it is found victim of fraud, quick reaction to the appearance of the first can prevent other frauds), iv) errors in class labels (quantify unreported frauds), v) reputation's cost for the company, etc.

As already mentioned in Section 2.1.4, standard classification measure such as MME, BER, TPR and TNR are misleading assessment measures in unbalanced class problem [49]. A well-accepted measure for unbalanced classification is the Area Under the ROC Curve (AUC) [56]. This metric gives a measure of how much the ROC curve is close to the point of perfect classification. Hand [66] considers standard calculation of the AUC as inappropriate, since this translates into making an average of different misclassification costs for the two classes. An alternative way of estimating AUC is based on the use of the Mann–Whitney statistic and consists in ranking the transactions according to the posterior probability to be fraudulent and measuring the likelihood that a fraud ranks higher than a genuine transaction [67]. By adopting the rank-based formulation of AUC we can avoid the problem raised by Hand of using different probability thresholds.

In many FDS (e.g. [68–70]), cost-based measures are defined to quantify the monetary loss due to fraud [71] by means of a cost-matrix that associates a cost to each entry of the confusion matrix. Elkan [35] claims that it is safer to assess cost-sensitive problem in terms of benefit (inverse of cost), since there is the risk of using different baselines

when using a cost-matrix to measure overall cost. To avoid this problem, normalized cost or savings [70] are used to assess the performance w.r.t. the maximum loss. When defining a cost measure, one could consider the cost of a FN fixed or dependent on the transaction amount. In the first case each fraud is equally costly, while in the latter the cost is example dependent. An argument for using fixed cost is to give equal importance to small and larger frauds (fraudsters usually test a card with small amounts), while transaction-dependent cost allows one to quantify the real loss that a company has to face.

In the transaction-dependent case, the cost of a missed fraud (FN) is often assumed to be equal to the transaction amount [35, 71], because it has to be reimbursed to the customer. Cost of correct or false alerts is considered to be equivalent to the cost of a phone call, because the investigators make a phone call to the cardholder to verify if it is the case of a false alert or a real fraud. The cost of a phone call is negligible compared to the loss that occurs in case of a fraud. However, when the number of false alerts is too large or the card is blocked by error, the impossibility to make transactions can translate into big losses for the customer.

The cost should also include the time taken by the detection system to react. The shorter is the reaction time, the larger is the number of frauds that it is possible to prevent. Typically, once fraudsters successfully perpetuate a fraud, then they try to spend all the money available on the card. As a consequence, when evaluating a FDS we should also consider the spending limit of each card: i.e. detecting a fraud on a card having a large spending limit (e.g. corporate cards) result in higher savings than detecting a fraud on a card having a small spending limit [72]. For all these reasons, defining a cost measure is a challenging problem in credit card detection and there is not agreement on which is the right way to measure the cost of frauds.

The performance of a detection task (like fraud detection) is not necessarily well described in terms of classification [73]. In a detection problem what matters most is whether the algorithm can rank the few useful items (e.g. frauds) ahead of the rest. In a scenario with limited resources, fraud investigators cannot revise all transactions marked as fraudulent from a classification algorithm. They have to put their effort into investigating transactions with the highest risk of fraud, which means that the detection system is asked to return the transactions ranked by their posterior fraud probability. The goal then is not to predict accurately each class, but to return a correct rank of the minority classes.

In this context a good detection algorithm should be able to give a high rank to relevant items (frauds) and low score to non-relevant. A well-known detection measure is Average Precision (AP) [73]. Let N^+ be the number of positive (fraud) case in the original dataset

and TP_k be the number of true positives in the first k ranked transactions ($TP_k \leq k$). Let us denote Precision and Recall at k as $\mathrm{P}_k = \frac{TP_k}{k}$ and $\mathrm{R}_k = \frac{TP_k}{N^+}$. We can then define Average Precision as:

$$\mathrm{AP} = \sum_{k=1}^{N} \mathrm{P}_k(\mathrm{R}_k - \mathrm{R}_{k-1}) \tag{2.12}$$

where N is the total number of observations in the dataset. The better the rank, the greater the AP and the optimal algorithm that ranks all the frauds ahead of the legitimates has AP = 1. Note that AP is also an estimate of the area under the Precision-Recall curve [74].

As explained in Section 2.2.1, each time a fraud alert is generated by the detection system, it has to be checked by investigators before proceeding with actions (e.g. customer contact or card stop). Given the limited number of investigators, it is crucial to have the best ranking within the maximum number k of alerts that they can investigate. In this setting it is important to have the highest Precision within the first k alerts, namely P_k. Note that, while AUC and AP give a measure of the quality of the ranking on the whole datasets, P_k focus only on a subset of k transactions. Precision at k (P_k) is also called alert Precision and it has emerged as a standard accuracy measure of FDSs [18, 20, 64].

Assessment of two FDSs (e.g. existing versus new version) can hide undesired bias. When evaluating between choosing an existing FDS and a new solution, Hand [75] warns against the bias that favors the old FDS. The data used to train a new FDS depend on the alerts and detection given by the FDS in place at the moment of training, i.e. we can train a new FDS only on the frauds discovered by the old FDS and the undetected ones cannot be used for training / evaluation. Therefore, there is a selection bias in the data collection process that is due to the performance of the previous FDS, as a result the evaluations are biased in favor of the existing system. This issue is similar to the problem of evaluating classifiers in the presence of Alert-Feedback Interaction (see Chapter 6), where we know the feedbacks only of the learner requesting the labels. The performance evaluation makes sense only conditional on the learner generating the alerts, so it is incorrect to compare two alternative algorithms when only one of the two requests the feedbacks.

Chapter 3

State-of-the-art

This chapter presents a review of the approaches that have been adopted for dealing with the challenges of a Data Driven FDS presented in Section 1.4. Section 3.1 provides an overview of state-of-the-art methods for unbalanced classification and Section 3.2 reviews the principal adaptation techniques proposed for the problem of learning in non-stationary distribution. Then in Section 3.3 we present methods that address the problem of class unbalance in evolving data streams. Finally, Section 3.4 presents state-of-the-art algorithmic solutions proposed for credit card fraud detection.

3.1 Techniques for unbalanced classification tasks

Learning from unbalanced datasets is a difficult task since most learning algorithms are not designed to cope with a large difference between the number of cases belonging to different classes [22]. There are several methods that deal with this problem and we can distinguish between methods that operate at the data and algorithmic levels [76]. At the data level, the unbalanced strategies are used as a pre-processing step to rebalance the dataset or to remove the noise between the two classes, before any algorithm is applied. At the algorithmic level, algorithms are themselves adjusted to deal with the minority class detection. Data level methods can be grouped into five main categories: sampling, ensemble, cost-based, distance-based and hybrid. Within algorithmic methods instead we can distinguish between: i) classifiers that are specifically designed to deal with unbalanced distribution and ii) classifiers that minimize overall classification cost. The latter are known in the literature as cost-sensitive classifiers [35]. Both data and algorithm level methods that are cost-sensitive target the unbalanced problem by using different misclassification costs for the minority and majority class. At the data level, cost-based methods sample the data to reproduce the different costs associated to each

class (see *translation theorem* [77]). Cost-sensitive classifiers instead directly minimize the costs by using cost specific loss function. Alternatively, *wrapper* methods can be used to convert a cost-insensitive (or cost-blind) classifier into a cost-sensitive ones (e.g. Metacost [78]).

All the methods presented in the following section will discuss the unbalanced problem as referred to *between* class imbalance, i.e. imbalance in class frequency. However, class imbalance can exist also *within* the class [79, 80] (due to small clusters within one class), and this problem is often linked to the presence of rare cases [81]. Within-class imbalances and rare cases are closely related to the problem of *small disjuncts*, which hinder classification performance [80, 82–84]. Small disjuncts are rules that cover a small cluster of examples resulting from concepts that are underrepresented [46, 85, 86]. For an in-depth analysis of other issues related to unbalanced classification we invite the reader to have a look at [87].

3.1.1 Data level methods

Sampling methods

Typically, sampling methods are used to rebalance the datasets, because studies have shown that standard classifiers have better performances when trained on a balanced training set [88–90]. Sampling techniques do not take into consideration any class information in removing or adding observations, yet they are easy to implement and to understand.

Undersampling [91] consists in downsizing the majority class by removing observations at random. In an unbalanced problem it is realistic to assume that many observations of the majority class are redundant and that by removing some of them at random the resulting distribution should not change much. However, the risk of removing relevant observations from the dataset is still present, since the removal is done in an unsupervised manner. In practice, this technique is often adopted since it is simple and speeds up the learning phase.

Oversampling [91] consists in up-sizing the small class at random decreasing the level of class imbalance. By replicating the minority class until the two classes have equal frequency, oversampling increases the risk of overfitting [91] by biasing the model towards the minority class. Other drawbacks of this approach are that it does not add any new *informative* minority examples and that it increases the training time. This can be particularly ineffective when the original dataset is fairly large.

SMOTE [92] oversamples the minority class by generating synthetic examples in the neighborhood of observed ones. The idea is to form new minority examples by interpolating between samples of the same class. This has the effect of creating clusters around each minority observation. By creating synthetic observations the classifier builds larger decision regions that contain nearby instances from the minority class. SMOTE has shown to improve the performances of a base classifier in many applications [92], but it has also some drawbacks. Synthetic observations are generated without considering neighboring examples, leading to an increase of overlap between the two classes [93]. Borderline-SMOTE [94] and ADASYN [95] have been proposed to overcome this problem.

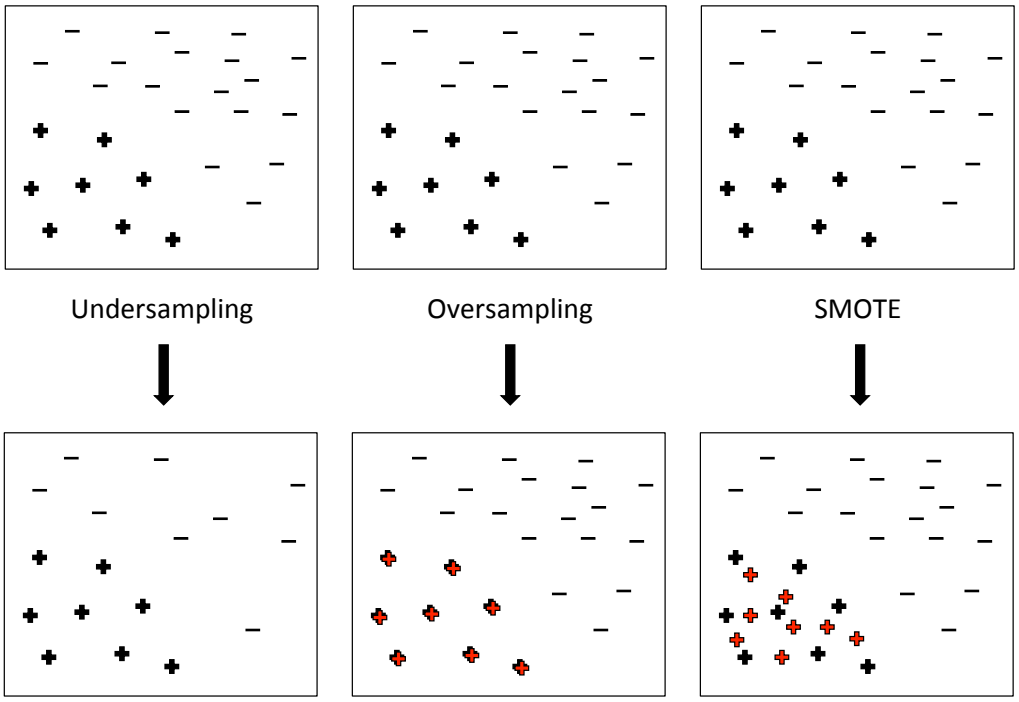


FIGURE 3.1: Resampling methods for unbalanced classification. The negative and positive symbols denote majority and minority class instances. In red the new observations created with oversampling methods.

Cost-based methods

Cost proportional sampling [35] consists into sampling training instances from the majority and minority classes by taking into consideration the ratio of the misclassification costs. Let $l_{i,j}$ denote the misclassification cost of class j predicted as i, so that the cost of a FP and a FN is written as $l_{1,0}$ and $l_{0,1}$. When we want to keep all minority examples, then the number of majority instances should be multiplied by $\frac{l_{1,0}}{l_{0,1}}$. If we assume that the cost of a FP is lower, i.e. $l_{1,0} < l_{0,1}$, this boils down to undersampling the majority

class. Alternatively, cost proportional sampling can be achieve by replicating observations from the minority class $\frac{l_{0,1}}{l_{1,0}}$ times (oversampling). As in the case of oversampling, replicating samples from the minority class may induce overfitting [77].

Costing [77] adopts "rejection sampling" to select the instances in the final dataset. Each instance in the original training set is drawn once, and accepted into the sample with the accepting probability $\frac{l_{j,i}}{Z}$ where Z is an arbitrary constant such that $Z \geq \max l_{j,i}$. When $Z = \max l_{j,i}$, this is equivalent to keeping all examples of the rare class, and sampling the majority class without replacement according to $\frac{l_{1,0}}{l_{0,1}}$.

Distance-based methods

The following methods make use of distance measures between input points either to undersample or to remove noisy and borderline examples of each class. These methods are very time consuming since they require computing distances between observations.

Tomek link [96] removes observations from the negative class that are close to the positive region in order to return a dataset that presents a better separation between the two classes. Let us consider two input examples x_i and x_j belonging to different classes, and let $d(x_i, x_j)$ be their distance. A (x_i, x_j) pair is called a *Tomek link* if there is no example x_k, such that $d(x_i, x_k) < d(x_i, x_j)$ or $d(x_j, x_k) < d(x_i, x_j)$. If two examples form a Tomek link, then one of these examples is noisy or both are borderline. Negative instances that are Tomek links are then removed reducing the majority class. This method is particularly useful in noisy datasets as it removes those samples for which noise can lead to a misclassification [97].

Condensed Nearest Neighbor (CNN) [98] is used to select a subset S from the original unbalanced set T which is consistent with T in the sense that S classifies T correctly with the one-nearest neighbor rule (1-NN). The idea behind this implementation of a consistent subset is to eliminate the examples from the majority class that are distant from the decision border, since these sorts of examples might be considered less relevant for learning. Since noisy samples are likely to be misclassified, many of them will be added to the S set which means CNN rule is extremely sensitive to noise [99]. Moreover, noisy training data will misclassify several of the subsequent test examples.

One Sided Selection (OSS) [100] is an undersampling method resulting from the combination of Tomek links followed by the application of CNN. Tomek links are used as an undersampling method and removes noisy and borderline majority class examples. Borderline examples can be considered unsafe since a small amount of noise can make them fall on the wrong side of the decision border [97]. CNN aims to remove examples from

the majority class that are distant from the decision border. The remaining examples, i.e. safe majority class instances and all minority class examples, are used for learning.

Edited Nearest Neighbor (ENN) [101] removes any example whose class label differs from the class of at least two of its three nearest neighbors. In this way majority examples that fall in the minority region are removed and likewise isolated minority examples are removed. To avoid the risk of losing relevant minority examples ENN is edited to remove only negative examples that are misclassified by their 3 nearest neighbors.

Neighborhood Cleaning Rule (NCL) [89] modifies the ENN method by increasing the role of data cleaning. Firstly, NCL removes negative examples that are misclassified by their 3-nearest neighbors. Secondly, the neighbors of each positive example are found and the ones belonging to the majority class are removed. Since this algorithm removes noisy instances, as well as close-border points, the decision boundary becomes smoother with a consequent reduction of the risk of overfitting [97].

Other hybrid strategies can be easily created by combining sampling, ensemble or distance-based techniques [27].

3.1.2 Algorithm level methods

Algorithm oriented methods are essentially a modification (or extension) of existing classification algorithms for unbalanced tasks. Depending on their applications we distinguish between imbalanced learning and cost-sensitive learning. In first case, the goal is to improve accuracy of the minority class, while in the second case the objective is to minimize the cost associated to the classification task.

Imbalance learning

Standard decision trees such as C4.5 [102] use Information Gain (IG) as splitting criteria in each node of the tree. However, this splitting criterion returns rules that are biased towards the majority class. Liu et al. [57] use the Class Confidence Proportion (CCP) metric for splitting in presence of class imbalance, while Cieslak and Chawla [103] suggest splitting with Hellinger Distance (HD). They show that HD is skew-insensitive and the proposed HDDT has better performance compared to standard C4.5. Other studies have also reported the negative effect of skewed class distributions not only on decision trees [21, 82], but also for Neural Networks [82, 104], k-Nearest Neighbor (kNN) [100, 105, 106] and SVM [107, 108].

A SVM optimized in terms of F-measure is presented by Callut and Dupont [109], while Li et al. [110] use SVM with RBF kernels as base classifier for AdaBoost. In the family of lazy learning classifiers, Liu and Chawla [111] propose a kNN weighting strategy designed for handling the problem of class unbalance. The algorithm, called CCW-kNN (Class Confidence Weights kNN), is able to correct the inherent bias towards the majority class in existing kNN classifiers.

Association rule mining with class unbalance is often achieved by specifying multiple support levels for each class to account for the difference in class frequency [112]. Within the family of rule-based classifiers, Verhein and Chawla [113] developed a classifier called SPARCCC that is specifically designed for unbalanced classification. In all these algorithms the general idea is to modify the original classifier in order to learn better patterns from the minority class. Weiss [114] argues that these algorithmic solution should be preferred to data level methods cause they are able to deal with the class unbalance directly without biasing the classifier towards one class.

Following the great success of ensemble learning in machine learning, lots of ensemble strategies have been proposed for imbalanced learning, with Bagging [41] and Boosting [115] being the most popular methods to aggregate classifiers. Usually, ensemble methods combine an unbalanced strategy with a classifier to explore the majority and minority class distribution. BalanceCascade [116] is a supervised strategy to under sample the majority class. This method iteratively removes the majority class instances that are correctly classified by a boosting algorithm. The idea is that observations of the majority class that are easy to classify are redundant and that by removing them the algorithm can concentrate on the hard cases. The drawback is that a classification algorithm has to be applied several times to reduce the majority class leading to an increase in computational needs.

EasyEnsemble [116] and UnderBagging [117] combine different models that learn distinct aspects of the original majority class. This is done by creating different balanced training sets by undersampling, learning a model for each dataset and then combining all predictions as in bagging. In the case of EasyEnsemble, boosting is used as classifier so that the method is able to integrate the advantages of boosting and bagging. In the same spirit, several studies have integrated oversampling/undersampling in ensembles of SVMs [118–121]. SMOTEBoost [122] combines boosting with SMOTE. Similarly, DataBoost-IM [45] generates synthetic samples as well within the boosting framework to improve the predictive accuracies of both the majority and minority classes. RareBoost [123] modifies the boosting algorithm to increase accuracy on the rare class by emphasizing the difference

of TN from FN, and TP from FP at each iteration of boosting. JOUS-Boost [124] generates duplicates of the minority class with oversampling, but also introduce perturbations ("jittering") by adding IID noise to minority examples.

Cost-sensitive learning

In unbalanced classification tasks, it is usually more important to correctly predict positive (minority) instances than negative (majority) instances. This is often achieved by associating different costs to erroneous predictions of each class. Cost-based methods operating at the algorithm level are able to consider misclassification costs in the learning phase without the need of sampling the two classes. Example of these classifiers are cost-sensitive boosting [125, 126], SVM [127] and Neural Network [128].

In the family of decision tree classifiers, cost-based splitting criteria are used to minimize costs [129]; or cost information determine whether a subtree should be pruned [130]. In general, pruning allows improving the generalization of a tree classifier since it removes leaves with few samples on which we expect poor probability estimates. Although, when the classification task is unbalanced, leaves containing few samples are often the one associated to the minority class and the first removed with pruning.

With Metacost [78] Domingos proposed a general framework that allows transforming any non cost-sensitive classifier into a cost-sensitive one. Similarly, thresholding [131] allows using cost-insensitive algorithms for cost minimization via different classification thresholds. These last methods are also called cost-sensitive meta-learners, since they are wrappers that minimize misclassification costs using standard classification algorithm. In many applications, however, costs are not explicitly available or easy to estimate, complicating the use of cost-sensitive algorithms [132].

3.2 Learning with non-stationarity

A standard assumption in machine learning is that the training and testing sets are drawn from the same underlying distribution. Let $\mathcal{P}_{tr}(x, y)$ and $\mathcal{P}_{ts}(x, y)$ denote joint probability of a sample (x, y) according to training and testing distribution. In a non-stationary environment we have $\mathcal{P}_{tr}(x, y) \neq \mathcal{P}_{ts}(x, y)$. When there is a distributional mismatch between the test and training data, the model fitted using the training data will be sub-optimal for the test scenario. Typically, this mismatch is generated by the following two causes: i) Sample Selection Bias (SSB) or ii) time evolving data.

In the first case the selection process of the training samples is responsible for a biased training set and depending on the bias different solutions exist to correct this bias. The second cause of non-stationarity is linked to streaming data where the data distributions evolve with time.

3.2.1 Sample Selection Bias

Sample Selection Bias (SSB) occurs when the training set available is biased because it has been selected in a way that is not representative of the whole population. For example, consider the problem where a bank wants to predict whether someone who is applying for a credit card will be able to repay the credit at the end of the month. The bank has data available on customers whose applications have been approved, but has no information on rejected customers. This means that the data available to the bank is a biased sample of the whole population. The bias in this case is intrinsic to the dataset collected by the bank.

We formalize the sample bias by means of a random variable $\mathbf{s} \in \{0, 1\}$ where samples included in the biased training set have $\mathbf{s} = 1$, and $\mathbf{s} = 0$ otherwise. Then the joint distribution of a sample (x, y) in the biased training set is $\mathcal{P}(x, y|\mathbf{s} = 1)$ and $\mathcal{P}(x, y)$ is the distribution in the case when the training set is unbiased. SSB requires what is known as the *support condition* [133]: the support of the probability measure of the training data has to be a subset of the support of the probability measure of the test data. Under this condition $\mathcal{P}_{tr}(y, x) = \mathcal{P}(y, x|\mathbf{s} = 1)$ and $\mathcal{P}_{ts}(y, x) = \mathcal{P}(y, x)$. The selection bias can be of different types: class prior bias, feature bias (also called covariate shift) and complete bias.

Class prior bias, also called prior probability shift, is essentially a change in class priors: $\mathcal{P}(y|\mathbf{s} = 1) \neq \mathcal{P}(y)$. It corresponds to the case when the selection is independent of the feature x given the class y: $\mathcal{P}(\mathbf{s} = 1|x, y) = \mathcal{P}(\mathbf{s} = 1|y)$. As a consequence of prior change, class-conditional probabilities are different ($\mathcal{P}(y|x, \mathbf{s} = 1) \neq \mathcal{P}(y|x)$), while within-class distribution remain unchanged ($\mathcal{P}(x|y, \mathbf{s} = 1) = \mathcal{P}(x|y)$). This type of bias can be introduced voluntarily for unbalanced classification tasks as in the case of undersampling and oversampling techniques presented in Section 3.1.1. As a result of class prior bias, classifiers have probability estimates that are poorly calibrated [28]. Well-know methods for correcting the prior bias are given by Saerens et al. [134] and Elkan [35].

Feature bias refers to changes in the distribution of the input variables without affecting the conditional probability $\mathcal{P}(y|x)$. The selection is independent of the class label y given the feature x (missing at random): $\mathcal{P}(\mathbf{s} = 1|x, y) = \mathcal{P}(\mathbf{s} = 1|x)$. This condition implies

$\mathcal{P}(y|x, \mathbf{s} = 1) = \mathcal{P}(y|x)$. Feature bias appears to be most studied type of bias [135–138] and a standard solution consist into re-weighting training instances [139].

Finally, complete bias is the most general case where the selection depended on both y and x (missing not at random). The most famous method for correcting complete bias is the one of Heckman [140] that gave him the Nobel Prize. Heckman defines two linear models, one for estimating $\mathbf{s}$ and one for y, which use different subset of features. The two-step procedure for bias correction consists into modeling $\mathbf{s}$ with ordinary least squares and then using the output as an additional feature in the model used to estimate y. If the same features are used to estimate both linear models estimating $\mathbf{s}$ and y, then the additional variable may end up highly correlated with the biased estimate of y and the Heckman procedure is not effective in correcting the bias [141]. Following the same idea Zadrozny and Elkan [142] use a classifier to predict $\mathcal{P}(\mathbf{s} = 1|x)$ and then incorporate this probability estimate as input feature in a new classifier used to predict $\mathcal{P}(y|x)$, but there is no theoretical guarantee that this method should correct SSB. For a recent survey on SSB we refer to [141].

The machine learning community has extensively investigated the SSB problem [35, 77, 142–145]. A standard remedy to SSB is *importance weighting* which consists of re-weighting the cost of training samples errors to more closely reflect that of the test distribution [77, 143, 144]. Using the Bayes formula we can write $\mathcal{P}(x, y)$ in terms of $\mathcal{P}(x, y|\mathbf{s} = 1)$:

$$\mathcal{P}(x, y) = \frac{\mathcal{P}(x, y|\mathbf{s} = 1)\mathcal{P}(\mathbf{s} = 1)}{\mathcal{P}(\mathbf{s} = 1|x, y)} = \frac{\mathcal{P}(\mathbf{s} = 1)}{\mathcal{P}(\mathbf{s} = 1|x, y)}\mathcal{P}(x, y|\mathbf{s} = 1) \tag{3.1}$$

Hence, we can correct SSB using a weight-sensitive algorithm, where a training instance (x, y) receive as weights $\frac{\mathcal{P}(\mathbf{s}=1)}{\mathcal{P}(\mathbf{s}=1|x,y)}$. The probability $\mathcal{P}(\mathbf{s} = 1)$ can easily be estimated knowing the proportion of sampled examples, however computing $\mathcal{P}(\mathbf{s} = 1|x, y)$ is not straightforward. In the case of feature bias we can rewrite (3.1) as:

$$\mathcal{P}(x, y) = \frac{\mathcal{P}(\mathbf{s} = 1)}{\mathcal{P}(\mathbf{s} = 1|x)}\mathcal{P}(x, y, \mathbf{s} = 1) \tag{3.2}$$

Now we can estimate term $\mathcal{P}(\mathbf{s} = 1|x)$ using a classifier that distinguish between sampled points and instances not included in the training set. However, Cortes et al. [133] argue that the re-weighting approach is able to remove the bias as long as the weights are estimated correctly. With a poor estimate of the weights we are not guaranteed to be able to remove the bias. In estimating $\mathcal{P}(\mathbf{s} = 1|x)$ we might have some values equal to zero, so it may be safer to assume complete bias [146]. SSB is closely related to the problem of learning under time evolving data streams, where importance weighting is used to correct the concept drift due to covariate shift [147].

3.2.2 Time evolving data

Most of the times, the main cause of differences between training and testing sets in data streams is due to a change in the data generating process. Using Bayes rule we can write the joint distribution of a sample (x, y) as:

$$\mathcal{P}(x, y) = \mathcal{P}(y|x)\mathcal{P}(x) = \mathcal{P}(x|y)\mathcal{P}(y) \tag{3.3}$$

Usually, in classification we are interested in a estimation of $\mathcal{P}(y|x)$, from (3.3) we have:

$$\mathcal{P}(y|x) = \frac{\mathcal{P}(x|y)\mathcal{P}(y)}{\mathcal{P}(x)} \tag{3.4}$$

Using (3.4), a change between the distribution of the data stream at time t and $t+1$, $\mathcal{P}_t(x, y) \neq \mathcal{P}_{t+1}(x, y)$, can come from [135]: i) $\mathcal{P}(y|x)$, ii)$\mathcal{P}(x|y)$, iii) $\mathcal{P}(y)$ and iv) combinations of the previous. Note that a change in $\mathcal{P}(x)$ does not affect y and can thus be ignored [148]. Most often, regardless of type of terms varying, a change in the distribution is referred in literature as *Concept Drift* [23] or *Dataset Shift* [149].

Change in the probability priors ($\mathcal{P}_t(y) \neq \mathcal{P}_{t+1}(y)$) can cause well-calibrated classifiers to become miscalibrated [28]. Concept drift due to $\mathcal{P}_t(x|y) \neq \mathcal{P}_{t+1}(x|y)$ affects the distribution of the observations within the class, but leaves the class boundary unchanged ($\mathcal{P}_t(y|x) = \mathcal{P}_{t+1}(y|x)$). This type of drift is often called *covariate shift* [141]. When $\mathcal{P}_t(y|x) \neq \mathcal{P}_{t+1}(y|x)$, there is a change in the class boundary that makes any previously learned classifiers biased. The latter is the worst type of drift, because it directly affects the performance of a classifier, as the distribution of the features, with respect to the class, has changed [148] (see Figure 3.2). In general it is hard to say where the change comes from, because we have access to only estimations of the previous probabilities and we have no knowledge (or control) of the data generating process.

Learning algorithms operating in non-stationary environments typically rely only on the supervised information that is up-to-date (thus relevant), and remove obsolete training samples [150]. However, concepts learned in the past may re-occur in the future, therefore when removing obsolete training samples, we could remove information that is still relevant. This is known as the *stability-plasticity* dilemma [151].

Concept Drift (CD) adaptation approaches can be grouped into two families: i) active and ii) passive adaptation [150]. The former [152–155], reacts to CD when a change is detected (e.g. by means of a Change Detection Test). Active approaches are mostly adopted for data distributions that change abruptly, because change detection with gradual drift is typically more difficult [156]. When a change is identified the classifier is updated or removed and replaced by a new one trained on the most recent samples available.

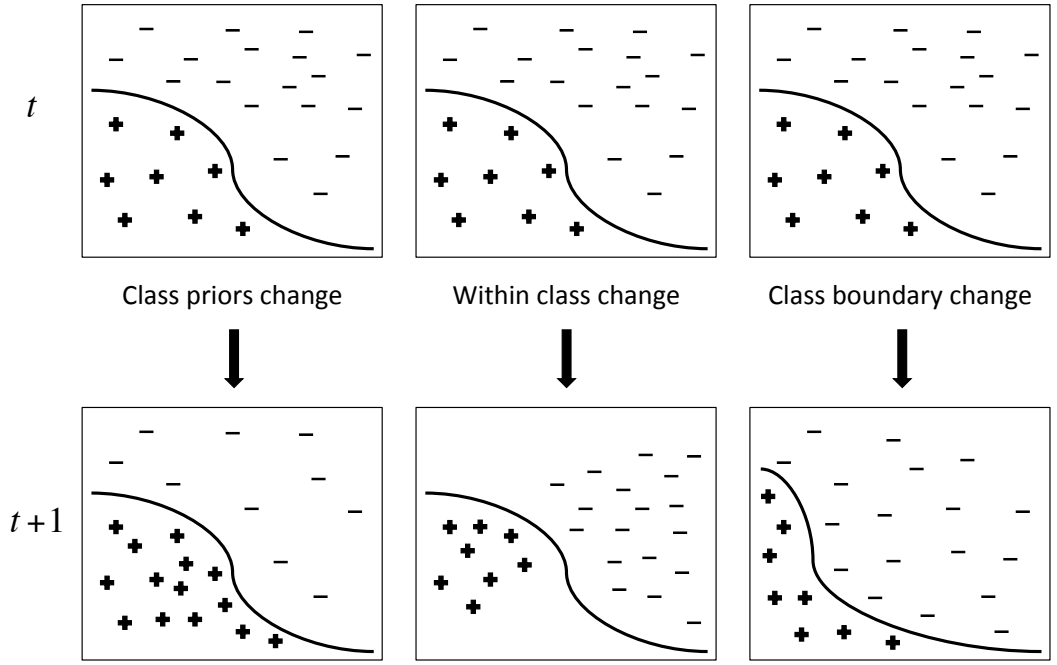


FIGURE 3.2: Illustrative example of different types of Concept Drift. Change in class priors is due to $\mathcal{P}_t(y) \neq \mathcal{P}_{t+1}(y)$, while within class change (covariate shift) occurs when $\mathcal{P}_t(x|y) \neq \mathcal{P}_{t+1}(x|y)$. Class boundary change happens when $\mathcal{P}_t(y|x) \neq \mathcal{P}_{t+1}(y|x)$.

The change detector can work by: a) checking features distributions or b) analyzing misclassification errors. In first case features are inspected to detect possible evolving distributions, whereas misclassification errors are used to identify changes in the class boundary. CD detection is particularly challenging in the presence of class unbalanced, because we might have to wait a while before having enough samples from the minority class [148].

In the case of passive approaches the classifier is continuously updated as soon as new supervised samples become available (no triggering mechanism involved) [18, 157, 158]. Generally, these approaches are used in the presence of gradual drifts and recurring concepts [158]. Passive approaches typically rely on ensemble of classifiers, where CD adaptation is obtained by weighting the ensemble members and creating/removing the classifiers composing the ensemble. By integrating several classifiers, the ensemble combines what is learned from new observations and the knowledge acquired before [159–161]. The weights of each member in the ensemble are computed to reflect how well a classifier is still relevant for the current concepts. If the weight of a classifier drop below a certain threshold, then the classifier is usually replaced, see for examples SEA [160] and DWM [162]. Other ensemble techniques [159–161] use ensemble of classifiers in order to combine what is learned from new observations and the knowledge acquired before.

Alternatively, passive CD adaptation is achieved by training a classifier over a sliding window of the recent supervised samples (e.g. STAGGER [163] and FLORA [164]).

It becomes critical then to set the rate of forgetting, defined by the window size, in order to match the rate of change in the distribution [165]. The simplest strategy uses a constant forgetting rate, which boils down to consider a fix window of recent observations to retrain the model. FLORA approach [164] uses a variable forgetting rate where the window is shrunk if a change is detected and expanded otherwise.

For a recent and in-deep review on CD adaptation we refer the reader to [156]. This thesis we will focus on passive approaches, where the data stream is received in batches of daily transactions. In the case of fraud detection, the data stream present not only non-stationary distributions, but also unbalanced classes. The problem of learning in the case of unbalanced data has been widely explored in the static learning settings [82, 91, 92, 116]. Learning from non-stationary data stream with skewed class distribution is however a relatively recent domain and will be treated in the next section.

3.3 Learning with evolving and unbalanced data streams

In many applications (e.g. network traffic monitoring and web access data) the data is received over time with high frequency and it is not possible to store all historical samples. The data has to be processed in real time and it may not be feasible to revisit previous transactions. This restriction is known in the literature as *one-pass constraint* [166].

One popular algorithm for mining data streams is Very Fast Decision Tree (VFDT) by Domingos and Hulten [167]. VFDT learns incrementally from each new observation without storing any examples, i.e. using constant memory and constant time per example, and uses Hoeffding bounds to determine the conversion of a tree leaf to a tree node. Since the seminal paper of Domingos, several similar algorithms have been proposed, e.g. CVFDT [168] uses a sliding window for concept drift adaptation, VFDTc [169] extends VFDT for continuous data and CD using a Naive Bayes classifiers in the leaves.

When the data streams are unbalanced, a certain time has to elapse in order to accumulate enough samples from the minority class. Since a classifier requires a large set of samples to learn an accurate model, typically the training is done batch wise, i.e. when a sufficient number of samples from both classes are available. Despite the large literature on data stream studies, few works have tried to address the problem of learning with non-stationary data streams with unbalanced class distribution. In these works the class unbalance problem is typically addressed my adopting one of the resampling methods presented in Section 3.1.1 to rebalance the batches of the data streams [19, 170–173]. For example, Ditzler and Polikar propose Learn++.NIE [172]

where they extend Learn++.NSE [174] for unbalanced data streams by training classifiers on multiple balanced batches obtained with undersampling. Following the same idea, Learn++.CDS [173] rebalances the training sets of each classifier in the ensemble using SMOTE [92].

Rebalancing a batch with few positive (minority) samples could mean removing many negative samples (in the case of undersampling) or significant replication of the minority class (oversampling) with a high risk of overfitting. A different way to compensate the class imbalance within the batches is to propagate minority class samples along the stream. For example Gao et al. [170, 171] combine minority class propagation and undersampling of the majority class. The positive examples are accumulated along the stream until they represent 40% of the observations. When this happens, the oldest positive examples are replaced by the new observations from the minority class. This propagation method ignores the similarity of the minority class instance to the current concept, relying only on its similarity in time.

Lichtenwalter and Chawla [175] suggest propagating not only minority samples, but also observations from the majority class that have been previously misclassified to increase the boundary definition between the two classes. Chen and He proposed REA [176] where they recommend propagating only examples from the minority class selected using a kNN algorithm. Similarly, SERA [177] propagates to the last batch only minority class that belong to the same concept using Mahalanobis distance. Hoens and Chawla [148] use instead an instance propagation mechanism based on a Naïve Bayes classifier. In this case, Naïve Bayes is used to select old positive instances which are relevant to the current minority class context. This method relies on finding instances that are similar to the current minority class context.

Wang, Minku and Yao [178] propose Sampling-based Online Bagging (SOB) to deal with unbalanced data streams. Their algorithm, is essentially a modification of Online Bagging [179], in which the sampling rate of the instances belonging to one class is determined adaptively based on the current imbalance status and classification performance. The problem with this approach is that it is not designed to handle CD as it aims to maximize G-mean greedily over all received examples [178].

3.4 Algorithmic solutions for Fraud Detection

In the literature, both supervised [64, 180, 181] and unsupervised [11, 182] ML algorithms have been proposed for credit card fraud detection. As explained in Section 2.1,

supervised techniques assume the availability of annotated datasets, i.e. transaction labeled as genuine or fraudulent. In this case a model is trained under the supervision of the class information to discover patterns associated to the fraudulent and genuine class. On the contrary, unsupervised methods work with datasets that contain unlabeled samples. These methods consist in outlier detection or anomaly detection techniques that associate fraudulent behaviours to any transaction that does not conform to the majority, without knowledge on transactions class [11]. Unsupervised techniques usually generate too many false alerts so it is often a good idea to combine both supervised and unsupervised methods as in [62].

The detection problem can be seen from the transaction and card level. At the card level is possible to group transactions from the same card and learn behavioral models of individual cards. Behavioral models are typically unsupervised methods that aim to characterize the genuine behavior of each individual card over the time. These models only consider the previous history of each card but do not attempt to identify global patterns of fraudulent behaviors; they only try to detect changes in behavior. Examples of these are given by Fawcett and Provost [183], Bolton [11] and Weston [184]. The problem with this approach is that a change in behavior may not be due to fraud. For example, the change in spending behavior during holiday seasons may be not necessary linked to fraudulent activities.

Models that operate at the transaction level try to differentiate legitimate transactions from fraudulent ones without knowing the behavior of a card. These customer-independent models can serve as a second line of defense, the first being customer-dependent models (behavioral models). This strategy considers only transactions in isolation from each other. Neither the previous history of the associated account, nor other transactions are taken into consideration. Alternatively, it is possible to work at the transaction level and include customer behavior in the model by using aggregate variable of the cardholder as presented in Section 2.2.2.

3.4.1 Supervised Approaches

In supervised algorithms fraudulent and non-fraudulent examples are used to predict the class of a new observation. Supervised techniques developed for fraud detection can be group into: i) supervised profiling, ii) classification, iii) cost-sensitive and iv) networks methods.

Supervised Profiling

When labeled transactions are available, it is possible to *profile* the distribution of relevant variables for genuine and fraudulent card [185]. This means that it is possible to create different profiles for each class. At this point every new transaction can be compared to see which profile is more similar. For example, Siddiqi [186] proposes to use Weight Of Evidence (WOE) as similarity measure between two profiles in credit risk. The same metric can be used to compare the genuine and fraudulent profiles [185]. Profiles can also be created using rule-based methods.

Rules can come from human expert or from statistical models and they have the advantage of being easy to understand and to implement. A set of rules is defined for each profile and if a new transaction matches these rules it is assumed to have the same profile. Chan [181] for instance uses rules to filter out safe transactions. An adaptive user profiling method was proposed by Fawcett and Provost [187] but for telecommunication frauds. Cortes [188] defines an *account signature* for profiling accounts in data streams.

As the criminal activities and legitimate user-behavior evolves, fraudulent profiles have to be updated as well. This means that statistical-based rules have to be updated periodically. Alternatively a weighted ensemble approach can be used to include new rules while maintaining old rules [189]. Profiles must be updated to reflect the dynamic patterns of criminal activity as well as changes in legitimate user behavior. This presents a challenge for static rule-based methods that are learned off-line, as they must be frequently validated and retrained.

Classification

Classification appears to be the standard way to approach Fraud Detection [190] and several classification algorithms have been used, e.g. Neural Networks [180, 191–193], Logistic Regression [65], Association Rules [194], Support Vector Machines [63], Fisher Discriminant Analysis [69], and Decision Trees [19, 68, 70].

Decision trees have found many applications in fraud detection, for both credit card fraud detection [126] and credit risk scoring [195]. Neural Networks as well have been extensively applied in detection system such as Falcon [196], Minerva (Dorronsoro [193]), FDS [191], Cardwatch [192] and Visa [197]. Dorronsoro [193] shows that a three-layer nets is capable of dealing with the highly skewed class distributions. However, as explained in Section 3.1, supervised methods in general suffer from the problem of unbalanced class sizes: the legitimate transactions generally far outnumber the fraudulent ones.

Neural network are able to learn difficult class boundaries, but they are *black-boxes* as it is not possible for a human being to understand how they behave. Rules Extraction and Decision Tree are widely used in fraud detection because unlike black-box techniques allow a better grasp of the classification. This is important because, often, the analysts are not Machine Learning specialists and need to understand classification mechanisms in order to trust and use them. These approaches are often based on the extraction of conjunction and disjunction of rules that are representational of the choice of the classification and directly understandable.

Probabilistic graphical model were used as well in fraud detection such as Bayesian Belief Networks [185]. Card's activities can be modeled using a Hidden Markov Models. Sudjianto [185] shows that is possible to use HMM to monitor transactions of a card defining different status of a card. Finally, between all the algorithms proposed in the literature, we recommend using Random Forest, because several studies have shown that it achieves the best results among different classifiers [18, 20, 61, 63, 64, 71].

Cost-sensitive methods

Since businesses are interested in reducing the monetary loss due to fraudulent activities, there is a large body of work on cost-sensitive classifiers. These methods are also called *cost-sensitive algorithms*, because they are able to take into account the different costs in misclassify a transaction as fraud or legitimate. As shown in Section 3.1.2, cost-sensitive learning [78], [35] is an alternative way to deal with the unbalanced problem that consist into assigning larger costs to errors made on the minority class. Many cost-sensitive algorithms available in the literature are based on boosting; see for example AsymBoost [198], AdaCost [126], CSB [199], DataBoost [45], AdaUBoost [200] and SMOTEBoost [122].

Traditional cost-sensitive learning such as AdaCost [126] assumes that the costs are fixed and class-dependent, however in fraud detection the cost is proportional to the transaction amount. The larger the amount, the greater the potential loss in case of fraud. In these settings the cost of missing a fraud (a false negative) is not fixed, but proportional to the transaction amount. Examples of cost-sensitive classifiers using transaction-dependent costs are [68–70]. Mahmoudi and Duman [69] use Modified Fisher Discriminant Analysis to consider example-dependent cost for each transaction to maximize total profit. Similarly Bahnsen et al. [70] and Sahin et al.[68] use example-dependent cost-sensitive decision trees for maximizing the savings. Cost-based methods are useful when the primary goal is to minimize some costs / maximize the benefits of the detection or savings since the loss function takes into account the financial loss occurred in each prediction.

Given the higher costs of misclassifying a fraud than a genuine transaction, cost-based algorithms could in principle prefer to produce a false alert rather than take the risk to predict a transaction as legitimate when it is not. As a consequence, these algorithms can generate many false positives and be of no practical use for investigators who require precise alerts (see Section 2.2.1).

Detection of Fraud Networks

The detection of links between data can also lead to fraud discovering. For example, in the telecommunication domain, experts have noticed that a fraudulent account is often connected to another by some calls given between fraudsters. By analyzing these links, they discover fraud networks. Such approach seems complementary to individual fraud detection [201].

When the fraudulent activity is spread over many transactions and card, illegal activities can be uncover by analyzing pattern of related transactions. In particular link analysis and graph mining methods may be able to detect these groups of fraudulent transaction [185]. Recently Van Vlasselaer et al. [61] have proposed APATE, a framework for credit card fraud detection that allows including network information as additional features to the original feature vector describing a transaction. They show that features including network information are able to improve significantly the performance of a standard supervised algorithm.

3.4.2 Unsupervised Approaches

Unsupervised methods are used when there are no prior sets of legitimate and fraudulent observations. Since they are not based on examples of fraud or genuine transactions, unsupervised strategies have the advantage of being independent of their selection, and are able, in theory, to discover frauds still unobserved, that have not been detected by an expert. Yet they are not affected by the problem of mislabeled dataset and class imbalance. However unsupervised credit card fraud detection has not received a lot of attention in the literature [11].

Techniques employed in fraud detection are usually a combination of profiling and outliers detection methods. They model a baseline distribution that represents normal behavior and then attempt to detect observations that show the greatest departure from this norm [202]. One of these methods is Peer Group Analysis [184] which clusters customers into different profiles and identifies frauds as transactions departing from the customer profile

(see also the recent survey by Phua [203]). Others model cardholder behavior by means of self-organizing maps [5, 204, 205].

In order to detect outliers or anomalies it is important first to define when an example is an outlier. Grubbs [206] gives the following definition: "An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs". For Barnett & Lewis [207] an outlier is: "An observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data".

When it is possible to define a region or distribution of the data representing the normal behavior then all observations falling outside can be flagged as outlier. This approach however comes with several challenges: i) defining a normal region which encompasses every possible normal behavior is very difficult, ii) normal behavior keeps evolving and an existing notion of normal behavior might not be sufficiently representative in the future, iii) the boundary between normal and outlying behavior is often fuzzy, iv) frauds adapt themselves to make the outlying observations appear like normal, thereby making the task of defining normal behavior more difficult and v) data contains noise which is similar to the actual outliers.

In general we can distinguish between two forms of outliers: local and global outliers. A global outlier is an observation anomalous to the entire data set. In the case of fraud detection, an example of global outlier can be a transaction of €10000 when all the other transactions in the dataset have smaller amount. A local outlier is an observation that looks anomalous when compared to subgroups of the data. When a card is mostly used in a restricted geographic area, a purchase made by the same card in foreign country is a local outlier within its transactions, but is not anomalous with respect to all possible transactions. Depending on the type of outlier that we want to detect we can use local and global approaches.

Local Approaches

In [209], the authors aim to detect outliers by analyzing local information of the space. To achieve this, they introduce the notion of *Rough Membership Function* that computes the degree of deviance of data, depending on the density of its local neighborhood.

In [11], Bolton et al. have based their work on the hypothesis that a fraud differs according to the considered data, similarly to what is proposed in [187, 210]. The idea of Peer Group Analysis [11] is that time series that are in some sense similar are grouped together to form a peer group. Transactions that deviate strongly from their peer group are flagged as potentially fraudulent. The behaviour of the peer group is summarized

at each subsequent time point and the behaviour of a time series compared with the summary of its peer group. The authors present their analysis more as a way to alert investigators of anomalous data than as a fraud classifier. In the paper, the authors consider time series of data that are clustered on the basis a distance function in groups of defined size. They next propose to resume each group by a model and then compute their deviance. They finally present results in a graph style that allows visualizing breakpoints. This approach is interesting because it is understandable and it is not based on an annotated set. It is then, a-priori, capable to detect any kind of fraud.

Global Approaches

The work presented in [211] addresses high dimensional spaces. In this kind of spaces, any observation can be considered as an outlier because of the sparsity of the data. The authors propose to project data in fewer dimensions spaces to avoid these phenomena and study the density of distribution to detect frauds. They split the original space into k subspaces of same depth on each dimension and then try to combine these spaces to form hypercubes of dimension n (n little enough) that are of a particularly weak density. To combine dimensions the best, they use an evolutionary approach that combines randomly dimensions to extract the best candidates. Experimental results show a quality close to systematic approach and a good quality of detection. Another benefit of this approach is, by projecting data, it is capable to deal with data having missing attributes. Its drawback is again its rendering, even if the selected dimensions can partially explain the analysis.

In [212], the question is to detect outliers in data having categorical attributes. The authors formalize the problem this way: "finding a small subset of a target dataset such that the degree of disorder of the resultant dataset after the removal of this subset is minimized". This degree of "disorder" is measured with the entropy of the dataset. Their technique aims to detect the k most deviant outliers in the dataset. They estimate that this parameter is not hard to define because, according to them, the problem is given this way in concrete cases (find for example the 5% of fraudsters). They develop an iterative algorithm that tries to optimize the set by minimizing the entropy. This approach, which could appear expensive, is in fact estimated experimentally as almost linear w.r.t. the size of the dataset and the number of outliers to extract (k). Results are interesting though the authors make the strong assumption that attributes are independent.

Part II

Contribution

Chapter 4

Techniques for unbalanced classification tasks

Results presented in this chapter have been published in the following papers:

- Andrea Dal Pozzolo, Olivier Caelen, and Gianluca Bontempi. *When is undersampling effective in unbalanced classification tasks?*. In European Conference on Machine Learning. ECML-KDD, 2015.
- Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. *Calibrating Probability with Undersampling for Unbalanced Classification.* In Symposium on Computational Intelligence and Data Mining (CIDM). IEEE, 2015.
- Andrea Dal Pozzolo, Olivier Caelen, Serge Waterschoot, Gianluca Bontempi, *Racing for unbalanced methods selection.* Proceedings of the 14th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL), IEEE, 2013

The chapter is divided into three main parts. The first part of the chapter (Section 4.1) is based on [213] and it analyses a well-known technique called *undersampling* (see Section 3.1.1), which consists into rebalancing (typically by resampling) of the classes before proceeding with the learning of the classifier. Though this seems to work for the majority of cases, no detailed analysis exists about the impact of undersampling on the accuracy of the final classifier. In particular we will propose a theoretical analysis specifying under which conditions undersampling is recommended and expected to be effective.

In the second part of the chapter (Section 4.2), which is based on [28], we show how sampling data used to train a model induces an artificial bias into the computed posterior probabilities, for which we show a corrective method. Although this bias does not

affect the ranking order returned by the posterior probability, it significantly impacts the classification accuracy and probability calibration. We use Bayes Minimum Risk theory to find the correct classification threshold and show how to adjust it after undersampling.

Finally, the third part of the chapter (Section 4.3) is based on [27]. In this section, we propose to use a racing algorithm to select adaptively the most appropriate strategy for a given unbalanced task. Racing allows one to test rapidly a large set of alternatives and we used it to compare undersampling against a large set of techniques for unbalanced classification. However, as confirmed by our experimental comparison, no technique appears to work consistently better in all conditions. The results show that racing is able to adapt the choice of the strategy to the specific nature of the unbalanced problem and to select rapidly the most appropriate strategy without compromising the accuracy.

4.1 When is undersampling effective in unbalanced classification tasks?

When the data is unbalanced, standard machine learning algorithms that maximize overall accuracy tend to classify all observations as majority class instances. This translates into poor accuracy on the minority class (low recall), which is typically the class of interest. Degradation of classification performance is not only related to a small number of examples in the minority class in comparison to the number of examples in the majority classes (expressed by the class imbalance ratio), but also to the minority class decomposition into small sub-clusters [214] (also known in the literature as *small disjuncts* [80]) and to the overlap between the two classes [84, 215–217]. In these studies it emerges that performance degradation is strongly caused by the presence of both unbalanced class distributions and a high degree of class overlap. Additionally, in unbalanced classification tasks, the performance of a classifier is also affected by the presence of noisy examples [218, 219].

One possible way to deal with this issue is to adjust the algorithms themselves [77, 78, 82]. Here we will consider instead a data-level strategy known as *undersampling* [21]. In an unbalanced problem, it is often realistic to assume that many observations of the majority class are redundant and that by removing some of them at random the data distribution will not change significantly. Nevertheless, the risk of removing relevant observations from the dataset is still present, since the removal is performed in an unsupervised manner. In practice, sampling methods are often used to balance datasets with skewed class distributions because several classifiers have empirically shown better performance when trained on balanced dataset [88, 90]. However, these studies do not imply that

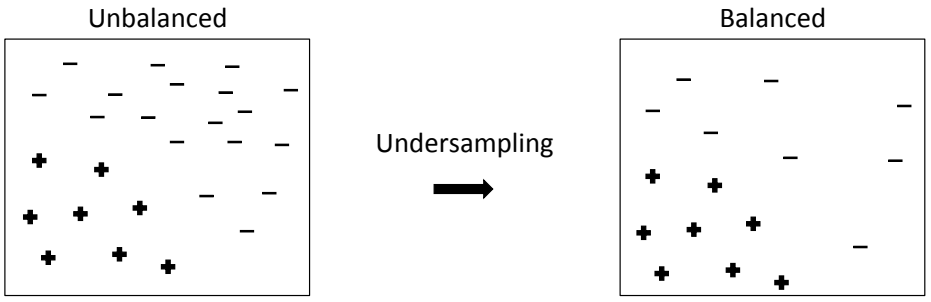


FIGURE 4.1: Undersampling: remove majority class observations until we have the same number of instances in the two classes.

classifiers cannot learn from unbalanced datasets. For instance, other studies have also shown that some classifiers do not improve their performances when the training dataset is balanced using sampling techniques [82, 106]. As a result, for the moment the only way to know if sampling helps the learning process is to run some simulations. Despite the popularity of undersampling, we have to remark that there is not yet a theoretical framework explaining how it can affect the accuracy of the learning process.

In this chapter we aim to analyze the role of the two side effects of undersampling on the final accuracy. The first side effect is that, by removing majority class instances, we perturb the a priori probability of the training set and we induce a warping in the posterior distribution [35, 134]. The second is that the number of samples available for training is reduced with an evident consequence in terms of accuracy of the resulting classifier. We study the interaction between these two effects of undersampling and we analyze their impact on the final ranking of posterior probabilities. In particular we show under which conditions an undersampling strategy is recommended and expected to be effective in terms of final classification accuracy.

4.1.1 The warping effect of undersampling on the posterior probability

Let us consider a binary classification task $f : \mathbb{R}^n \to \{0, 1\}$, where $\mathrm{X} \in \mathbb{R}^n$ is the input and $\mathrm{Y} \in \{0, 1\}$ is the output. In the following we will also use the label negative (resp. positive) to denote the label 0 (resp. 1). Suppose that the training set $\mathrm{T}_N = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ of size N is unbalanced (i.e. the number N^+ of positive cases is small compared to the number N^- of negative ones) and that rebalancing is performed by undersampling, i.e. the resulting dataset contains a subset of the negatives. Let us introduce a random binary selection variable $\mathbf{s}$ associated to each sample in T_N, which takes the value 1 if the point is sampled and 0 otherwise. We now derive how the posterior probability of a model learned on a balanced subset relates to the one learned on the original unbalanced dataset, on the basis of [149]. Let us assume that the selection

variable $\mathbf{s}$ is independent of the input x given the class y (*class-dependent selection*):

$$\mathcal{P}(\mathbf{s}|y,x) = \mathcal{P}(\mathbf{s}|y) \tag{4.1}$$

where $\mathcal{P}(\mathbf{s}=1|y,x)$ is the probability that a sample (x,y) is included in the balanced training sample. This assumption implies $\mathcal{P}(x|y,\mathbf{s}) = \mathcal{P}(x|y)$, i.e. by removing observation at random in the majority class we do not change within-class distributions. With undersampling there is a change in the prior probabilities ($\mathcal{P}(y|\mathbf{s}=1) \neq \mathcal{P}(y)$) and as a consequence the class-conditional probabilities are different as well, $\mathcal{P}(y|x,\mathbf{s}=1) \neq \mathcal{P}(y|x)$. Let the sign + denote $\mathbf{y}=1$ and $-$ denote $\mathbf{y}=0$, e.g. $\mathcal{P}(+|x) = \mathcal{P}(\mathbf{y}=1|x)$ and $\mathcal{P}(-|x) = \mathcal{P}(\mathbf{y}=0|x)$. From Bayes' rule we can write:

$$\mathcal{P}(+|x,\mathbf{s}=1) = \frac{\mathcal{P}(\mathbf{s}=1|+,x)\mathcal{P}(+|x)}{\mathcal{P}(\mathbf{s}=1|+,x)\mathcal{P}(+|x) + \mathcal{P}(\mathbf{s}=1|-,x)\mathcal{P}(-|x)} \tag{4.2}$$

Using condition (4.1) in (4.2) we obtain:

$$\mathcal{P}(+|x,\mathbf{s}=1) = \frac{\mathcal{P}(\mathbf{s}=1|+)\mathcal{P}(+|x)}{\mathcal{P}(\mathbf{s}=1|+)\mathcal{P}(+|x) + \mathcal{P}(\mathbf{s}=1|-)\mathcal{P}(-|x)} \tag{4.3}$$

With undersampling we keep all positives and a subset of negatives (see Figure 4.1), therefore we have:

$$\mathcal{P}(\mathbf{s}=1|+) = 1 \tag{4.4}$$

and

$$\frac{\mathcal{P}(+)}{\mathcal{P}(-)} \leq \mathcal{P}(\mathbf{s}=1|-) < 1 \tag{4.5}$$

Note that if we set $\mathcal{P}(\mathbf{s}=1|-) = \frac{\mathcal{P}(+)}{\mathcal{P}(-)}$, we obtain a balanced dataset where the number of positive and negative instances is the same. At the same time, if we set $\mathcal{P}(\mathbf{s}=1|-)=1$, no negative instances are removed and no undersampling takes place. Using (4.4), we can rewrite (4.3) as

$$\mathcal{P}(+|x,\mathbf{s}=1) = \frac{\mathcal{P}(+|x)}{\mathcal{P}(+|x) + \mathcal{P}(\mathbf{s}=1|-)\mathcal{P}(-|x)} \tag{4.6}$$

Let us denote $\beta = \mathcal{P}(\mathbf{s}=1|-)$ as the probability of selecting a negative instance with undersampling, $p = \mathcal{P}(+|x)$ as the true posterior probability of class + on the original dataset, and $p_s = \mathcal{P}(+|x,\mathbf{s}=1)$ as the true posterior probability of class + after sampling. We can rewrite equation (4.6) as:

$$p_s = \frac{p}{p + \beta(1-p)} \tag{4.7}$$

Equation (4.7) quantifies the amount of warping of the posterior probability due to undersampling.[1] From it, we can derive p as a function of p_s:

$$p = \frac{\beta p_s}{\beta p_s - p_s + 1} \tag{4.8}$$

The relation between p and p_s (parametric in β) is illustrated in Figure 4.2. The top

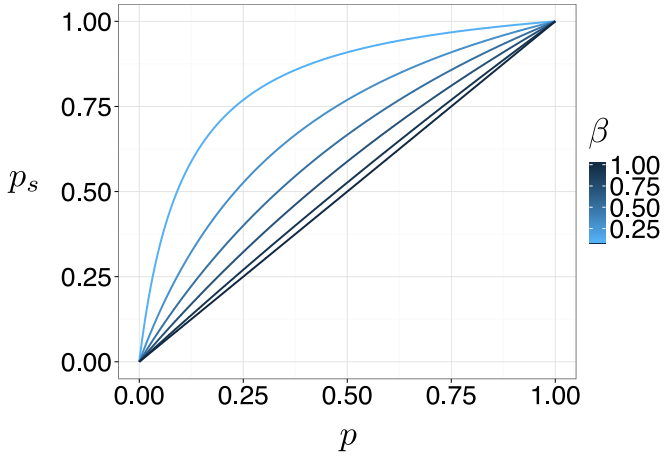


FIGURE 4.2: p and p_s at different β. When β is low, undersampling is strong, which means it is removing a lot of negatives, while for high values the removal is less strong. Low values of β leads to a more balanced problem.

curve of Figure 4.2 refers to the complete balancing which corresponds to $\beta = \frac{\mathcal{P}(+)}{\mathcal{P}(-)} \approx \frac{N^+}{N^-}$, assuming that $\frac{N^+}{N^-}$ provides an accurate estimation of the ratio of the prior probabilities.

Figure 4.4 illustrates the warping effect for two univariate ($n = 1$) classification tasks (see Figure 4.3). In both tasks the two classes are normally distributed ($\mathrm{X}^- \sim \mathcal{N}(0, \sigma)$ and $\mathrm{X}^+ \sim \mathcal{N}(\mu, \sigma)$), $\sigma = 3$ and $\mathcal{P}(+) = 0.1$ but the degree of separability is different (on the left large overlap for $\mu = 3$ and on the right small overlap for $\mu = 15$). It is easy to remark that the warping effect is larger in the low separable case.

As a final remark, consider that when $\beta = \frac{N^+}{N^-}$, the warping due to undersampling maps two close and low values of p into two values p_s with a larger distance. The opposite occurs for high values of p. In Section 4.1.3 we will show how this has an impact on the ranking returned by estimations of p and p_s.

[1] In the case of oversampling it can be showed that $p_s = \frac{\alpha p}{\alpha p + 1 - p}$, where α denotes the number of times a positive instance is replicated. The larger the class imbalanced, the large α has to be in order to obtain a balanced distribution.

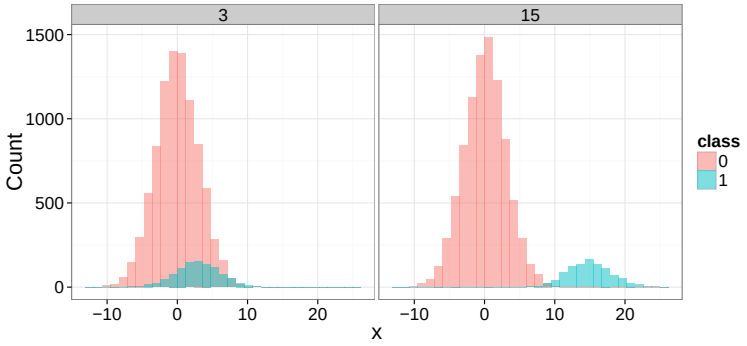


FIGURE 4.3: Synthetic datasets with positive and negative observations sampled from two different normal distributions. Positives account for 10% of the 10,000 random values. On the left we have a difficult problem with overlapping classes ($\mu = 3$), on the right an easy problem where the classes are well-separated ($\mu = 15$).

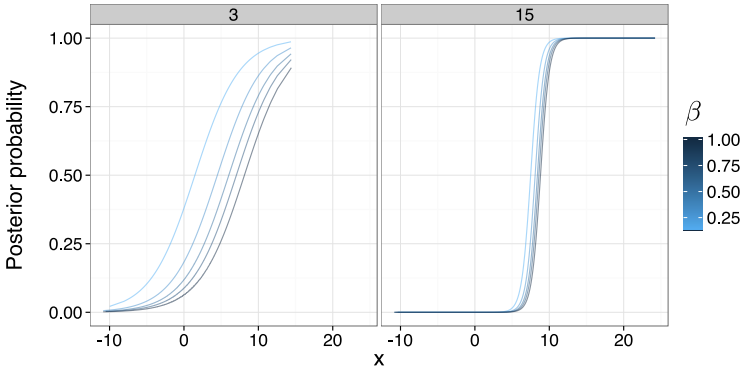


FIGURE 4.4: Posterior probability p_s as a function of β for two univariate binary classification tasks with norm class conditional densities $\mathrm{X}^- \sim \mathcal{N}(0, \sigma)$ and $\mathrm{X}^+ \sim \mathcal{N}(\mu, \sigma)$ (on the left $\mu = 3$ and on the right $\mu = 15$, in both examples $\sigma = 3$). Note that the original probability p corresponds to p_s when $\beta = 1$.

4.1.2 Warping and class separability

In this section we are going to show how the impact of warping depends on the separability nature of the classification task. Let ω^+ and ω^- denote the class conditional probabilities $\mathcal{P}(x|+)$ and $\mathcal{P}(x|-)$, and π^+ (π_s^+) the class priors before (after) undersampling. It is possible to derive the relation between the warping and the difference $\delta = \omega^+ - \omega^-$ between the class conditional distributions. From Bayes' theorem we have:

$$p = \frac{\omega^+\pi^+}{\omega^+\pi^+ + \omega^-\pi^-} \tag{4.9}$$

Suppose $\delta = \omega^+ - \omega^-$, we can write (4.9) as:

$$p = \frac{\omega^+\pi^+}{\omega^+\pi^+ + (\omega^+ - \delta)\pi^-} = \frac{\omega^+\pi^+}{\omega^+(\pi^+ + \pi^-) - \delta\pi^-} = \frac{\omega^+\pi^+}{\omega^+ - \delta\pi^-} \tag{4.10}$$

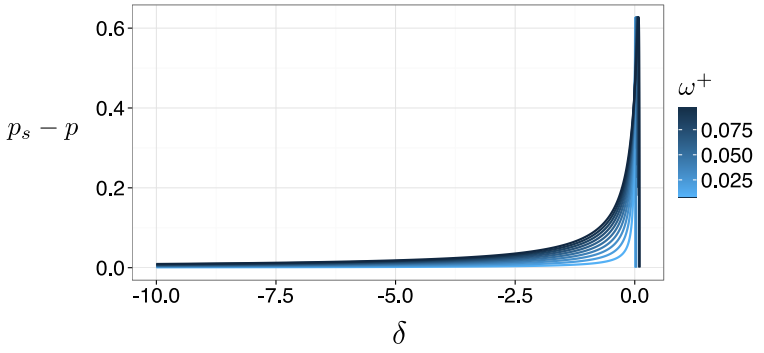


FIGURE 4.5: $p_s - p$ as a function of δ, where $\delta = \omega^+ - \omega^-$ for values of $\omega^+ \in \{0.01, 0.1\}$ when $\pi_s^+ = 0.5$ and $\pi^+ = 0.1$. Note that δ is upper bounded to guarantee $0 \leq p_s \leq 1$ and $0 \leq p \leq 1$.

since $\pi^+ + \pi^- = 1$. Similarly, since ω^+ does not change with undersampling:

$$p_s = \frac{\omega^+ \pi_s^+}{\omega^+ - \delta \pi_s^-} \tag{4.11}$$

Now we can write $p_s - p$ as:

$$p_s - p = \frac{\omega^+ \pi_s^+}{\omega^+ - \delta \pi_s^-} - \frac{\omega^+ \pi^+}{\omega^+ - \delta \pi^-} \tag{4.12}$$

Since $p_s \geq p$ because of (4.7), $0 \leq p_s \leq 1$ and $0 \leq p \leq 1$ we have: $0 \leq p_s - p \leq 1$. In Figure 4.5 we plot $p_s - p$ as a function of δ when $\pi_s^+ = 0.5$ and $\pi^+ = 0.1$. For small values of the class conditional densities it appears that the difference have the highest values for δ values close to zero. This means that the warping is higher for similar class conditional probabilities (i.e. low separable configurations).

4.1.3 The interaction between warping and variance of the estimator

Section 4.1.1 discussed the first consequence of undersampling, i.e. the transformation of the original conditional distribution p into a warped conditional distribution p_s according to equation (4.7). The second consequence of undersampling is the reduction of the training set size, which inevitably leads to an increase of the variance of the classifier. This section discusses how these two effects interact and their impact on the final accuracy of the classifier, by focusing in particular on the accuracy of the ranking of the minority class (typically the class of interest).

Undersampling transforms the original classification task (i.e. estimating the conditional distribution p) into a new classification task (i.e. estimating the conditional distribution p_s). In what follows we aim to assess whether and when undersampling has a beneficial effect by changing the target of the estimation problem.

Let us denote by $\hat{p}$ (resp. $\hat{p}_s$) the estimation of the conditional probability p (resp. p_s). Assume we have two distinct test points having probabilities $p_1 < p_2$ where $\Delta p = p_2 - p_1$ with $\Delta p > 0$. A correct classification aiming to rank the most probable positive samples should rank p_2 before p_1, since the second test sample has an higher probability of belonging to the positive class. Unfortunately the values p_1 and p_2 are not known and the ranking should rely on the estimated values $\hat{p}_1$ and $\hat{p}_2$. For the sake of simplicity we will assume here that the estimator of the conditional probability has the same bias and variance in the two test points. This implies $\hat{p}_1 = p_1 + \epsilon_1$ and $\hat{p}_2 = p_2 + \epsilon_2$, where ϵ_1 and ϵ_2 are two realizations of the random variable $\varepsilon \sim \mathcal{N}(b, \nu)$ where b and ν are the bias and the variance of the estimator of p. Note that the estimation errors ϵ_1 and ϵ_2 may induce a wrong ranking if $\hat{p}_1 > \hat{p}_2$.

What happens if instead of estimating p we decide to estimate p_s, as in undersampling? Note that because of the monotone transformation (4.7), $p_1 < p_2 \Rightarrow p_{s,1} < p_{s,2}$. Is the ranking based on the estimations of $p_{s,1}$ and $p_{s,2}$ more accurate than the one based on the estimations of p_1 and p_2?

In order to answer this question let us suppose that also the estimator of p_s is biased but that its variance is larger given the smaller number of samples.[2] Then $\hat{p}_{s,1} = p_{s,1} + \eta_1$ and $\hat{p}_{s,2} = p_{s,2} + \eta_2$, where $\eta \sim \mathcal{N}(b_s, \nu_s)$, $\nu_s > \nu$ and $\Delta p_s = p_{s,2} - p_{s,1}$. Let us now compute the derivative of p_s w.r.t. p. From (4.7) we have:

$$\frac{dp_s}{dp} = \frac{\beta}{(p + \beta(1-p))^2} \tag{4.13}$$

corresponding to a concave function. In particular for $p = 0$ we have $dp_s = \frac{1}{\beta} dp$, while for $p = 1$ it holds $dp_s = \beta dp$. We will now show that $\frac{dp_s}{dp}$ is bounded in the range $[\beta, \frac{1}{\beta}]$. Let λ be the value of p for which $\frac{dp_s}{dp} = 1$:

$$\lambda = \frac{\sqrt{\beta} - \beta}{1 - \beta}$$

from (4.13) we have

$$1 < \frac{dp_s}{dp} < \frac{1}{\beta}, \qquad \text{when } 0 < p < \lambda$$

and

$$\beta < \frac{dp_s}{dp} < 1 \qquad \text{when } \lambda < p < 1.$$

It follows that:

$$\beta \leq \frac{dp_s}{dp} \leq \frac{1}{\beta} \tag{4.14}$$

[2]It is well-known that training a classifier on a reduced dataset leads to probability estimates with larger variance [220].

Let us now suppose that the quantity Δp is small enough to have an accurate approximation $\frac{\Delta p_s}{\Delta p} \approx \frac{dp_s}{dp}$. We can define the probability of obtaining a wrong ranking of $\hat{p}_1$ and $\hat{p}_2$ as:

$$\begin{aligned}\mathcal{P}(\hat{p}_2 < \hat{p}_1) &= \mathcal{P}(p_2 + \epsilon_2 < p_1 + \epsilon_1) \\ &= \mathcal{P}(\epsilon_2 - \epsilon_1 < p_1 - p_2) = \mathcal{P}(\epsilon_1 - \epsilon_2 > \Delta p)\end{aligned}$$

where $\epsilon_2 - \epsilon_1 \sim \mathcal{N}(0, 2\nu)$.[3] By making a hypothesis of normality we have

$$\mathcal{P}(\epsilon_1 - \epsilon_2 > \Delta p) = 1 - \Phi\left(\frac{\Delta p}{\sqrt{2\nu}}\right) \tag{4.15}$$

where Φ is the cumulative distribution function of the standard normal distribution. Similarly, the probability of a ranking error with undersampling is:

$$\mathcal{P}(\hat{p}_{s,2} < \hat{p}_{s,1}) = \mathcal{P}(\eta_1 - \eta_2 > \Delta p_s)$$

and

$$\mathcal{P}(\eta_1 - \eta_2 > \Delta p_s) = 1 - \Phi\left(\frac{\Delta p_s}{\sqrt{2\nu_s}}\right) \tag{4.16}$$

We can now say that a classifier learned after undersampling has better ranking w.r.t. a classifier learned with unbalanced distribution when

$$\mathcal{P}(\epsilon_1 - \epsilon_2 > \Delta p) > \mathcal{P}(\eta_1 - \eta_2 > \Delta p_s) \tag{4.17}$$

or equivalently from (4.15) and (4.16) when

$$1 - \Phi\left(\frac{\Delta p}{\sqrt{2\nu}}\right) > 1 - \Phi\left(\frac{\Delta p_s}{\sqrt{2\nu_s}}\right) \iff \Phi\left(\frac{\Delta p}{\sqrt{2\nu}}\right) < \Phi\left(\frac{\Delta p_s}{\sqrt{2\nu_s}}\right)$$

which boils down to

$$\frac{\Delta p}{\sqrt{2\nu}} < \frac{\Delta p_s}{\sqrt{2\nu_s}} \iff \frac{\Delta p_s}{\Delta p} > \sqrt{\frac{\nu_s}{\nu}} > 1 \tag{4.18}$$

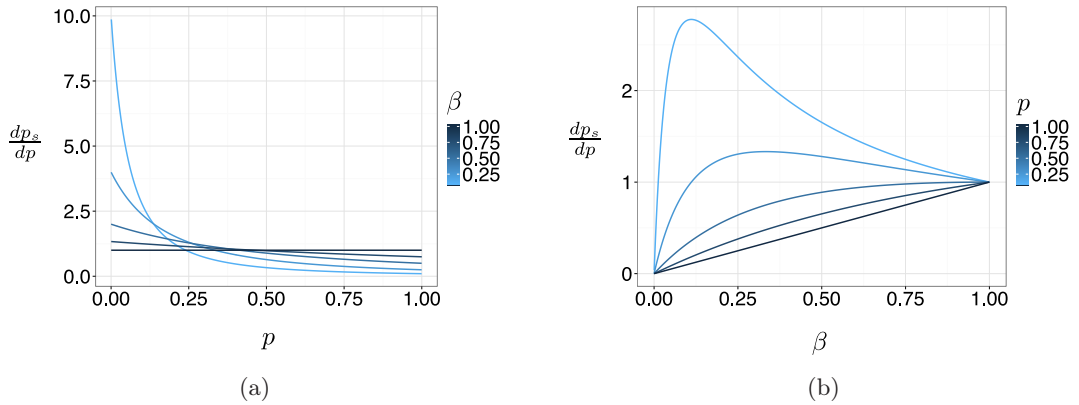
since Φ is monotone non decreasing and we have assumed that $\nu_s > \nu$. Then from (4.18), it follows that undersampling is useful in terms of more accurate ranking if:

$$\frac{dp_s}{dp} > \sqrt{\frac{\nu_s}{\nu}} \tag{4.19}$$

or, using (4.13), when:

$$\frac{\beta}{(p + \beta(1-p))^2} > \sqrt{\frac{\nu_s}{\nu}} \tag{4.20}$$

[3] We assume that the bias of $\hat{p}_1$ and $\hat{p}_2$ is similar: $E[\epsilon_2 - \epsilon_1] = b_2 - b_1 = 0$.

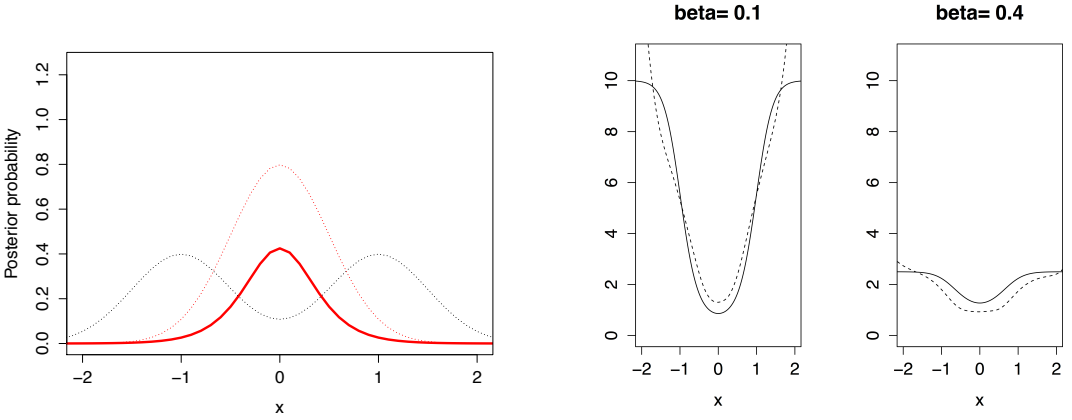


(a) (b)

FIGURE 4.6: Left: $\frac{dp_s}{dp}$ as a function of p. Right: $\frac{dp_s}{dp}$ as a function of β

The value of this inequality depends on several terms: the rate of undersampling β, the ratio of the variances of the two classifiers and the posteriori probability p of the testing point. Also the nonlinearity of the first left-hand term suggests a complex interaction between the involved terms. For instance if we plot the left-hand term of (4.20) as a function of the posteriori probability p (Figure 4.6(a)) and of the value β (Figure 4.6(b)), it appears that the most favorable configurations for undersampling occur for the lowest values of the posteriori probability (e.g. non separable or badly separable configurations) and intermediate β (neither too unbalanced nor too balanced). However if we modify β, this has an impact on the size of the training set and consequently on the right-hand term (i.e. variance ratio) too. Also, though the designer can control the β term, the other two terms vary over the input space. This means that the condition (4.20) does not necessarily hold for all the test points.

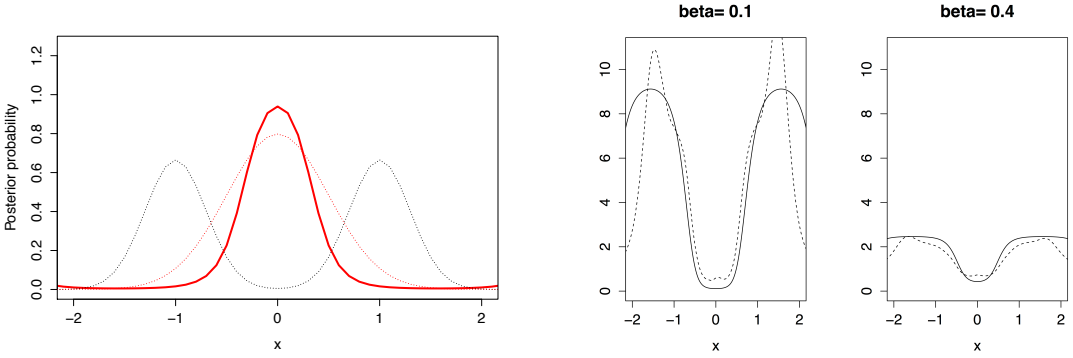
In order to illustrate the complexity of the interaction, let us consider two univariate ($n = 1$) classification tasks where the minority class is normally distributed around zero and the majority class is distributed as a mixture of two Gaussians. Figure 4.7 and 4.8 show the non separable and separable case, respectively: on the left side we plot the class conditional distributions (thin lines) and the posterior distribution of the minority class (thicker line), while on the right side we show the left and the right term of the inequality (4.20) (solid: left-hand term, dotted: right-hand term). What emerges form the figures is that the least separable regions (i.e. the regions where the posteriori of the minority class is low) are also the regions where undersampling helps more. However, the impact of undersampling on the overall accuracy is difficult to be predicted since the regions where undersampling is beneficial change with the characteristics of the classification task and the rate β of undersampling.



(a) Class conditional distributions (thin lines) and the posterior distribution of the minority class (thicker line).

(b) $\frac{dp_s}{dp}$ (solid lines), $\sqrt{\frac{\nu_s}{\nu}}$ (dotted lines).

FIGURE 4.7: Non separable case. On the right we plot both terms of inequality 4.20 (solid: left-hand, dotted: right-hand term) for $\beta = 0.1$ and $\beta = 0.4$



(a) Class conditional distributions (thin lines) and the posterior distribution of the minority class (thicker line).

(b) $\frac{dp_s}{dp}$ (solid lines), $\sqrt{\frac{\nu_s}{\nu}}$ (dotted lines).

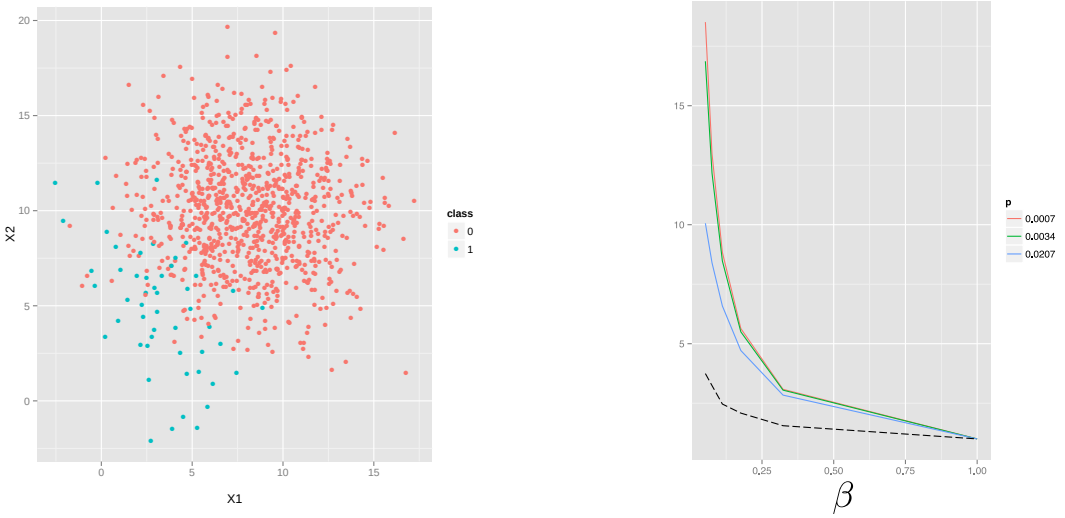
FIGURE 4.8: Separable case. On the right we plot both terms of inequality 4.20 (solid: left-hand, dotted: right-hand term) for $\beta = 0.1$ and $\beta = 0.4$

4.1.4 Experimental validation

In this section we assess the validity of the condition (4.20) by performing a number of tests on synthetic and real datasets. We first simulate two unbalanced synthetic tasks (5% and 25% of positive samples) with overlapping classes and generate a testing set and several training sets from the same distribution. Figures 4.9(a) and Figure 4.11(a) show the distributions of the testing sets for the two tasks.

In order to compute the variance of $\hat{p}$ and $\hat{p}_s$ in each test point, we generate 1000 times a training set and we estimate the conditional probability on the basis of sample mean and covariance.

In Figure 4.9(b) (first task) we plot $\sqrt{\frac{\nu_s}{\nu}}$ (dotted line) and three percentiles $(0.25, 0.5, 0.75)$ of $\frac{dp_s}{dp}$ vs. the rate of undersampling β. It appears that for at least 75% of the testing points, the term $\frac{dp_s}{dp}$ is higher than $\sqrt{\frac{\nu_s}{\nu}}$. In Figure 4.10(a) the points surrounded with a triangle are those one for which $\frac{dp_s}{dp} > \sqrt{\frac{\nu_s}{\nu}}$ hold when $\beta = 0.053$ (balanced dataset). For such samples we expect that ranking returned by undersampling (i.e. based on $\hat{p}_s$) is better than the one based on the original data (i.e. based on $\hat{p}$). The plot shows that undersampling is beneficial in the region where the majority class is situated, which is also the area where we expect to have low values of p. Figure 4.10(b) shows also that this region moves towards the minority class when we do undersampling with $\beta = 0.323$ (90% negatives, 10% positives after undersampling).



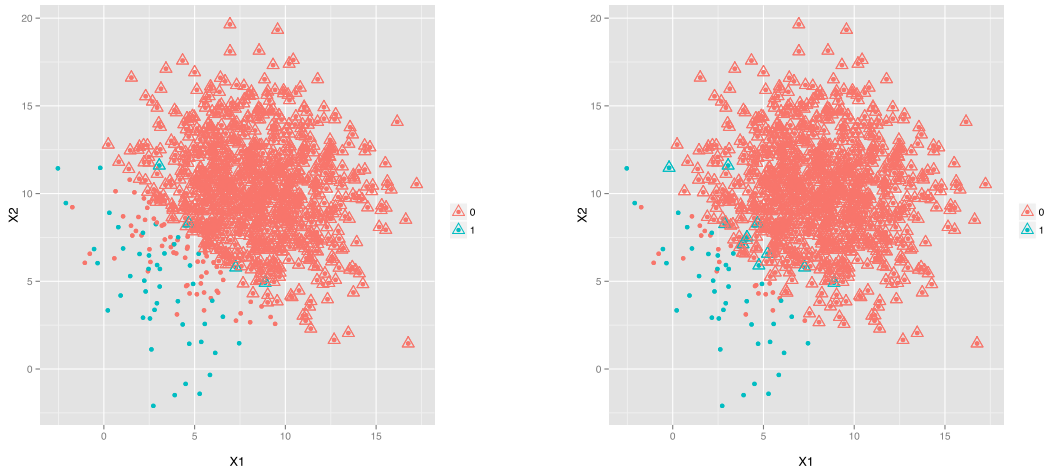
(a) Synthetic dataset 1

(b) $\sqrt{\frac{\nu_s}{\nu}}$ and $\frac{dp_s}{dp}$ for different β

FIGURE 4.9: Left: distribution of the testing set where positives account for 5% of the total. Right: plot of $\frac{dp_s}{dp}$ percentiles (25^{th}, 50^{th} and 75^{th}) and of $\sqrt{\frac{\nu_s}{\nu}}$ (black dashed).

In order to measure the quality of the rankings based on $\hat{p}_s$ and $\hat{p}$ we compute the Kendall rank correlation of the two estimates with p, which is the true posterior probability of the testing set that defines the correct ordering. In Table 4.1 we show the ranking correlations of $\hat{p}_s$ (and $\hat{p}$) with p for the samples where the condition (4.20) (first five rows) holds and where it does not (last five rows). The results indicate that points for which condition (4.20) is satisfied have indeed better ranking with $\hat{p}_s$ than $\hat{p}$.

We repeated the experiments for the second task having a larger proportion of positives (25%) (dataset 2 in Figure 4.11(a)). From the Figure 4.11(b), plotting $\frac{dp_s}{dp}$ and $\sqrt{\frac{\nu_s}{\nu}}$ as a function of β, it appears that only the first two percentiles are over $\sqrt{\frac{\nu_s}{\nu}}$. This means that less points of the testing set satisfy the condition (4.20). This is confirmed from the results in Table 4.2 where it appears that the benefit due to undersampling is less significant than for the first task.



(a) Undersampling with $\beta = 0.053$ (b) Undersampling with $\beta = 0.323$

FIGURE 4.10: Regions where undersampling should work. Triangles indicate the testing samples where the condition (4.20) holds for the dataset in Figure 4.9.

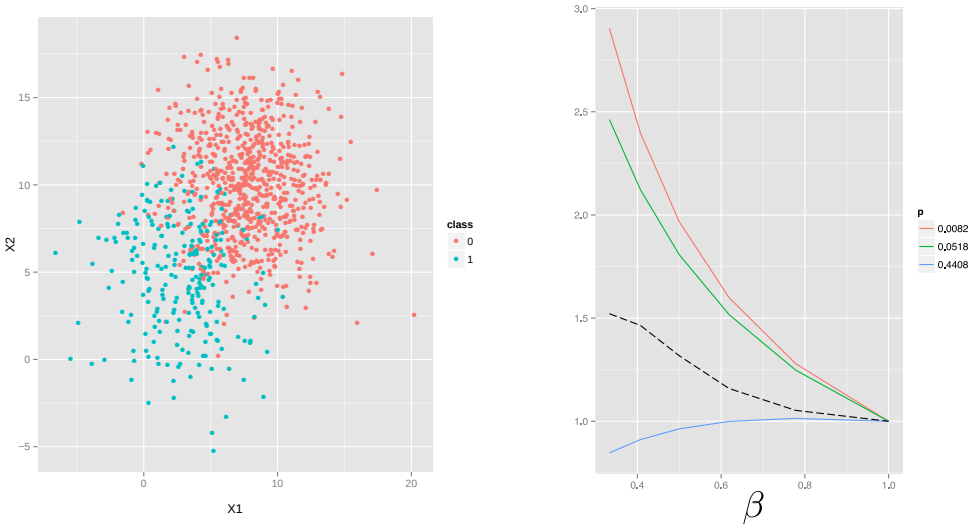
TABLE 4.1: Classification task in Figure 4.9: Ranking correlation between the posterior probability $\hat{p}$ ($\hat{p}_s$) and p for different values of β. The value K (K_s) denotes the Kendall rank correlation without (with) undersampling. The first (last) five lines refer to samples for which the condition (4.20) is (not) satisfied.

β	K	K_s	$K_s - K$	%points satisfying (4.20)
0.053	0.298	0.749	0.451	88.8
0.076	0.303	0.682	0.379	89.7
0.112	0.315	0.619	0.304	91.2
0.176	0.323	0.555	0.232	92.1
0.323	0.341	0.467	0.126	93.7
0.053	0.749	0.776	0.027	88.8
0.076	0.755	0.773	0.018	89.7
0.112	0.762	0.764	0.001	91.2
0.176	0.767	0.761	-0.007	92.1
0.323	0.768	0.748	-0.020	93.7

Now we assess the validity of the condition (4.20) on a number of real unbalanced binary classification tasks obtained by transforming some datasets from the UCI repository [1] (Table 4.3)[4].

Given the unavailability of the conditional posterior probability function, we first approximate p by fitting a Random Forest over the entire dataset in order to compute the left-hand term of (4.20). Then we use a bootstrap procedure to estimate $\hat{p}$ and apply

[4] Transformed datasets are available at `http://www.ulb.ac.be/di/map/adalpozz/imbalanced-datasets.zip`



(a) Synthetic dataset 2

(b) $\sqrt{\frac{\nu_s}{\nu}}$ and $\frac{dp_s}{dp}$ for different β

FIGURE 4.11: Left: distribution of the testing set where positives account for 25% of the total. Right: plot of $\frac{dp_s}{dp}$ percentiles (25^{th}, 50^{th} and 75^{th}) and of $\sqrt{\frac{\nu_s}{\nu}}$ (black dashed).

TABLE 4.2: Classification task in Figure 4.11: Ranking correlation between the posterior probability $\hat{p}$ ($\hat{p}_s$) and p for different values of β. The value K (K_s) denotes the Kendall rank correlation without (with) undersampling. The first (last) five lines refer to samples for which the condition (4.20) is (not) satisfied.

β	K	K_s	$K_s - K$	% points satisfying (4.20)
0.333	0.586	0.789	0.202	66.4
0.407	0.588	0.761	0.172	66.6
0.500	0.605	0.738	0.133	68.1
0.619	0.628	0.715	0.087	70.3
0.778	0.653	0.693	0.040	73
0.333	0.900	0.869	-0.030	66.4
0.407	0.899	0.875	-0.024	66.6
0.500	0.894	0.874	-0.020	68.1
0.619	0.885	0.869	-0.016	70.3
0.778	0.870	0.856	-0.014	73

undersampling to the original dataset to estimate $\hat{p}_s$. We repeat bootstrap and undersampling 100 times to compute the right hand term $\sqrt{\frac{\nu_s}{\nu}}$. This allows us to define the subsets of points for which the condition (4.20) holds.

Figure 4.12 reports the difference between Kendall rank correlation of $\hat{p}_s$ and $\hat{p}$, averaged over different levels of undersampling (proportions of majority vs. minority: $90/10, 80/20, 60/40, 50/50$). Higher difference means that $\hat{p}_s$ returns a better ordering than $\hat{p}$ (assuming that the ranking provided by p is correct). The plot distinguishes between samples for which condition (4.20) is satisfied and not. In general we see that points with a positive difference corresponds to those having the condition satisfied and

TABLE 4.3: Selected datasets from the UCI repository [1]

Datasets	N	N^+	N^-	N^+/N
ecoli	336	35	301	0.10
glass	214	17	197	0.08
letter-a	20000	789	19211	0.04
letter-vowel	20000	3878	16122	0.19
ism	11180	260	10920	0.02
letter	20000	789	19211	0.04
oil	937	41	896	0.04
page	5473	560	4913	0.10
pendigits	10992	1142	9850	0.10
PhosS	11411	613	10798	0.05
satimage	6430	625	5805	0.10
segment	2310	330	1980	0.14
boundary	3505	123	3382	0.04
estate	5322	636	4686	0.12
cam	18916	942	17974	0.05
compustat	13657	520	13137	0.04
covtype	38500	2747	35753	0.07

the opposite for negative differences. These results seem to confirm the experiments with synthetic data, where a better ordering is given by $\hat{p}_s$ when the condition (4.20) holds.

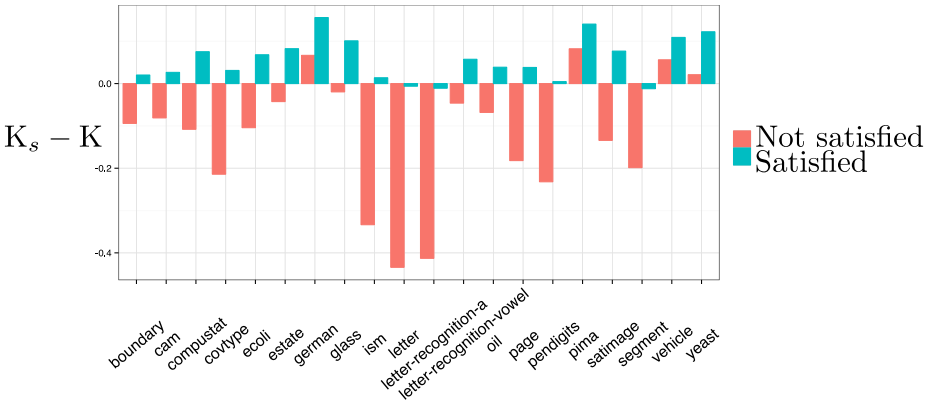


FIGURE 4.12: Difference between the Kendall rank correlation of $\hat{p}_s$ and $\hat{p}$ with p, namely K_s and K, for points having the condition (4.20) satisfied and not. K_s and K are calculated as the mean of the correlations over all βs.

In Figure 4.13 we show the ratio of samples in each dataset satisfying condition (4.20) averaged over all the βs. The proportion of points in which undersampling is useful changes heavily with the dataset considered. For example, in the datasets *vehicle, yeast, german* and *pima*, underdamping returns a better ordering for more than 80% of the samples, while the proportion drops to less than 50% in the *page* dataset.

This seems to confirm our intuition that the right amount of undersampling depends on the classification task (e.g. degree of non separability), the learning algorithm and the targeted test set. It follows that there is no reason to believe that undersampling until the two classes are perfectly balanced is the default strategy to adopt.

It is also worth to remark that the check of the condition (4.20) is not easy to be done, since it involves the estimation of $\sqrt{\frac{\nu_s}{\nu}}$ (ratio of the variance of the classifier before and after undersampling) and of $\frac{dp_s}{dp}$, which demands the knowledge of the true posterior probability p. In practice since p is unknown in real datasets, we can only rely on a data driven approximation of $\frac{dp_s}{dp}$. Also the estimation of $\sqrt{\frac{\nu_s}{\nu}}$ is a hard statistical problem, as known in the statistical literature on ratio estimation [221].

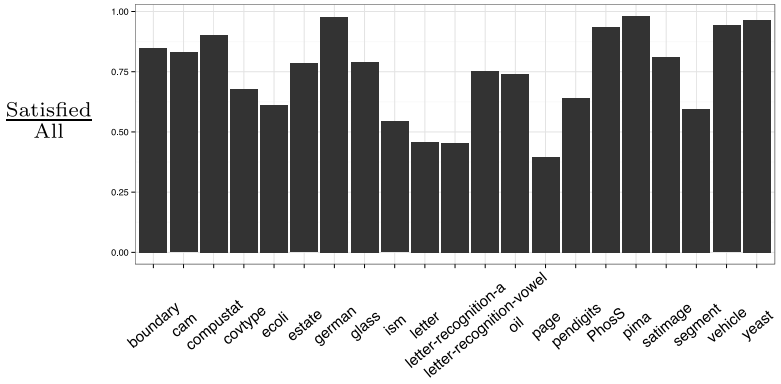


FIGURE 4.13: Ratio between the number of samples satisfying condition (4.20) and all the instances available in each dataset averaged over all the βs.

4.1.5 Discussion

Undersampling has become the de facto strategy to deal with skewed distributions, but, though easy to be justified, it conceals two major effects: i) it increases the variance of the classifier and ii) it produces warped posterior probabilities. The first effect is typically addressed by the use of averaging strategies (e.g. UnderBagging [117]) to reduce the variability while the second requires the calibration of the probability to the new priors of the testing set [134]. Despite the popularity of undersampling for unbalanced classification tasks, it is not clear how these two effects interact and when undersampling leads to better accuracy in the classification task.

In this first part of the chapter, we aimed to analyze the interaction between undersampling and the ranking error of the posterior probability. We derive the condition (4.20) under which undersampling can improve the ranking and we show that when it is satisfied, the posterior probability obtained after sampling returns a more accurate ordering of testing instances. To validate our claim we used first synthetic and then real datasets, and in both cases we registered a better ranking with undersampling when condition (4.20) was met. It is important to remark how this condition shows that the beneficial impact of undersampling is strongly dependent on the nature of the classification task (degree of imbalance and non separability), on the variance of the classifier and as a consequence is extremely dependent on the specific test point. We think that this result sheds light

on the reason why several discordant results have been obtained in the literature about the effectiveness of undersampling in unbalanced tasks.

However, the practical use of this condition is not straightforward since it requires the knowledge of the posteriori probability and of the ratio of variances before and after undersampling. It follows that this result should be used mainly as a warning against a naive use of undersampling in unbalanced tasks and should suggest instead the adoption of specific adaptive selection techniques (e.g. racing [27], see Section 4.3) to perform a case-by-case use (and calibration) of undersampling.

4.2 Using calibrated probability with undersampling

In the previous section we showed with equation (4.7) that undersampling induces an artificial bias (warping) into the posterior probabilities returned by a model. We now demonstrate how to correct for this bias using the equations derived in Section 4.1.1. Although this method does not affect the ranking order returned by the posterior probability, it significantly impacts the classification accuracy and probability calibration. We use Bayes Minimum Risk theory [30] to find the correct classification threshold and show how to adjust it after undersampling. Experiments on several real-world unbalanced datasets validate our results.

4.2.1 Adjusting posterior probabilities to new priors

The first part of the chapter showed that undersampling is responsible for a drift in posterior probabilities and induces warped probability estimates (see Section 4.1). However, the first and most direct effect of undersampling is the change in the class priors. To show this effect, let us use an illustrative example.

Let's suppose we have an unbalanced problem where the positives account for 10% of 10,000 observations (i.e., we have 1,000 positives and 9,000 negatives). Suppose we want to have a balanced dataset $\beta = \frac{N^+}{N^-} \approx 0.11$, where $\approx 88.9\%$ (8000/9000) of the negative instances are discharged. Table 4.4 shows how, by reducing β, the original unbalanced dataset becomes more balanced and smaller as negative instances are removed. After undersampling, the number of negatives is $N_s^- = \beta N^-$, while the number of positives stays the same $N_s^+ = N^+$. The percentage of negatives ($perc^-$) in the dataset decreases as $N_s^- \to N^+$.

After a classification model is learned on a balanced training set, it is normally used to predict a testing set, which is likely to have an unbalanced distribution similar to the

TABLE 4.4: Undersampling a dataset with 1,000 positives in 10,000 observations. N_s defines the size of the dataset after undersampling and N_s^- (N_s^+) the number of negative (positive) instances for a given β. When $\beta = 0.11$ the negative samples represent 50% of the observations in the dataset.

N_s	N_s^-	N_s^+	β	$perc^-$
2,000	1,000	1,000	0.11	50.00
2,800	1,800	1,000	0.20	64.29
3,700	2,700	1,000	0.30	72.97
4,600	3,600	1,000	0.40	78.26
5,500	4,500	1,000	0.50	81.82
6,400	5,400	1,000	0.60	84.38
7,300	6,300	1,000	0.70	86.30
8,200	7,200	1,000	0.80	87.80
9,100	8,100	1,000	0.90	89.01
10,000	9,000	1,000	1.00	90.00

original training set. This means that the posterior probability of a model learned on the balanced training set should be adjusted for the change in priors between the training and testing sets. In this section we propose to use equation (4.8) to correct the posterior probability estimates after undersampling. Let us call p' the bias-corrected probability obtained from p_s using (4.8):

$$p' = \frac{\beta p_s}{\beta p_s - p_s + 1} \tag{4.21}$$

Equation (4.21) can be seen as a special case of the framework proposed by Saerens et al. [134] and Elkan [35] for correcting the posterior probability in the case of testing and training sets sharing the same priors.

Let $p_t = p(y_t = +|x_t)$ be the posterior probability for a testing instance (x_t, y_t), where the testing set has priors: $\pi_t^- = \frac{N_t^-}{N_t}$ and $\pi_t^+ = \frac{N_t^+}{N_t}$. In the unbalanced training set we have $\pi^- = \frac{N^-}{N}$, $\pi^+ = \frac{N^+}{N}$ and after undersampling the training set $\pi_s^- = \frac{\beta N^-}{N^+ + \beta N^-}$, $\pi_s^+ = \frac{N^+}{N^+ + \beta N^-}$. If we assume that the class conditional distributions $\mathcal{P}(x|+)$ and $\mathcal{P}(x|-)$ remain the same between the training and testing sets, Saerens et al. [134] show that, given different priors between the training and testing sets, the posterior probability can be corrected with the following equation:

$$p_t = \frac{\frac{\pi_t^+}{\pi_s^+} p_s}{\frac{\pi_t^+}{\pi_s^+} p_s + \frac{\pi_t^-}{\pi_s^-}(1 - p_s)} \tag{4.22}$$

Let us assume that the training and testing sets share the same priors: $\pi_t^+ = \pi^+$ and $\pi_t^- = \pi^-$:

$$p_t = \frac{\frac{\pi^+}{\pi_s^+} p_s}{\frac{\pi^+}{\pi_s^+} p_s + \frac{\pi^-}{\pi_s^-}(1 - p_s)}$$

Then, since

$$\frac{\pi^+}{\pi_s^+} = \frac{\frac{N^+}{N^+ + N^-}}{\frac{N^+}{N^+ + \beta N^-}} = \frac{N^+ + \beta N^-}{N^+ + N^-} \tag{4.23}$$

$$\frac{\pi^-}{\pi_s^-} = \frac{\frac{N^-}{N^+ + N^-}}{\frac{\beta N^-}{N^+ + \beta N^-}} = \frac{N^+ + \beta N^-}{\beta(N^+ + N^-)} \tag{4.24}$$

we can write

$$p_t = \frac{\frac{N^+ + \beta N^-}{N^+ + N^-} p_s}{\frac{N^+ + \beta N^-}{N^+ + N^-} p_s + \frac{N^+ + \beta N^-}{\beta(N^+ + N^-)}(1 - p_s)}$$

$$p_t = \frac{p_s}{p_s + \frac{(1-p_s)}{\beta}} = \frac{\beta p_s}{\beta p_s - p_s + 1}$$

Hence, the transformation proposed by Saerens et al. [134] is essentially equivalent to (4.21). Similarly, Elkan [35] proposes to adjust the posterior probability after undersampling with the following equation:

$$p_t = \pi_t^+ \frac{p_s - \pi_s^+ p_s}{\pi_s^+ - \pi_s^+ p_s + \pi_t^+ p_s - \pi_t^+ \pi_s^+} \tag{4.25}$$

$$p_t = \frac{(1 - \pi_s^+) p_s}{\frac{\pi_s^+}{\pi_t^+}(1 - p_s) + p_s - \pi_s^+}$$

using equation (4.23) and the assumption that $\pi_t^+ = \pi^+$ and $\pi_t^- = \pi^-$:

$$p_t = \frac{\frac{\beta N^-}{N^+ + \beta N^-} p_s}{\frac{N^+ + N^-}{N^+ + \beta N^-}(1 - p_s) + p_s - \frac{N^+}{N^+ + \beta N^-}}$$

$$p_t = \frac{\beta N^- p_s}{(N^+ + N^-)(1 - p_s) + (N^+ + \beta N^-) p_s - N^+}$$

$$p_t = \frac{\beta N^- p_s}{N^- - N^- p_s + \beta N^- p_s} = \frac{\beta p_s}{\beta p_s - p_s + 1}$$

Equation (4.25) is equivalent to (4.21) and therefore to the one proposed by Saerens et al. [134].

In summary, when we know the priors in the testing set we can correct the probability with Elkan's and Saerens' equations. However, these probabilities are usually unknown and must be estimated. If we make the assumption that training and testing have the same priors we can use (4.21) for calibrating p_s. Note that the above transformation will not affect the ranking produced by p_s. Equation (4.21) defines a monotone transformation, hence the ranking of p_s will be the same as p'. While p is estimated using all the samples in the unbalanced dataset, p_s and p' are computed considering a subset of the original samples and therefore their estimations are subjected to higher variance [213].

4.2.2 Warping correction and classification threshold adjustment

As previously described in Section 2.1.2, a classifier typically defines the optimal class of a sample as the one minimizing the risk. In practice, this translates into calculating the posterior probability and predicting an instance as positive or negative when the probability is above a certain threshold. If we assume that there is no cost in case of correct prediction, then from (2.9) the threshold minimizing the risk is:

$$\tau = \frac{l_{1,0}}{l_{1,0} + l_{0,1}}$$

where $l_{i,j}$ is the cost occurred in deciding i when the true class is j. In an unbalanced problem, the cost of missing a positive instance (false negative) is usually higher than the cost of missing a negative (false positive). If the costs of a false negative and false positive are unknown, a natural solution is to set these costs using the priors (π^- and π^+). Let $l_{1,0} = \pi^+$ and $l_{0,1} = \pi^-$. Then, since $\pi^- > \pi^+$ we have $l_{0,1} > l_{1,0}$ as desired. We can then write

$$\tau = \frac{l_{1,0}}{l_{1,0} + l_{0,1}} = \frac{\pi^+}{\pi^+ + \pi^-} = \pi^+ \tag{4.26}$$

since $\pi^+ + \pi^- = 1$. This is also the optimal threshold in a cost-sensitive application where the goal is to minimize overall costs and the misclassification costs are defined using the priors [35].

Even if undersampling produces warped probability estimates, it is often used to balance datasets with skewed class distributions because several classifiers have empirically shown better performance when trained on a balanced dataset [88, 90]. Let τ_s denote the threshold used to classify an observation after undersampling, from (4.26) we have $\tau_s = \pi_s^+$, where π_s^+ is the positive class prior after undersampling. In the case of undersampling with $\beta = \frac{N^+}{N^-}$ (balanced training set) we have $\tau_s = 0.5$.

When correcting p_s with (4.21), we must also correct the probability threshold to maintain the predictive accuracy defined by τ_s (this is needed otherwise we would use different misclassification costs for p'). Let τ' be the threshold for the unbiased probability p'. From Elkan [35]:

$$\frac{\tau'}{1-\tau'}\frac{1-\tau_s}{\tau_s} = \beta \tag{4.27}$$

$$\tau' = \frac{\beta\tau_s}{(\beta-1)\tau_s + 1} \tag{4.28}$$

Using $\tau_s = \pi_s^+$, (4.28) becomes:

$$\tau' = \frac{\beta\pi_s^+}{(\beta-1)\pi_s^+ + 1}$$

$$\tau' = \frac{\beta \frac{N^+}{N^+ + \beta N^-}}{(\beta - 1)\frac{N^+}{N^+ + \beta N^-} + 1} = \frac{N^+}{N^+ + N^-} = \pi^+$$

The optimal threshold to use with p' is equal to the one for p. As an alternative to classifying observations with p_s and τ_s, we can obtain equivalent results with p' and τ'. In summary, as a result of undersampling, a higher number of observations are predicted as positive, but the posterior probabilities are biased due to a change in the priors. Equation (4.28) allows us find the threshold that guarantees equal accuracy after the posterior probability correction. Therefore, in order to classify observations with unbiased probabilities after undersampling, we have to first obtain p' from p_s with (4.21) and then use τ' as a classification threshold.

4.2.3 Experimental results

The rational of the following experiments is to compare the probability estimates of two models, one learned in the presence and the other in the absence of undersampling, and test the benefit of probability calibration after undersampling. We use G-mean as a measure of classification accuracy, AUC to assess the quality of the ranking produced by the probability and Brier Score (BS) as a measure of probability calibration (see Section 2.1.4). In our experiments we use the same datasets of Section 4.1.4 (Table 4.3). For each dataset we used 10-fold CV to test our models and we repeated the CV 10 times. In particular, we used a stratified CV, where the class proportion in the datasets is kept the same over all the folds. As the original datasets are unbalanced, the resulting folds are unbalanced as well. For each fold of CV we learn two models: one using all the observations and the other with the ones remaining after undersampling. Then both models are tested on the same testing set (Figure 4.14). We used several supervised classification algorithms available in R [25] with default parameters: RF [222], SVM [223] and Logit Boost (LB) [224].

We denote as $\hat{p}_s$ and $\hat{p}$ the posterior probability estimates obtained with and without undersampling and as $\hat{p}'$ the bias-corrected probability obtained from $\hat{p}_s$ with equation (4.21). Let τ, τ_s and τ' be the probability thresholds used for $\hat{p}$, $\hat{p}_s$ and $\hat{p}'$ respectively, where $\tau = \pi^+$, $\tau_s = \pi_s^+$ and $\tau' = \pi^+$. The goal of these experiments is to compare which probability estimates return the highest ranking (AUC), calibration (BS) and classification accuracy (G-mean) when coupled with the thresholds defined before. In undersampling, the amount of sampling defined by β is usually set to be equal to $\frac{N^+}{N^-}$, leading to a balanced dataset where $\pi_s^+ = \pi_s^- = 0.5$. However, there is no reason to believe that this is the optimal sampling rate. Often, the optimal rate can be found only a posteriori after trying different values of β. For this reason we replicate the CV with

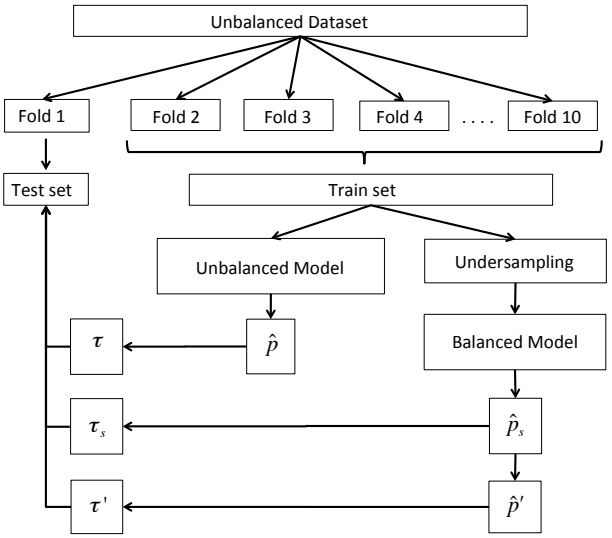


FIGURE 4.14: Learning framework for comparing models with and without undersampling using CV. We use one fold of the CV as testing set and the others for training, and iterate the framework to use all the folds once for testing.

different β such that $\{\frac{N_+}{N_-} \leq \beta \leq 1\}$ and for each CV the accuracy is computed as the average G-mean (or AUC) over all the folds.

In Table 4.5 we report the results over all the datasets. For each dataset, we rank the probability estimates $\hat{p}_s$, $\hat{p}$ and $\hat{p}'$ from the worst to the best performing for different values of β. We then sum the ranks over all the values of β and over all datasets. More formally, let $R_{i,k,b} \in \{1, 2, 3\}$ be the rank of probability i on dataset k when $\beta = b$. The probability with the highest accuracy in k when $\beta = b$ has $R_{i,k,b} = 3$ and the one with the lowest has $R_{i,k,b} = 1$. Then the sum of ranks for the probability i is defined as $\sum_k \sum_b R_{i,k,b}$. The higher the sum, the higher the number of times that one probability has higher accuracy than the others.

For AUC, a higher rank sum means a higher AUC and hence a better ranking returned by the probability. Similarly, with G-mean, a higher rank sum corresponds to higher predictive accuracy. However, in the case of BS, a higher rank sum means poorer probability calibration (larger bias). Table 4.5 has in bold the probabilities with the best rank sum according to the different metrics. For each metric and classifier it reports the p-values of the paired t-test based on the ranks between $\hat{p}$ and $\hat{p}'$ and between $\hat{p}$ and $\hat{p}_s$.

In terms of AUC, we see that $\hat{p}_s$ and $\hat{p}'$ have better performances than $\hat{p}$ for LB and SVM. The rank sum is the same for $\hat{p}_s$ and $\hat{p}'$ since the two probabilities are linked by a monotone transformation (equation (4.21)). If we look at G-mean, $\hat{p}_s$ and $\hat{p}'$ return better accuracy than $\hat{p}$ two times out of three. In this case, the rank sums of $\hat{p}_s$ and $\hat{p}'$ are the same since we used τ_s and τ' as the classification threshold, where τ' is obtained

from τ_s using (4.28). If we look at the p-values, we can strongly reject the null hypothesis that the accuracy of $\hat{p}_s$ and $\hat{p}$ are from the same distribution. For all classifiers, $\hat{p}$ is the probability estimate with the best calibration (lower rank sum with BS), followed by $\hat{p}'$ and $\hat{p}_s$. The rank sum of $\hat{p}'$ is always lower than the one of $\hat{p}_s$, indicating that $\hat{p}'$ has lower bias than $\hat{p}_s$. This result confirms our claim that equation (4.21) allows one to reduce the bias introduced by undersampling.

In summary from this experiment we can conclude that undersampling does not always improve the ranking or classification accuracy of an algorithm, but when it is the case we should use $\hat{p}'$ instead of $\hat{p}_s$ because the first has always better calibration.

TABLE 4.5: Sum of ranks and p-values of the paired t-test between the ranks of $\hat{p}$ and $\hat{p}'$ and between $\hat{p}$ and $\hat{p}_s$ for different metrics. In **bold** the probabilities with the best rank sum (higher for AUC and G-mean, lower for BS).

Metric	Algo	$\sum R_{\hat{p}}$	$\sum R_{\hat{p}_s}$	$\sum R_{\hat{p}'}$	$\rho(R_{\hat{p}}, R_{\hat{p}_s})$	$\rho(R_{\hat{p}}, R_{\hat{p}'})$
AUC	LB	22,516	**23,572**	**23,572**	0.322	0.322
AUC	RF	**24,422**	22,619	22,619	0.168	0.168
AUC	SVM	19,595	**19,902.5**	**19,902.5**	0.873	0.873
G-mean	LB	**23,281**	23,189.5	23,189.5	0.944	0.944
G-mean	RF	22,986	**23,337**	**23,337**	0.770	0.770
G-mean	SVM	19,550	**19,925**	**19,925**	0.794	0.794
BS	LB	**19809.5**	29448.5	20402	0.000	0.510
BS	RF	**18336**	28747	22577	0.000	0.062
BS	SVM	**17139**	23161	19100	0.001	0.156

We now consider a real-world dataset, composed of credit card transactions from September 2013 made available by our industrial partner.[5] It contains a subset of online transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, where the positive class (frauds) account for 0.172% of all transactions, and the minimum value of β is ≈ 0.00173. In Figure 4.15 we have the AUC for different values of β. The boxplots of $\hat{p}_s$ and $\hat{p}'$ are identical because of (4.21), they increase with $\beta \to \frac{N^+}{N^-}$ and have higher median than the one of $\hat{p}$. This example shows how in case of extreme class imbalance, undersampling can improve predictive accuracy of several classification algorithms.

In Figure 4.16 we have the BS for different values of β. The boxplots of $\hat{p}'$ show in general smaller calibration error (lower BS) than those of $\hat{p}_s$ and the latter have higher BS especially for small values of β. This supports our previous results, which found that the loss in probability calibration for $\hat{p}_s$ is greater the stronger the undersampling.

[5]The dataset is available at `http://www.ulb.ac.be/di/map/adalpozz/data/creditcard.Rdata`

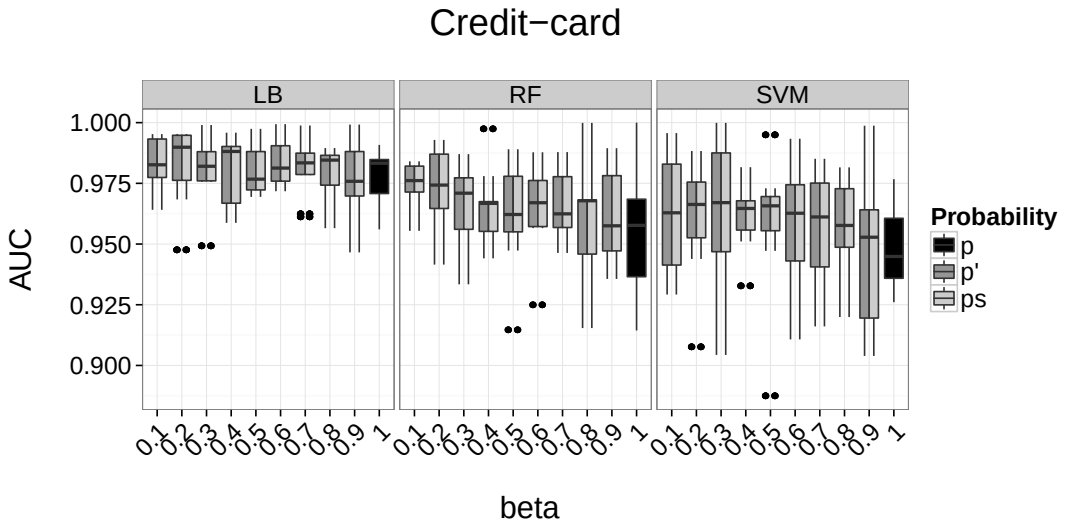


FIGURE 4.15: Boxplot of AUC for different values of β in the *Credit-card* dataset.

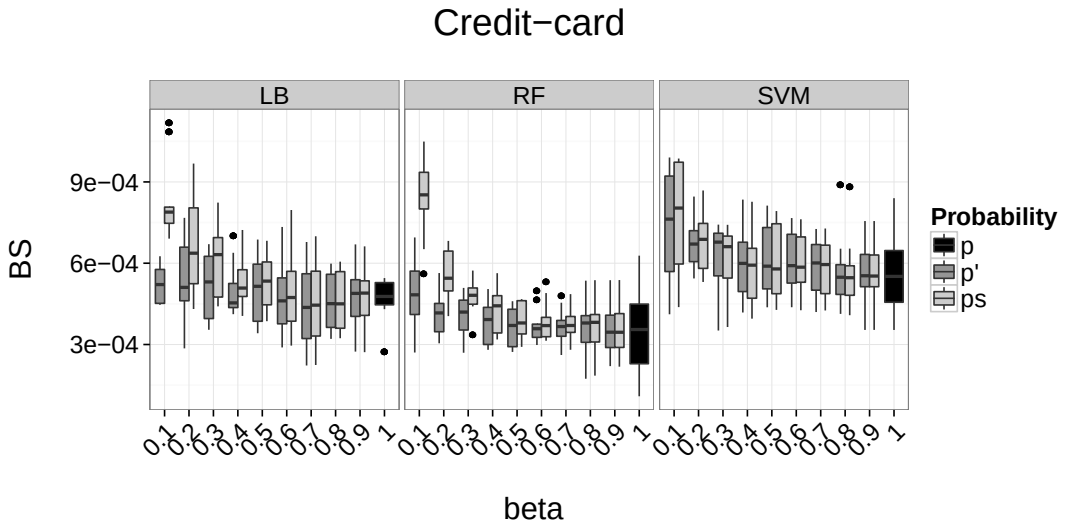


FIGURE 4.16: Boxplot of BS for different values of β in the Credit-card dataset.

4.2.4 Discussion

The warping due to the instance selection procedure in undersampling is essentially equivalent to the bias that occurs with a change in the priors when class-within distributions remain stable. With undersampling, we create a different training set, where the classes are less unbalanced. However, if we make the assumption that the training and testing sets come from the same distribution, it follows that the probability estimates obtained after undersampling are biased. As a result of undersampling, the posterior probability $\hat{p}_s$ is shifted away from the true distribution, and the optimal separation boundary moves towards the majority class so that more cases are classified into the minority class.

By making the assumptions that prior probabilities do not change from training and testing, i.e. they both come form the same data generating process, we propose the transformation given in equation (4.21), which allows us to remove the drift in $\hat{p}_s$ due to undersampling.

The bias on $\hat{p}_s$ registered by BS gets larger for small values of β, which means stronger undersampling produces probabilities with poorer calibration (larger loss). With synthetic, UCI and *Credit-card* datasets, the drift-corrected probability ($\hat{p}'$) has significantly better calibration than $\hat{p}_s$ (lower Brier Score).

Even if undersampling produces poorly calibrated probability estimates $\hat{p}_s$, several studies have shown that it often provides better predictive accuracy than $\hat{p}$ [88, 90]. To improve the calibration of $\hat{p}_s$ we propose to use $\hat{p}'$ since this transformation does not affect the ranking. In order to maintain the accuracy obtained with $\hat{p}_s$ and the probability threshold τ_s, we proposed to use $\hat{p}'$ together with τ' to account for the change in priors. By changing the undersampling rate β we give different costs to false positives and false negatives, combining $\hat{p}'$ with τ' allows one to maintain the same misclassification costs of a classification strategy with $\hat{p}_s$ and τ_s for any value of β.

Finally, we considered a highly unbalanced dataset (*Credit-card*), where the minority class accounts for only 0.172% of all observations. In this dataset, the large improvement in accuracy obtained with undersampling was coupled with poor calibrated probabilities (large BS). By correcting the posterior probability and changing the threshold we were able to improve calibration without losing predictive accuracy.

4.3 Racing for sampling methods selection

As already seen in Section 4.1, the degree of imbalance is not the only factor that determines the difficulty of a classification/detection task. Another influential factor is the amount of overlapping of the classes of interest [86]. Prati [84] showed that class unbalance, by itself, does not seem to be a problem. Most studies [78, 82, 84, 106] propose one method that seems to work well under certain conditions, however there is no empirical evidence than one technique is superior to all the others. In general the best method does not exits, however in some cases some techniques are better than others, this is known in the literature as *no-free-lunch Theorem* [225, 226].

All these support the idea that under different conditions, such as distinct datasets, algorithms, metrics, the best methods may change. Since in real large tasks it is hard to know a priori the nature of the unbalanced tasks, the user is recommended to test all

techniques with a consequent computational overhead. We make an exhaustive comparison of these methods on a real credit-card fraud dataset and nine public benchmark datasets. The results show that there is no balancing technique which is consistently the best one and that the best method depends on the algorithm applied as well as the dataset used. For this reason, we propose the adoption of a racing strategy [227] to automatically select the most adequate technique for a given dataset. The rationale of the racing strategy consists in testing multiple balancing strategies on a subset of the dataset and to remove progressively the alternatives that are significantly worse. Our results show that by adopting a racing strategy we are able to select in an efficient manner either the best balancing method or a method that is not significantly different from the best one. Moreover, racing is able to reduce consistently the computation needed before finding the right methods for the dataset.

4.3.1 Racing for strategy selection

The variety of approaches discussed in Section 3.1 suggests that in a real situation where we have no prior information about the data distribution, it is difficult to decide which unbalanced strategy to use. In this case testing all alternatives is not an option either because of the associated computational cost.

A possible solution comes form the adoption of the Racing approach which was proposed in [227] to perform efficiently model selection in a learning task. The principle of Racing consists in testing in parallel a set of alternatives and using a statistical test to determine if an alternative is significantly worse than the others. In that case such alternative is discarded from the competition, and the computational effort is devoted to differentiate the remaining ones. Historically the first example of racing method is called Hoeffding Race since it relies on the Hoeffding theorem to decide when a model is significantly worse than the others. The *F-race* version was proposed in [26] and combines the Friedman test with Hoeffding Races [227] to eliminate inferior candidates as soon as enough statistical evidence arises against them. In F-race, the Friedman test is used to check whether there is evidence that at least one of the candidates is significantly different from others and post-tests are applied to eliminate those candidates that are significantly worse than the best one.

Here we adopt F-Race to search efficiently for the best strategy for unbalanced data. The candidates are assessed on different subsets of data and, each time a new assessment is made, the Friedman test is used to dismiss significantly inferior candidates. We used a 10 fold CV to provide the assessment measure to the race. If a candidate is significantly better than all the others then the race is terminated without the need of using the whole

dataset. In case there is no evidence of worse/better methods, the race terminates when the entire dataset is explored and the best candidate is the one with the best average result.

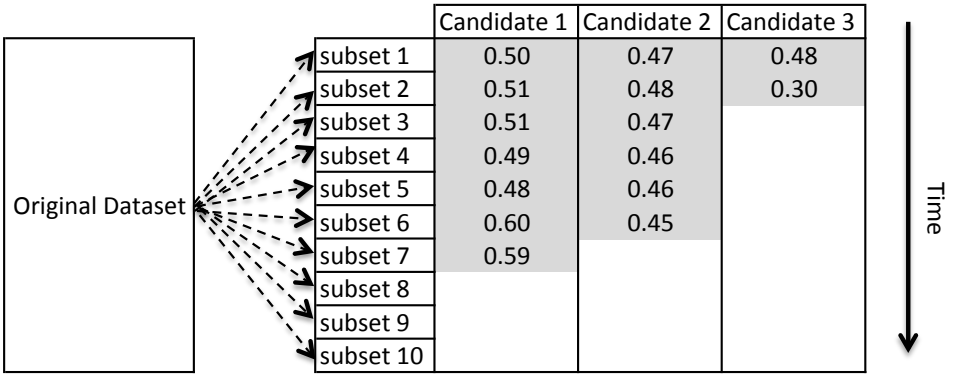


FIGURE 4.17: Illustrative example of Racing: test in parallel a set of candidates using a subset of the dataset and remove from the race those that are significantly worse than the best. The race continues with the remaining candidates until only one is selected.

4.3.2 Experimental results

In these experiments we tested some of the techniques for unbalanced classification discussed in Section 3.1.1 on the datasets of Table 4.3 and the credit card data used in [28]. In particular, we considered the following techniques: undersampling, oversampling, SMOTE, CNN, ENN, NCL, OSS and Tomek Link.

We started by performing a CV for each technique with different classification algorithms: RF [222], Neural Network (NNET) [228], SVM [228], LB [224] and Decision Tree [229]. Figure 4.18 displays the results of the CV for all the algorithms and datasets. For each dataset we compute the average G-mean in the CV and then calculate the average accuracy over all the datasets. We also include the performances in the case of unbalanced datasets as benchmark. In this study, we see that the combination of RF and undersampling appears to return the largest accuracy.

However, the results are highly dependent on the dataset and classifier considered. In Figure 4.19 we show the accuracy on two datasets. In the case of the *cam* dataset undersampling is the best technique for all the classification algorithms used. On the contrary, we see that for *ecoli* dataset, there is not a single technique that clearly outperforms the others.

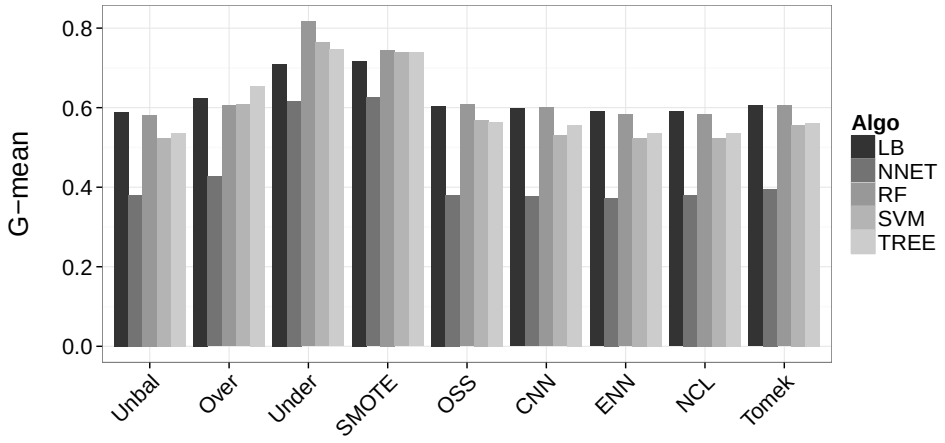


FIGURE 4.18: Comparison of strategies for unbalanced data with different classifiers over all datasets of Table 4.3 in terms of G-mean (the higher the better).

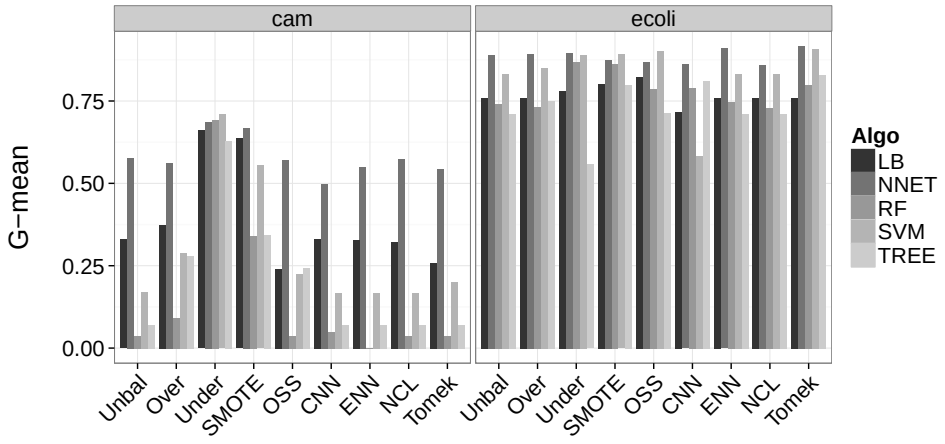


FIGURE 4.19: Comparison of strategies for unbalanced data with different classifiers on *cam* and *ecoli* datasets in terms of G-mean (the higher the better).

In these experiments the classification accuracy was measured in terms of G-mean, but we could have different outcomes when using another metric [27]. In general there is no single strategy that is coherently superior to all the others in all conditions (i.e. algorithm and dataset). Even if sometimes it is possible to find a strategy that is statistically better than others it is computationally demanding testing all strategies on several datasets and algorithms. For this reason in [27] we proposed to adopt the F-race algorithm to automatize the way to select the best strategy for unbalanced data. The algorithm is available in the companion `unbalanced` package [24] available for the R language (see Appendix A). Tables 4.6 and 4.7 show the results of the F-race.

The first thing we notice is that for almost all datasets F-race is able to return the best

Dataset	Exploration	Method	Ntest	% Gain	Mean	Sd
ecoli	Race	Under	46	49	0.836	0.04
	CV	SMOTE	90	-	0.754	0.112
letter-a	Race	Under	34	62	0.952	0.008
	CV	SMOTE	90	-	0.949	0.01
letter-vowel	Race	Under	34	62	0.884	0.011
	CV	Under	90	-	0.887	0.009
letter	Race	SMOTE	37	59	0.951	0.009
	CV	Under	90	-	0.951	0.01
oil	Race	Under	41	54	0.629	0.074
	CV	SMOTE	90	-	0.597	0.076
page	Race	SMOTE	45	50	0.919	0.01
	CV	SMOTE	90	-	0.92	0.008
pendigits	Race	Under	39	57	0.978	0.011
	CV	Under	90	-	0.981	0.006
PhosS	Race	Under	19	79	0.598	0.01
	CV	Under	90	-	0.608	0.016
satimage	Race	Under	34	62	0.843	0.008
	CV	Under	90	-	0.841	0.011
segment	Race	SMOTE	90	0	0.978	0.01
	CV	SMOTE	90	-	0.978	0.01
estate	Race	Under	27	70	0.553	0.023
	CV	Under	90	-	0.563	0.021
covtype	Race	Under	42	53	0.924	0.007
	CV	SMOTE	90	-	0.921	0.008
cam	Race	Under	34	62	0.68	0.007
	CV	Under	90	-	0.674	0.015
compustat	Race	Under	37	59	0.738	0.021
	CV	Under	90	-	0.745	0.017
creditcard	Race	Under	43	52	0.927	0.008
	CV	SMOTE	90	-	0.924	0.006

TABLE 4.6: Comparison of CV and F-race results in terms of G-mean for RF classifier.

method according to CV. In the case where there is no agreement between F-race and CV on the best method, the difference in performances is however not significant. The main advantage of Racing is that bad methods are not tested on the whole dataset reducing the computation needed. Taking into consideration the 8 methods and the unbalanced case, in a 10 fold CV we have 90 tests to make (10 folds x 9 methods). In the case of F-race the number of total tests depends upon how many folds are needed before F-race finds the best method. The *Gain* column of Tables 4.6 and 4.7 shows the computational gain (in percentage of the CV tests) obtained by using F-race. In the case of the *segment* dataset and RF classifier the gain is 0, meaning that the Race did not find significant worse candidates. In all the other cases F-race allows a significant computational saving with no loss in performance.

4.3.3 Discussion

Recent literature in data mining and machine learning has plenty of research works on strategies to deal with unbalanced data. However, a definitive answer on the best

Dataset	Exploration	Method	Ntest	% Gain	Mean	Sd
ecoli	Race	SMOTE	55	39	0.83	0.068
	CV	OSS	90	-	0.613	0.361
letter-a	Race	SMOTE	31	66	0.96	0.009
	CV	SMOTE	90	-	0.961	0.008
letter-vowel	Race	SMOTE	31	66	0.878	0.007
	CV	SMOTE	90	-	0.876	0.005
letter	Race	SMOTE	31	66	0.96	0.007
	CV	SMOTE	90	-	0.96	0.007
oil	Race	Over	90	0	0.356	0.155
	CV	Tomek	90	-	0.311	0.272
page	Race	SMOTE	31	66	0.91	0.015
	CV	SMOTE	90	-	0.905	0.012
pendigits	Race	SMOTE	63	30	0.992	0.003
	CV	SMOTE	90	-	0.992	0.003
PhosS	Race	Under	31	66	0.571	0.008
	CV	Under	90	-	0.509	0.112
satimage	Race	Under	41	54	0.842	0.011
	CV	SMOTE	90	-	0.837	0.012
segment	Race	SMOTE	84	7	0.971	0.021
	CV	CNN	90	-	0.972	0.014
boundary	Race	Under	34	62	0.401	0.051
	CV	Under	90	-	0.423	0.079
estate	Race	Under	31	66	0.56	0.068
	CV	Under	90	-	0.593	0.016
covtype	Race	SMOTE	34	62	0.927	0.007
	CV	SMOTE	90	-	0.924	0.006
cam	Race	Under	31	66	0.684	0.026
	CV	Under	90	-	0.686	0.017
compustat	Race	Under	27	70	0.742	0.012
	CV	Under	90	-	0.741	0.013
creditcard	Race	SMOTE	42	53	0.919	0.011
	CV	Under	90	-	0.916	0.008

TABLE 4.7: Comparison of CV and F-race results in terms of G-mean for SVM classifier.

strategy to adopt is yet to come. Our experimental results support the idea that the final performance is extremely dependent on the data nature and distribution.

This consideration has led us to adopt the F-race strategy where different candidates (unbalanced methods) are tested simultaneously. We have showed that this algorithm is able to select few candidates that perform better than other without exploring the whole dataset. F-race was able to get results similar to the cross validation for most of the datasets.

In general we saw that undersampling and SMOTE together with RF are often the methods returning the larger accuracy. As far as the fraud dataset is concerned, we prefer undersampling over SMOTE because it reduces the dataset set allowing faster training of the classifier. However, as the frauds evolve over the time the same method could become sub-optimal in the future. In this context the F-race contribution to the selection of the best strategy is crucial in order to have a detection system that quickly adapts to the new data distribution. Within the UCI datasets we noticed that some tasks are much easier (high accuracy) than the others and they may not have an unbalanced method

that performs significantly better than the others.

4.4 Conclusion

A standard approach to deal with unbalanced classification tasks is to rebalance the dataset before training a learning algorithm. One of the most straightforward ways to achieve a balanced distribution is to use undersampling, i.e. remove observations from the majority class. Despite the popularity of this technique, no detailed analysis about the impact of undersampling on the accuracy of the final classifier was available yet.

This chapter aims to fill this gap by proposing an integrated analysis of the two elements which have the largest impact on the effectiveness of an undersampling strategy: the increase of the variance due to the reduction of the number of samples and the warping of the posterior distribution due to the change of priori probabilities.

We proposed a theoretical analysis specifying under which conditions undersampling is recommended and expected to be effective. It emerges that undersampling is not always the best solution and several factors affect its power (e.g. variance of the classifier, the degree of imbalance, class separability and the value of the posterior probability.) However, when it appears to improve predictive accuracy we show that it is important to re-calibrate the posterior probability of a classifier to account for the change in class priors. Finally, we propose to use a Racing algorithm to choose between multiple strategies for unbalanced classification in order to avoid testing all possible techniques and configurations.

Chapter 5

Learning from evolving data streams with skewed distributions

Results presented in this chapter have been published in the following papers:

- Andrea Dal Pozzolo, Olivier Caelen, Yann-Ael Le Borgne, Serge Waterschoot, and Gianluca Bontempi. *Learned lessons in credit card fraud detection from a practitioner perspective.* Expert Systems with Applications, 41(10):4915-4928, 2014.
- Andrea Dal Pozzolo, Reid A. Johnson, Olivier Caelen, Serge Waterschoot, Nitesh V Chawla, and Gianluca Bontempi. *Using HDDT to avoid instances propagation in unbalanced and evolving data streams.* In Neural Networks (IJCNN), The 2014 International Joint Conference on. IEEE, 2014.

Fraud Detection problems are typically addressed in two different ways. In the static learning setting, a detection model is periodically retrained from scratch (e.g. once a year or month). In the online learning setting, the detection model is updated as soon as new data arrives. Though this strategy is the most adequate to deal with issues of non-stationarity, little attention has been devoted in the literature to the unbalanced problem in a changing environment. Another problematic issue in credit card fraud detection is the scarcity of available data due to confidentiality issues that give little chance to the community to share real datasets and assess existing techniques.

The first part of the chapter (Section 5.1) is based on [18] and it aims at making an experimental comparison of several state-of-the-art algorithms and modeling techniques on one real dataset, focusing in particular on some open questions like: Which machine learning algorithm should be used? Is it enough to learn a model once a month or it is necessary to update the model everyday? How many transactions are sufficient to train

the model? Should the data be analyzed in their original unbalanced form? If not, which is the best way to rebalance them? Which performance measure is the most adequate to assess results?

We address these questions with the aim of assessing their importance on real data and from a practitioner perspective. These are just some of potential questions that could rise during the design of a detection system. We do not claim to be able to give a definite answer to the problem, but we aim to give some guidelines to other people in the field. Our goal is to show what worked and what did not in a real case study.

Section 5.1 starts by formalizing the credit card fraud detection task and present a way to create new features in the datasets that can trace the cardholder spending habits. Then, we propose and compare three approaches for online learning in order to identify what is important to retain or to forget in a changing and non-stationary environment. We show the impact of the rebalancing technique on the final performance when the class distribution is skewed. In doing this we merge techniques developed for unbalanced static datasets with online learning strategies. Our experimental analysis shows that to adapt to changing environments, it is imperative to update the learning algorithm. A static approach that never updates the learning algorithm often fails to maintain good predictive performances.

The second part of the chapter (Section 5.2) builds upon this result and proposes an algorithm solution for unbalanced data streams based on Hellinger Distance Decision Tree (HDDT) [19], which removes the need of sampling techniques or instances propagation. HDDT [230] has been previously used for static datasets with skewed distributions and it has shown better performance than standard decision trees. In unbalanced data streams, state-of-the-art techniques use instance propagation and decision trees (e.g. C4.5 [102]) to cope with the unbalanced problem. However, it is not always possible to either revisit or store old instances of a data stream. Using HDDT allows us to: i) remove instance propagations between batches and ii) using all information available in a batch without need to rebalance the classes before training a classifier. This has several benefits, for example: i) improved predictive accuracy, ii) speed, and iii) single-pass through the data. We also use a Hellinger weighted ensemble of HDDTs to combat concept drift and increase the accuracy of single classifiers. We test our framework on several streaming datasets with unbalanced classes and concept drift.

5.1 Learning strategies in credit card fraud detection

5.1.1 Formalization of the learning problem

In this section, we formalize the credit card fraud detection task as a statistical learning problem. Each transaction is described by a feature vector x containing basic information such as amount of the expenditure, the shop where it was performed, the currency, etc. However, these variables do not provide any information about the normal card usage. The normal behavior of a cardholder can be measured by using a set of historical transactions from the same card. For example, as previously explained in Section 2.2.2, we can get an idea of the cardholder spending habits by looking at the average amount spent in different merchant categories (e.g. restaurant, online shopping, gas station, etc.) in the last 3 months preceding the transaction.

Let x_{ij} be the transaction number j of a card number i and $dt(x_{ij})$ be the corresponding transaction date-time. We assume that the transactions are ordered in time such that if x_{iv} occurs before x_{iw} then $dt(x_{iv}) < dt(x_{iw})$. Let $x_{i\lambda}$ be a new incoming transaction and Δt denote the time-frame of a set of historical transactions for the same card. $\mathrm{H}_{i\lambda}$ is then the set of the historical transactions occurring in the time-frame Δt before $x_{i\lambda}$ such that $\mathrm{H}_{i\lambda} = \{x_{ij}\}$, where $dt(x_{i\lambda}) > dt(x_{ij}) \geq dt(x_{i\lambda}) - \Delta t$. For instance, with $\Delta t = 90$ days, $\mathrm{H}_{i\lambda}$ is the set of transactions for the same card occurring in the 3 months preceding $dt(x_{i\lambda})$. The card behavior can be summarized using classical aggregation methods (e.g. *mean*, *max*, *min* or *count*) on the set $\mathrm{H}_{i\lambda}$. This means that it is possible to create new *aggregated* variables that can be added to the original variables in x to include information of the card. In this way we have included information about the user behavior at the transaction level. As a consequence, transactions from cardholders with similar spending habits will share analogous aggregate variables. Let $\bar{x}_k$ indicate the feature vector associated to a transaction x_{ij} obtained by adding aggregated features. At day t, a classifier $\mathcal{K}_t$ is trained on a batch of supervised transactions, denoted as $\mathrm{B}_t = \{(\bar{x}_k, y_k), k = 1, \ldots, N\}$, to predict $\mathcal{P}(+|\bar{x}_\lambda)$, the probability of a new incoming transaction $\bar{x}_\lambda$ to be fraudulent. Note that in this formulation we ignore the status of the card. We also assume to know the true class of all transactions, i.e. no mislabeled samples due to undetected frauds nor frauds reported with a delay by the cardholders.

5.1.2 Strategies for learning with unbalanced and evolving data streams

The most conventional way to deal with sequential fraud data is to adopt a *Static* approach (see Figure 5.1 and Algorithm 1), which creates once in a while a classification model and uses it as a predictor during a long horizon. Though this approach reduces the

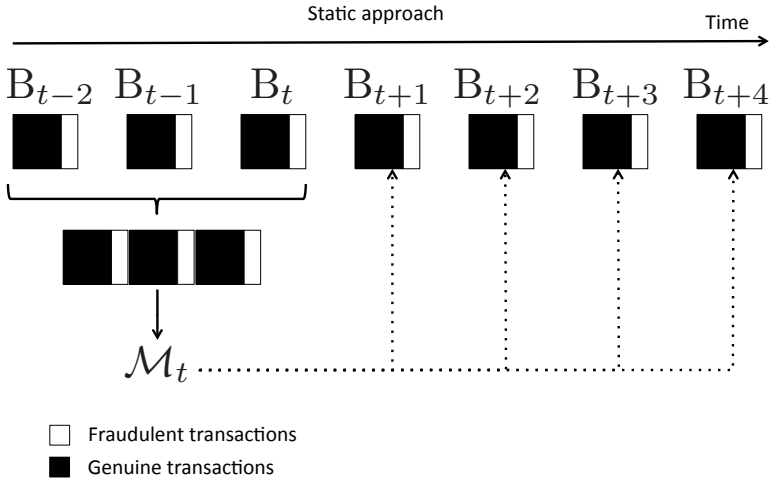


FIGURE 5.1: Static approach: a model $\mathcal{M}_t$ is trained on $K = 3$ batches and used to predict future batches.

Algorithm 1 Static approach

Require: K: number of batches used for training, B: total number of batches available ($B > K$), $\mathcal{K}$: classification algorithms, *sampling*: sampling method.
T $\leftarrow$ empty set of transactions.
for $t \in \{1, \ldots, K\}$ **do**
  $B_t \leftarrow t^{th}$ batch of transactions.
  T $\leftarrow$ merge B_t with T.
$\mathcal{M} \leftarrow$ BUILDMODEL(T, $\mathcal{K}$, *sampling*)
for $t \in \{K+1, \ldots, B\}$ **do**
  $\hat{p} \leftarrow$ use $\mathcal{M}$ to predict $\mathcal{P}(+, \bar{x})$ for transactions in B_t.
  $acc_t \leftarrow$ predictive accuracy of $\hat{p}$ ▷ measured by AUC, AP, P_k.
return *Average*(*acc*)

learning effort, its main problem resides in the lack of adaptivity that makes it insensitive to any change of distribution in the upcoming batches.

On the basis of the state-of-the-work described in Section 3.3, it is possible to conceive two alternative strategies to address both the incremental and the unbalanced nature of the fraud detection problem. The first approach, denoted as the *Update* approach and illustrated in Figure 5.2 and Algorithm 2, is inspired by Wang et al. [189]. It uses a set of M models and a number K of batches to train each model. Note that for $M > 1$ and $K > 1$ the training sets of the M models are overlapping. This approach adapts to changing environment by forgetting batches at a constant rate. The classifier used to make predictions is an ensemble of last M models.

The second approach denoted as the *Propagate and Forget* approach and illustrated in Figure 5.3 and Algorithm 3 is inspired by Gao et all's work [170]. In order to mitigate the unbalanced effects, each time a new batch is available, a model is learned on the genuine transactions of the previous K_{gen} batches and all past fraudulent transactions.

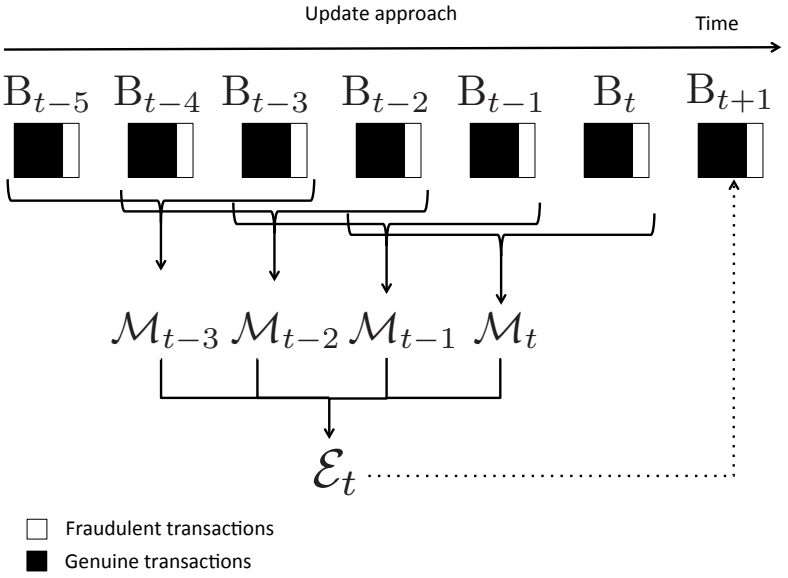


FIGURE 5.2: Updating approach for $K = 3$ and $M = 4$. For each new batch a model is trained on the K latest batches. Single models are used to predict the following batch or can be combined into an ensemble $\mathcal{E}_t$.

Algorithm 2 Update approach

Require: K: number of batches used for training, B: total number of batches available ($B > K$), M: number of models in the ensemble, $\mathcal{K}$: classification algorithms, *sampling*: sampling method.

```
T ← empty set of transactions.
E_t ← empty array of models.
k ← 0
m ← 0
for t ∈ {1, ..., B} do
    B_t ← t^th batch of transactions.
    k ← k + 1.
    T ← merge B_t with T.
    if k ≥ K then
        remove B_{t−K} from T
        k ← k − 1.
    m ← m + 1.
    M_t ← buildModel(T, K, sampling)
    E_t ← add M_t to E_t.
    if m ≥ M then
        remove M_{t−M} from E_t
        m ← m − 1.
    p̂ ← use E_t to predict P(+, x̄) for transactions in B_{t+1}.
    acc_{t+1} ← predictive accuracy of p̂        ▷ measured by AUC, AP, P_k.
return Average(acc)
```

Since this approach leads to training sets which grow in size over the time, a maximum training size is set to avoid overloading. Once this size is reached, older observations are removed in favor of the more recent ones. An ensemble of models is obtained by combining the last M models as in the update approach.

Note that in all these approaches (including the static one), a balancing technique (e.g. undersampling, SMOTE) can be introduced to reduce the skewness of the training set.

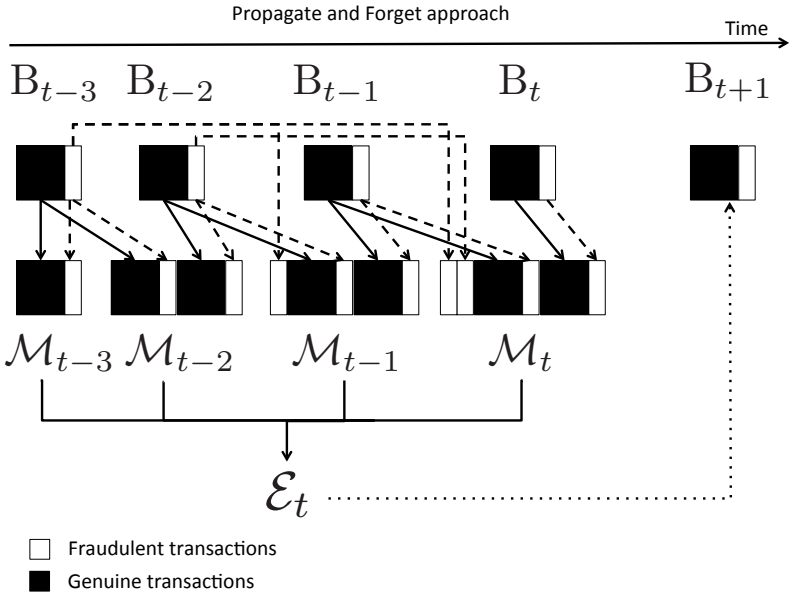


FIGURE 5.3: Propagate and Forget approach: for each new batch a model is created by keeping all previous fraudulent transactions and a small set of genuine transactions from the last 2 batches ($K_{gen} = 2$). Single models are used to predict the following batch or can be combined into an ensemble ($M = 4$).

Algorithm 3 Propagate and Forget approach

Require: K_{gen}: number of batches from which to retain genuine transactions, B: total number of batches available ($B > K$), M: number of models in the ensemble, N: max size of the training set, $\mathcal{K}$: classification algorithms, *sampling*: sampling method.

```
T ← empty set of transactions.
E_t ← empty array of models.
k ← 0
m ← 0
for t ∈ {1, ..., B} do
    B_t ← t^th batch of transactions.
    k ← k + 1.
    T ← merge B_t with T.
    if k ≥ K_gen then
        remove genuine transactions of B_{t−K_gen} from T
        k ← k − 1.
    if |T| > N then
        remove oldest fraudulent transactions from T until |T| = N
    m ← m + 1.
    M_t ← buildModel(T, K, sampling)
    E_t ← add M_t to E_t.
    if m ≥ M then
        remove M_{t−M} from E_t
        m ← m − 1.
    p̂ ← use E_t to predict P(+, x̄) for transactions in B_{t+1}.
    acc_{t+1} ← predictive accuracy of p̂        ▷ measured by AUC, AP, P_k.
return Average(acc)
```

In Table 5.1 we have summarized the strengths and weaknesses of these approaches. The *Static* strategy has the advantage of being fast as the training of the model is done only once, but this does not return a model that follows the changes in the distribution of the

Table 5.1: Strengths and weaknesses of the different learning approaches.

Approach	**Strengths**	**Weaknesses**
Static	• Speed	• No CD adaptation
Update	• No instances propagation • CD adaptation	• Needs several batches for the minority class
Propagate and Forget	• Accumulates minority instances faster • CD adaptation	• Propagation leads to larger training time

data. The other two approaches on the contrary can adapt to Concept Drift (CD). They differ essentially in the way the minority class is accumulated in the training batches. The *Propagate and Forget* strategy propagates instances between batches leading to bigger training sets and computational burden.

5.1.3 Experimental assessment

In this section we perform an extensive experimental assessment on the basis of real data in order to address common issues that the practitioner has to solve when facing large credit card fraud datasets.

Dataset and features transformation

The credit card fraud dataset contains a subset of the transactions from the first of February 2012 to the twentieth of May 2013 (details in Table 5.2). The dataset was divided in daily batches and contained e-commerce fraudulent transactions.

Table 5.2: Fraudulent dataset

# Days	# Features	# Transactions	Period
422	45	2'202'228	1Feb12 - 20May13

This dataset is strongly unbalanced (the percentage of fraudulent transactions is lower than 0.4%) and has a total of 422 batches containing daily transactions with an average of 5218 transactions per batch.

The original variables available in the dataset included the transaction amount, point of sale, currency, country of the transaction, merchant type and many others. However, these variables do not explain cardholder behavior, so aggregated features are added to the original ones in order to profile the user behavior (see Section 5.1.1). For example, the transaction amount and the $CARD\_ID$ is used to compute the average expenditure

per week and per month of one card, the difference between the current and previous transaction and many others. For each transaction and card we took 3 months ($\Delta t =$ 90 days) to compute the aggregated variables.

A great number of the variables in the original dataset are discrete features taking several values. However, some classifiers such as NNET are not able to handle discrete variable, so these features had to be transformed into numerical features. In the R environment [25], discrete variables are also known as factors and their values as levels. One straightforward way to transform factors into numerical feature is to compute the risk of having a fraudulent transaction for each level of a factor. Let's consider for example a factor variable f. We can compute the probability of the j^{th} level of factor f to be fraudulent, denoted as β_j, as follows: $\beta_j = \frac{N^+_{f=j}}{N_{f=j}}$, where $N^+_{f=j}$ is the number of fraudulent transactions and $N_{f=j}$ the total number of transactions for factor f with level j.

As first option we could use β_j to associate to each level a fraud score, but this would lead to the following issues: i) with few fraudulent transactions, many levels are going to have $\beta_j = 0$ making the distribution of the new feature skewed towards zero, and ii) β_j could be the same for levels with big and small frequency. For example if we have a level with 1 fraudulent transaction out of 10 we have $\beta_j = 0.1$ as for a level with 100 fraudulent transactions out of 1000. In the first example however we compute β_j using only 10 observations whereas in the second we can use 1000 observations, therefore in the second scenario we are more confident about the probability calculation.

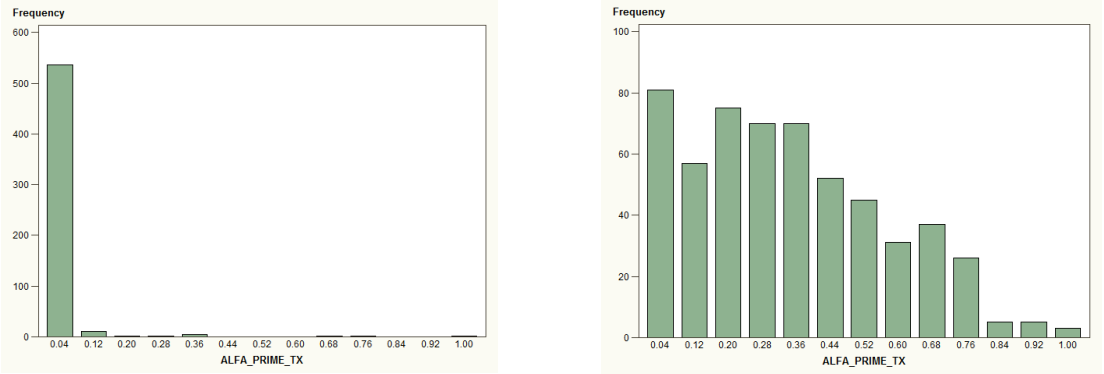
In our work we used a scoring function which combine some a-priori fraud probability with β_j. The idea is that when there's not much information (levels with few transactions) we should use what we called the *Average Fraud Probability* (AFP) that is the mean probability of having a fraud in a day. On the other hand, when we have enough information we want to consider the fraud probability coming from the factor levels (defined by β_j). We used a weighted average of AFP and β_j where the weight α_j given to β_j is defined by the proportion of transactions for that level to all the transactions: $\alpha_j = \frac{N_{f=j}}{N}$. As a result a factor can be transformed into a numerical feature by associating to each level a fraud score S_j as follow:

$$S_j = \alpha_j \beta_j + (1 - \alpha_j)\text{AFP} \tag{5.1}$$

In this scoring function however, even with the maximum value of α_j (which is always less than 1) we still have the influence of the a-priori part given by $(1 - \alpha_j)$AFP. To avoid this problem, we can transform the previous function to bind the weights to be in the range $[0, 1]$ using a new weight $\alpha'_j = \frac{\alpha_j - \min \alpha_j}{\max \alpha_j - \min \alpha_j}$. We can now rewrite (5.1) as:

$$S_j = \alpha'_j \beta_j + (1 - \alpha'_j)\text{AFP} \tag{5.2}$$

A new problem with this new type of scoring function emerges when there is a factor that does not have a uniform distribution of the transactions over its levels. For example the factor variable *TERM_MCC* (terminal Merchant Category Code) has few frequent levels and many rare levels. With this factor we would end up with few levels having a big weight α_j and many levels with small α_j (Figure 5.4(a)). In this kind of situation we can apply the log function to have a smoother α'_j (Figure 5.4(b)).



(a) $\alpha'_j = \frac{\alpha_j - \min \alpha_j}{\max \alpha_j - \min \alpha_j}$ (b) $\alpha'_j = \frac{\log(\alpha_j - \min \alpha_j)}{\log(\max \alpha_j - \min \alpha_i)}$

FIGURE 5.4: Weight α'_j for variable *TERM_MCC*.

Solved the factors problem, the dataset now contains only numerical variables, where the original discrete variables have been transformed into numerical features ranging from 0 (low risk) to 1 (high risk).

Results

Our experimental analysis allows one to compare several strategies for credit card fraud detection. Based on the approaches presented in Section 5.1.2, our experimental findings aim to give guidelines to practitioners working on credit card fraud detection. To evaluate our results, we measure the performance of the detection in terms of AUC, AP and P_k (see Sections 2.2.3).

Influence of algorithm and training set size in a static approach

The static approach (described in Section 5.1.2) is one of the most commonly used by practitioners because of its simplicity and rapidity. However, open questions remain about which learning algorithm should be used and the consequent sensitivity of the accuracy to the training size. We tested three different supervised algorithms: RF,

NNET and SVM provided by the R software [25]. We used R version 3.0.1 with packages `randomForest` [222], `e1071` [228], `unbalanced` [24] and `MASS` [231][1].

In order to assess the impact of the training set size (in terms of days/batches) we carried out the predictions with different windows (K = 30, 60 and 90). All training sets were rebalanced using undersampling and all experiments are replicated five times to reduce the variance caused by the sampling implicit in unbalanced techniques. Figure 5.5 shows the sum of the ranks from the Friedman test [232] for each strategy in terms of AP, AUC and P_k. For each batch, we rank the strategies from the least to the best performing. Then we sum the ranks over all batches. More formally, let $r_{s,b} \in \{1, ..., S\}$ be the rank of strategy s on batch b and S be the number of strategies to compare. The strategy with the highest accuracy in b has $r_{s,b} = S$ and the one with the lowest has $r_{s,b} = 1$. Then the sum of ranks for the strategy s is defined as $\sum_{b=1}^{B} r_{s,b}$, where B is the total number of batches. The higher the sum, the higher is the number of times that one strategy is superior to the others. The white bars denote models which are significantly worse than the best (paired t-test based on the ranks of each batch).

The strategy names follow a structure built on the following options:

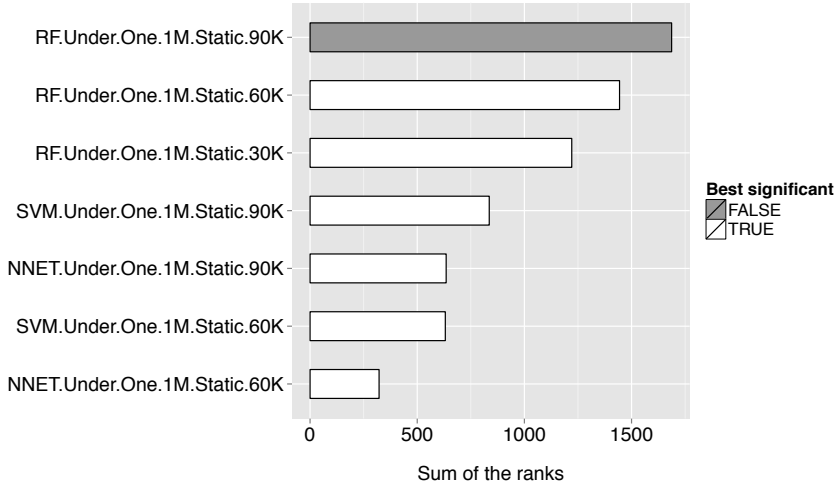
- Algorithm used (RF, SVM, NNET)
- Sampling method (Under, SMOTE, EasyEnsemble)
- Model update frequency (One, Daily, 15days, Weekly)
- Number of models in the ensemble (M)
- Learning approach (Static, Update, Propagate and Forget)
- Learning parameter (K, K_{gen})

Then the strategy options are concatenated using the dot as separation point (e.g. RF.Under.Daily.10M.Update.60K). In both datasets, Random Forests clearly outperforms its competitors and, as expected, accuracy is improved by increasing the training size (Figure 5.5). Because of the significative superiority of Random Forests with respect to the other algorithms, in what follows we will limit to consider only this learning algorithm.

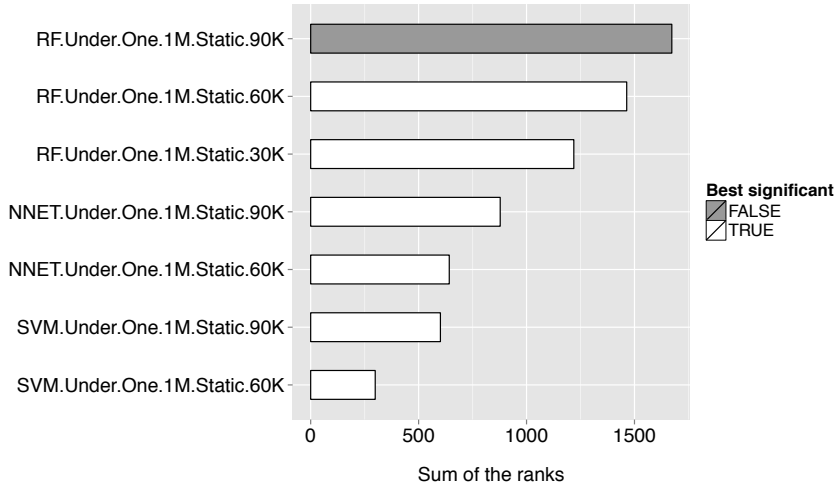
Advantage of updating models

Here we assess the advantage of adopting the *update* approach described in Section 5.1.2. Figure 5.6 reports the results for different values of K and M. The strategies are called

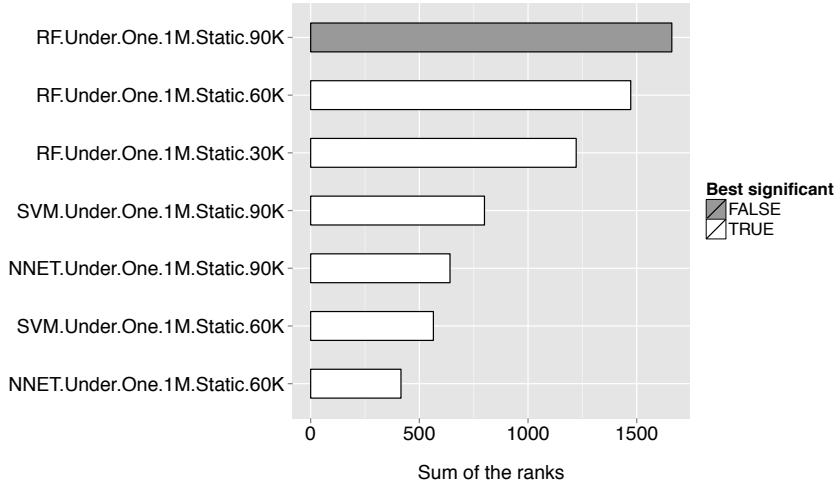
[1]For each classifier we used the default settings provided by the package.



(a) Metric: AP



(b) Metric: AUC



(c) Metric: P_k

FIGURE 5.5: Comparison of static strategies using sum of ranks in all batches.

daily if a model is built every day, *weekly* if once a week or *15days* if every 15 days. We compare ensemble strategies ($M > 1$) with models built on single batches ($K = 1$) against single models strategies ($M = 1$) using several batches in the training ($K > 1$).

For all metrics, the best strategy is *RF.Under.Daily.5M.Update.90K*. It creates a new model at each batch using previous 90 days ($K = 90$) for training and keeps the last 5 models created ($M = 5$) for predictions. In the case of AP however this strategy is not statistically better than the ensemble approaches ranked as second.

For all metrics, the strategies that use only the current batch to build a model ($K = 1$) are coherently the worst. This confirms the result of previous analysis showing that a too short window of data (and consequently a very small fraction of frauds) is insufficient to learn a reliable model. When comparing the update frequency of the models using the same number of batches for training ($K = 90$), *daily* update is ranking always better than *weekly* and *15days*. This confirms the intuition that the fraud distribution is always evolving and therefore it is better to update the models as soon as possible.

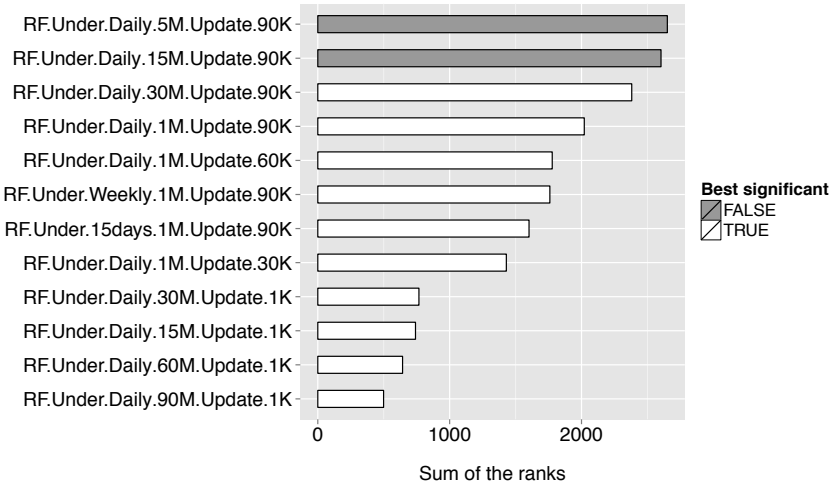
Benefit of propagating old frauds

This section assesses the accuracy of the *Propagate and Forget* approach described in Section 5.1.2 whose rationale is to avoid discarding old fraudulent observations.

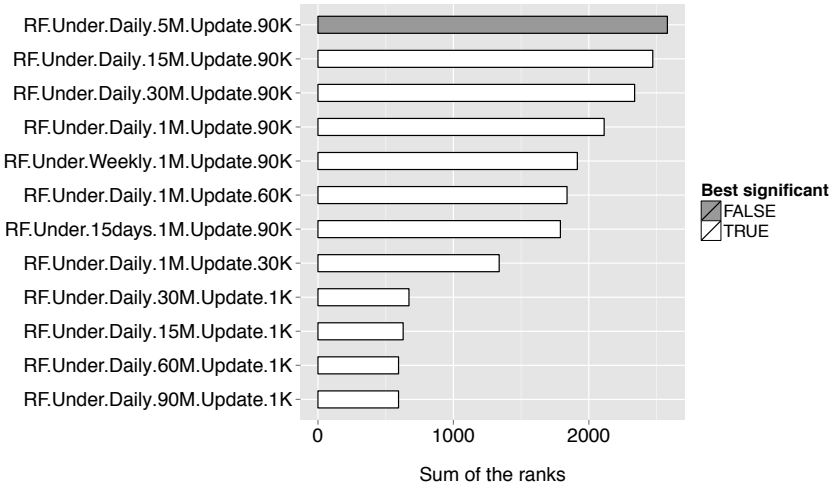
Accumulating old frauds leads to less unbalanced batches. In order to avoid having batches where the accumulated frauds outnumber the genuine transactions, two options are available: i) forgetting some of the old frauds ii) accumulating old genuine transactions as well. In the first case when the accumulated frauds represent 40% of the transactions, new frauds replace old frauds as in Gao [171]. In the second case we accumulate genuine transactions from previous K_{gen} batches, where K_{gen} defines the number of batches used (see Figure 5.3).

Figure 5.7 shows the sum of ranks for different strategies where the genuine transactions are taken from a different number of days (K_{gen}). The best strategy according to AP and P_k uses an ensemble of 5 models for each batch ($M = 5$) and 30 days for genuine transactions ($K_{gen} = 30$). The same strategy ranks third in terms of AUC and is significantly worse than the best. To create ensembles we use a time-based array of models of fixed size M, which means that when the number of models available is greater than M, the most recent in time model replaces the M^{th} model in the array removing the oldest model in the ensemble.

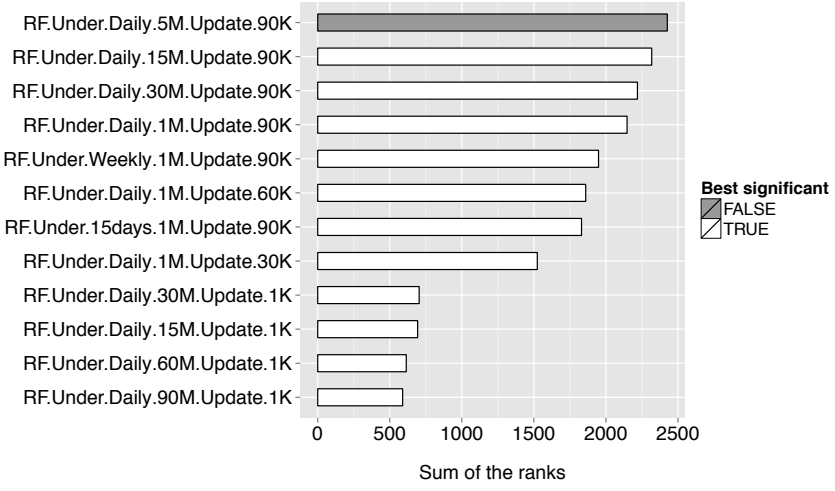
In general we see better performances when K_{gen} increases from 0 to 30 and only in few cases $K_{gen} > 30$ leads to significantly better accuracy. Note that in all our strategies



(a) Metric: AP



(b) Metric: AUC



(c) Metric: P_k

FIGURE 5.6: Comparison of update strategies using sum of ranks in all batches.

after selecting the observations to include in the training sets we use undersampling to make sure we have the two classes equally represented.

Impact of balancing techniques on detection accuracy

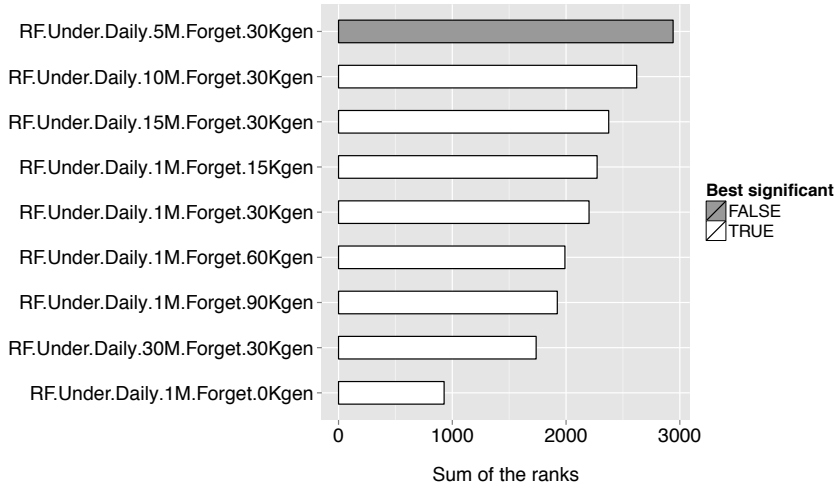
So far we considered exclusively undersampling as balancing technique in our experiments. In this section we assess the impact of using alternative methods like SMOTE and EasyEnsemble. Experimental results (Figure 5.8) show that they both over-perform undersampling.

In our datasets, the number of frauds is on average 0.4% of all transactions in the batch. Undersampling randomly selects a number of genuine transactions equal to the number of frauds, which means removing about 99.6% of the genuine transactions in the batch. EasyEnsemble is able to reduce the variance of undersampling by using several sub-models for each batch, while SMOTE creates new artificial fraudulent transactions. In our experiments we used 5 sub-models in EasyEnsemble. For all balancing techniques, between the three approaches presented in Section 5.1.2, the *static* approach is consistently the worse.

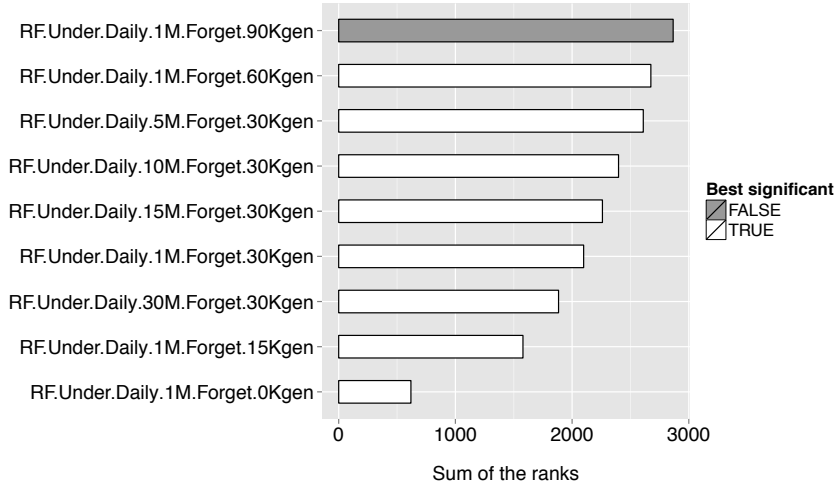
In Figure 5.9 we compare the previous strategies in terms of average prediction time over all batches. SMOTE is computationally heavy since it consists in oversampling, leading to bigger batch sizes. EasyEnsemble replicates undersampling and learns from several sub-batches. This gives higher computational time than undersampling. Between the different incremental approaches, static has the lowest time as the model is learned once and not retrained. Propagate and Forget strategy has the highest prediction time over all balancing methods. This is expected since it retains old transactions to deal with unbalanced batches.

Analysis of the best overall strategy

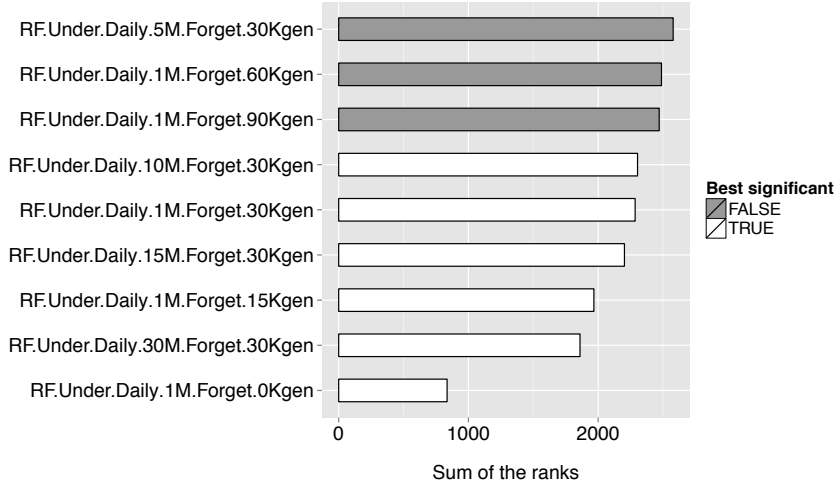
The large number of possible alternatives (in terms of learning classifier, balancing technique and incremental learning strategy) requires a joint assessment of several combinations in order to come up with a recommended approach. Figure 5.10 summaries the best strategies in terms of different metrics. The combinations of EasyEnsemble with *Propagate and Forget* emerge as best for all metrics. SMOTE with update is not significantly worse of the best for AP and P_k, but it is not ranking well in terms of AUC. The fact that within the best strategies we see different balancing techniques confirms that in unbalanced data streams, the adopted balancing strategy may play a major role. As expected the static approach ranks low in Figures 5.10 as it is not able to adapt to the



(a) Metric: AP

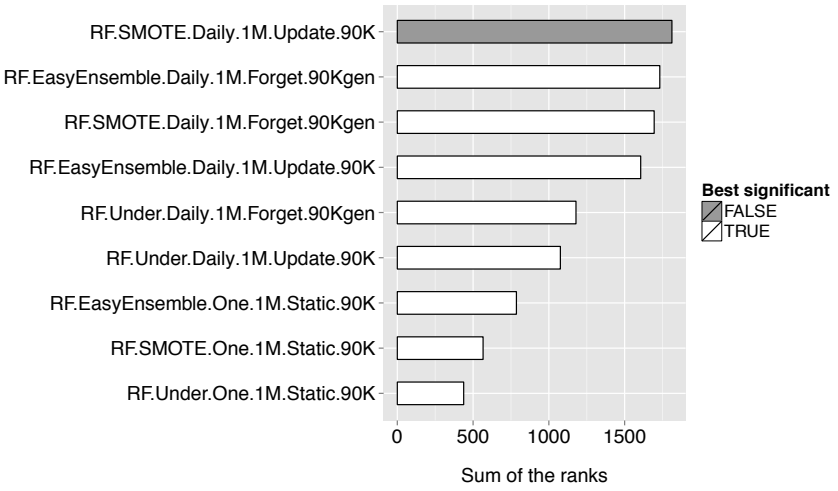


(b) Metric: AUC

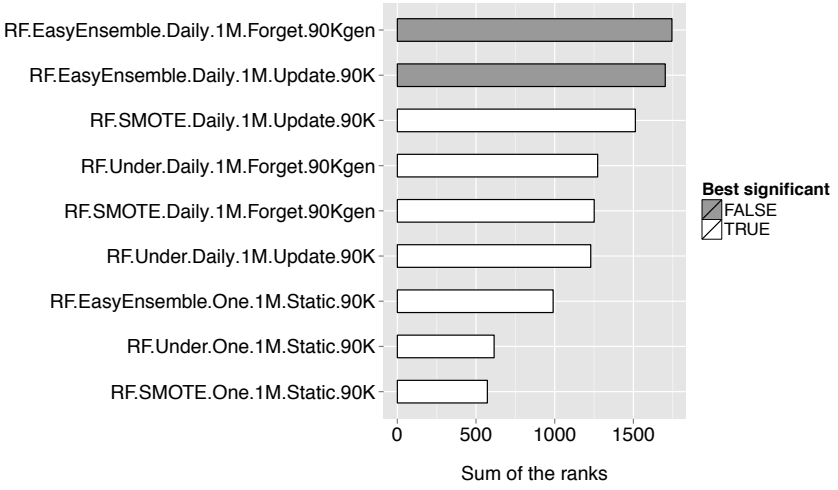


(c) Metric: P_k

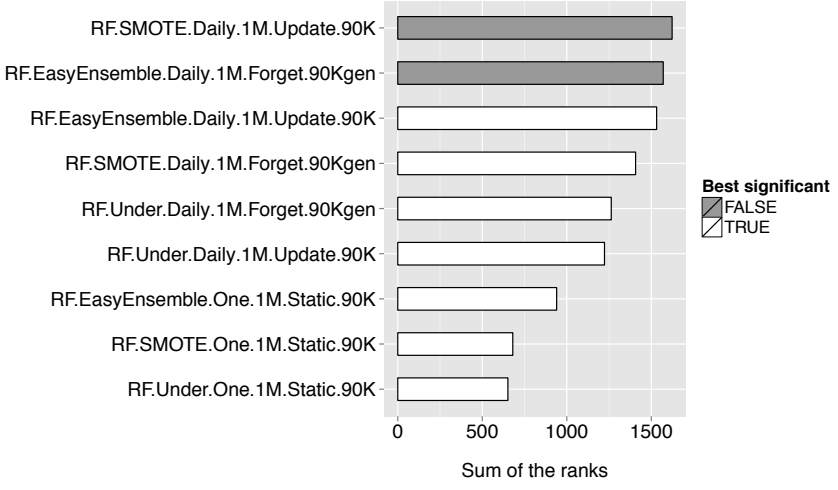
FIGURE 5.7: Comparison of forgetting strategies using sum of ranks in all batches.



(a) Metric: AP



(b) Metric: AUC



(c) Metric: P_k

FIGURE 5.8: Comparison of different balancing techniques and strategies using sum of ranks in all batches.

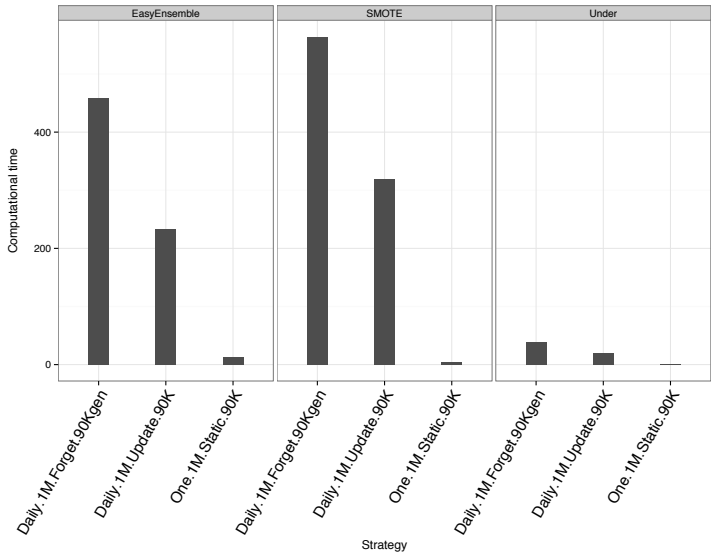


FIGURE 5.9: Comparison of different balancing techniques and strategies in terms of average prediction time (in seconds) over all batches. Experiments run on a HP ProLiant BL465c G5 blades with 2x AMD Opteron 2.4 GHz, 4 cores each and 32 GB DDR3 RAM.

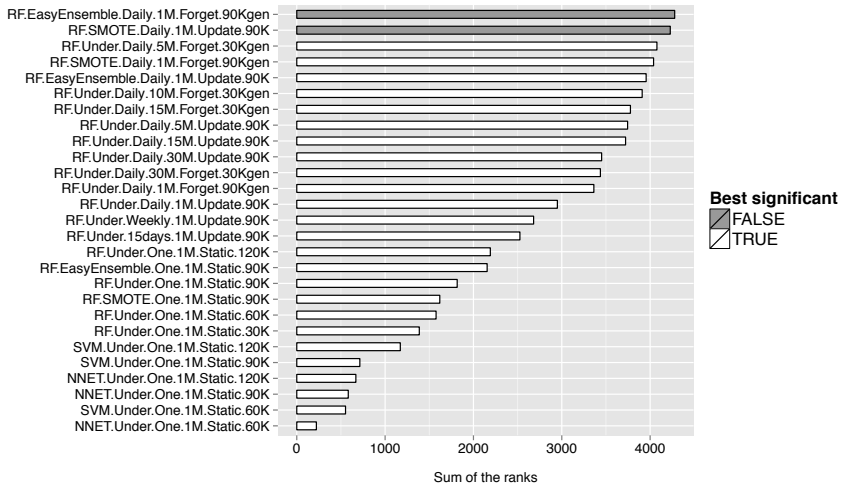
changing distribution. The *Propagate and Forget* approach is significantly better than *Update* for EasyEnsemble, while SMOTE gives better ranking with update.

It is worth to note that strategies which combine more than one model ($M > 1$) together with undersampling are not superior to the predictions with a single model and EasyEnsemble. EasyEnsemble learns from different samples of the majority class, which means that for each batch different concepts of the majority class are learned.

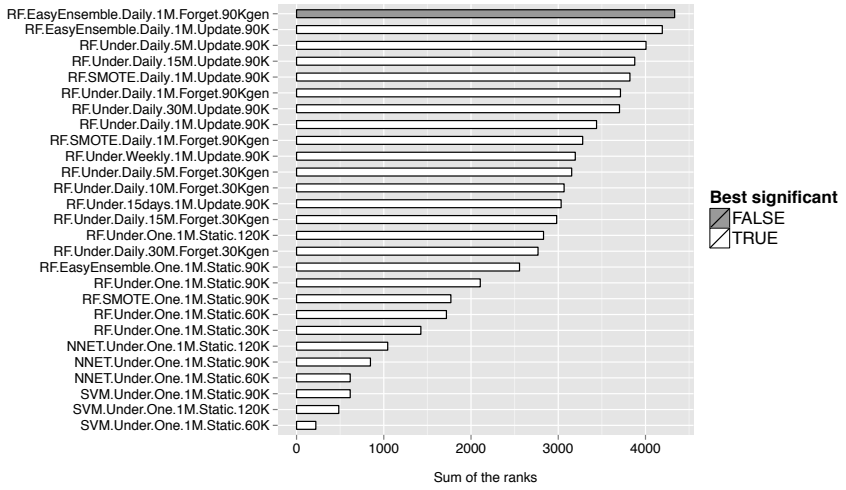
5.1.4 Discussion

The need to detect fraudulent patterns in huge amounts of data demands the adoption of automatic methods. The scarcity of public available dataset in credit card transactions gives little chance to the community to test and assess the impact of existing techniques on real data. The goal of this first part of the chapter is to give some guidelines to practitioners on how to tackle the detection problem.

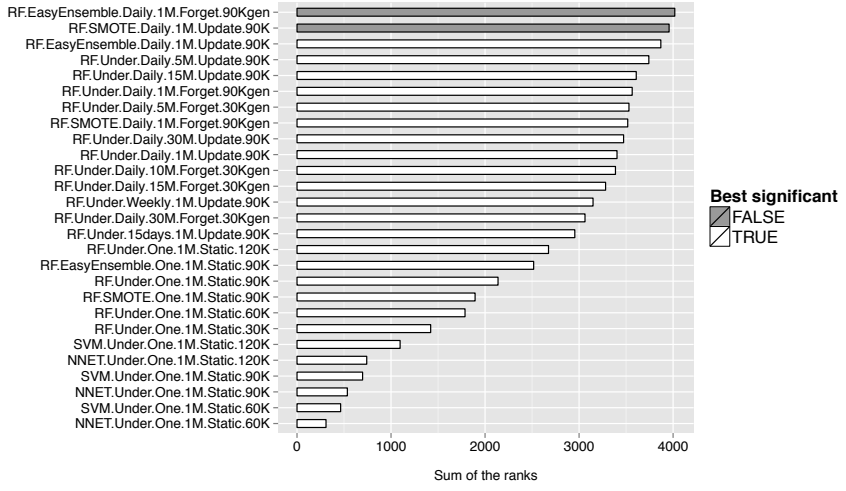
Credit card fraud detection relies on the analysis of recorded transactions. However, single transaction information is not considered sufficient to detect a fraud occurrence [11] and the analysis has to take into consideration the cardholder behavior. To this purpose we include cardholder information into the transaction by computing aggregated variables on historical transactions of the same card. As new credit-card transactions keep arriving, the detection system has to process them as soon as they arrive and avoid retaining in memory too many old transactions. We compare three alternative approaches



(a) Metric: AP



(b) Metric: AUC



(c) Metric: P_k

FIGURE 5.10: Comparison of all strategies using sum of ranks in all batches.

(*Static*, *Update* and *Propagate and Forget*) to study which approach is better to adopt in unbalanced and non-stationary credit card data streams.

In the static learning setting a wide range of techniques have been proposed to deal with unbalanced datasets. However, in online/incremental learning few attempts have tried to deal with unbalanced data streams [52, 148, 170]. In these works, the most common balancing technique consists in undersampling the majority class in order to reduce class imbalance. In our work we adopted two alternatives to undersampling: SMOTE and EasyEnsemble. In particular we show that they are both able to return higher accuracies. Our framework can be easily extended to include other data-level balancing techniques.

The experimental part has shown that in data streams, with standard classification algorithms, when the distribution is skewed towards one class it is important maintaining previous minority examples in order to learn a better separation of two classes. Instance propagation from previous batches has the effect of increasing the minority class in the current batch, but it is of limited impact given the small number of frauds. We tested several ensembles and single models strategies using different number of batches for training. In general we see that models trained on multiple batches have better accuracy than single batch models. Multi-batches models learn on overlapping training sets, when this happens single models strategies can beat ensembles.

Our framework addresses the problem of non-stationarity in data streams by creating a new model every time a new batch is available. This approach has showed better results than updating the models at a lower frequency (*weekly* or every *15days*). Updating the models is crucial in a non-stationary environment, this intuition is confirmed by the bad results of the static approach. In our dataset, overall we saw Random Forest beating Neural Network and Support Vector Machine. The final best strategy implemented the *Propagate and Forget* approach together with EasyEnsemble and daily update.

5.2 Using HDDT to avoid instances propagation

The results from the previous sections have shown that, in fraud detection, it is important to update the model as soon as new data arrives in order to adapt to possible changes in the data distributions. In this second part of the chapter we aim to improve the update approach presented in Section 5.1.2 by proposing Hellinger Distance Decision Tree (HDDT) [103] as a base learner for data streams.

Several state-of-the-art techniques for unbalanced data streams have addressed the imbalance problem by propagating minority observations between batches, with C4.5 decision

trees being the most common algorithm used [148, 171, 175, 176]. These techniques retain previous minority class instances in order to provide the algorithm balanced batches, since the classifier is not able to learn from skewed distributions. However, when data arrive as a continuous stream of transactions, it can become impossible to store or propagate previous observations. Therefore, there is the need of tools that are able to process streams of data as soon as they arrive, without revising old observations.

This section is based on our previous work [19], where we used HDDT [103] as a base classifier for data streams in order to avoid instance propagation between subsequent batches of data. We show that HDDT is able to produce superior performances than standard C4.5 decision trees in terms of predictive accuracy, computational time and resources needed.

In order to combat concept drift we have used a batch-ensemble model combination based on Hellinger Distance and Information Gain as in [175]. This choice has proved to be beneficial in the presence of changing distributions in the data. We have tested our framework with different types of datasets: unbalanced datasets without known concept drift, artificial datasets with known concept drift and a highly unbalanced credit card fraud dataset with concept drift.

5.2.1 Hellinger Distance Decision Trees

Originally introduced to quantify the similarity between two probability distributions [233], the Hellinger Distance (HD) has been recently proposed as a splitting criterion in decision trees to improve the accuracy in unbalanced problems [103, 230]. In the context of data streams, it has produced excellent results in detecting classifier performance degradation due to concept drift [175, 234]. Let P_1 and P_2 be two discrete probability distributions taking values $\phi \in \Phi$, Hellinger Distance is defined as:

$$HD(P_1, P_2) = \sqrt{\sum_{\phi \in \Phi} \left(\sqrt{P_1(\phi)} - \sqrt{P_2(\phi)} \right)^2} \tag{5.3}$$

Hellinger Distance has several properties:

- $HD(P_1, P_2) = HD(P_2, P_1)$ (symmetric)
- $HD(P_1, P_2) >= 0$ (non-negative)
- $HD(P_1, P_2) \in [0, \sqrt{2}]$

Starting from (5.3), Cieslak and Chawla [103] derive a new decision tree splitting criteria based on Hellinger Distance that is skew insensitive. They start from the assumption

that all numerical features are partitioned into q bins, so that the resulting dataset is made of only categorical variables. Then for each feature f, they compute the distance between the classes over all the feature's partitions. In the case of a binary classification problem where f^+ denotes the instances of the positive class and f^- the negatives, the Hellinger distance between f^+ and f^- is:

$$HD(f^+, f^-) = \sqrt{\sum_{j=1}^{q} \left(\sqrt{\frac{|f_j^+|}{|f^+|}} - \sqrt{\frac{|f_j^-|}{|f^-|}} \right)^2} \tag{5.4}$$

where j defines the j^{th} bin of feature f and $|f_j^+|$ is the number of positive instances of feature f in the j^{th} bin. At each node of the tree, $HD(f^+, f^-)$ is computed for each feature and then the feature with the maximum distance is used to split. The authors of [55, 103, 235] recommend to leave the tree unpruned and to use Laplace smoothing for obtaining posterior probabilities.[2] Note that the class priors do not appear explicitly in equation (5.4), which means that class imbalance ratio does not influence the distance calculation.

5.2.2 Hellinger Distance as weighting ensemble strategy

In evolving data streams it is important to understand how similar two consecutive data batches are in order to decide whether a model learned on a previous batch is still valid. Lichtenwalter and Chawla [175] propose to employ HD as a measure of the distance between two separate batches. Let us define as B_t the batch at time t used for training and B_{t+1} as the subsequent testing batch. First numeric features are discretized into equal-width bins, then Hellinger distance between B_t and B_{t+1} for a given feature f is calculated as:

$$HD(\mathrm{B}_t^f, \mathrm{B}_{t+1}^f) = \sqrt{\sum_{v \in f} \left(\sqrt{\frac{|\mathrm{B}_t^{f=v}|}{|\mathrm{B}_t|}} - \sqrt{\frac{|\mathrm{B}_{t+1}^{f=v}|}{|\mathrm{B}_{t+1}|}} \right)^2} \tag{5.5}$$

where $|\mathrm{B}_t^{f=v}|$ is the number of instances of feature f taking value v in the batch at time t, while $|\mathrm{B}_t|$ is the total number of instances in the same batch.

Equation (5.5) does not account for differences in feature relevance. In general, feature distance should have a higher weight when the feature is relevant, while a small weight should be assigned to a weak feature. Making the assumption that feature relevance remains stable over time, Lichtenwalter and Chawla [175] suggest using the information gain to weight the distances. For a given feature f of a batch B, the Information Gain

[2] When the class distribution is unbalanced, pruning would remove those leaves with few samples which are also the ones more likely to be associated with the minority class.

is defined as the decrease in entropy H of a class c:

$$IG(\mathrm{B}, f) = H(\mathrm{B}^c) - H(\mathrm{B}^c|\mathrm{B}^f) \tag{5.6}$$

where B^c defines the class of the observations in batch B and B^f the observations of feature f. For the testing batch we cannot compute $IG(\mathrm{B}, f)$, as the labels are not provided, therefore the feature relevance is calculated on the training batch. We can now define a new distance function that combines Information Gain and Hellinger Distance as:

$$HDIG(\mathrm{B}_t, \mathrm{B}_{t+1}, f) = HD(\mathrm{B}_t^f, \mathrm{B}_{t+1}^f) * (1 + IG(\mathrm{B}_t, f)) \tag{5.7}$$

$HDIG(\mathrm{B}_t, \mathrm{B}_{t+1}, f)$ provides a relevance-weighted distance for each single feature. The final distance between two batches is then computed by taking the average over all the features.

$$AHDIG(\mathrm{B}_t, \mathrm{B}_{t+1}) = \frac{\sum_{f \in \mathrm{B}_t} HDIG(\mathrm{B}_t, \mathrm{B}_{t+1}, f)}{|f \in \mathrm{B}_t|} \tag{5.8}$$

We learn a new model as soon as a new batch is available and combine learned models into an ensemble where the weights of the models are inversely proportional to the batches' distances. The lower the distance between two batches the more similar is the concept between them. In a streaming environment with concept drift we should expect good performances on the current batch from models learned on similar concepts. With this reasoning in mind, the ensemble weights should be higher for smaller distances. We used the same transformation of [175], where the weight w_t of a classifier trained on B_t and that predicts on B_{t+1} is defined as follows:

$$w_t = AHDIG(\mathrm{B}_t, \mathrm{B}_{t+1})^{-M} \tag{5.9}$$

where M represents the ensemble size.

5.2.3 Experimental assessment

Many of the data streaming frameworks for concept drift and unbalanced data use C4.5 [102] decision tree as the base learner [148, 171, 175, 236]. In our experiments we compare the results of C4.5 to the Hellinger Distance Decision Tree (HDDT) with the parameters suggested in [103] (unpruned and with Laplace smoothing). The comparison is done using different propagation/sampling methods and model combinations (ensemble vs. single models).

In an unbalanced data stream, for each batch/chunk, the positive class examples represent the minority of the observations. Each batch can be considered as a small unbalanced dataset, permitting all the techniques already developed for static unbalanced datasets

to be implemented. In a streaming environment, however, it is possible to collect minority observations from previous batches to combat class imbalance. For our experiments we considered instance propagation methods that assume no sub-concepts within the minority class. In particular we used Gao's [171] and Lichtenwalter's [175] propagation methods presented in Section 3.3 and two other benchmark methods (UNDER and BL):

- SE (Gao's [171] propagation of rare class instances and undersampling at 40%)
- BD (Lichtenwalter's Boundary Definition [175]: propagating rare-class instances and instances in the negative class that the current model misclassifies.)
- UNDER (Undersampling: no propagation between batches, undersampling at 40%)
- BL (Baseline: no propagation, no sampling)

The first two methods can be considered as oversampling methods since the minority proportion in the batches is augmented. From now on, for simplicity we will call all the previously discussed instance propagation methods *sampling* strategies.

For each of the previous sampling strategies we tested: i) HDIG: ensemble with weights from (5.9) and ii) No ensemble: single classifier. In the first case, an ensemble is built combining all models learned with weights given by (5.9). In the second case, we use the model learned in the current batch to predict the incoming batch. This option has the advantage of being faster as no models are stored during the learning phase.

In all our experiments we reported the results in terms of AUC. The framework was written in Java and we used the Weka [237] implementation of C4.5 and HDDT. We used UCI datasets [1] to first study the unbalanced problem without worrying about concept drift. These datasets are not inherently sequential and exhibit no concept drift; we render them as data streams by randomizing the order of instances and processing them in batches as in [175]. Then we used the MOA [238] framework to generate some artificial datasets with drifting features to test the behavior of the algorithms under concept drift. Finally we used a real-world credit card dataset which is highly unbalanced and whose frauds are changing in type and distribution. This dataset contains credit card transactions from online payment between the 5^{th} of September 2013 and the 25^{th} of September 2013, where only 0.15% of the transactions are fraudulent.

We first tested the different sampling strategies using HDDT and C4.5. On the left side of Figure 5.11 we see the results where $HDIG$ distance (discussed in Section 5.2.2) is used to weight the models from different batches according to (5.9). On the right are the results where only the model of the current batch is used for prediction. The columns indicate the batch mean AUC for each strategy averaged over all UCI datasets.

Table 5.3: Datasets

Dataset	Source	# Instances	# Features	Imbalance Ratio
Adult	UCI	48,842	14	3.2:1
can	UCI	443,872	9	52.1:1
compustat	UCI	13,657	20	27.5:1
covtype	UCI	38,500	10	13.0:1
football	UCI	4,288	13	1.7:1
ozone-8h	UCI	2,534	72	14.8:1
wrds	UCI	99,200	41	1.0:1
text	UCI	11,162	11465	14.7:1
DriftedLED	MOA	1,000,000	25	9.0:1
DriftedRBF	MOA	1,000,000	11	1.02:1
DriftedWave	MOA	1,000,000	41	2.02:1
Creditcard	FRAUD	3,143,423	36	658.8:1

This means that for each dataset we computed the mean AUC over all batches and then averaged the results between all datasets. In general we notice that HDDT is able to

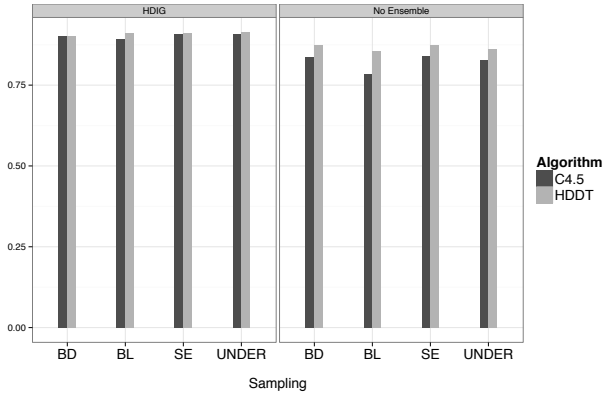


Figure 5.11: Batch average results in terms of AUC (higher is better) using different sampling strategies and batch-ensemble weighting methods with C4.5 and HDDT over all UCI datasets.

outperform C4.5. For each sampling method we see that the ensembles counterpart of the single models have higher accuracy.

In Figure 5.12 we display the average computational time. As expected, when a single classifier is used the framework is much faster, but it comes at the cost of lower accuracy (see Figure 5.11). When undersampling is used in the framework we have the smallest computational time, as it uses a subset of the observations in each batch and no instances are propagated between batches.

Figure 5.13 shows the results for the datasets with concept drift generated using the MOA framework. HDIG-based ensembles return better results than a single classifier and HDDT again gives better accuracy than C4.5.

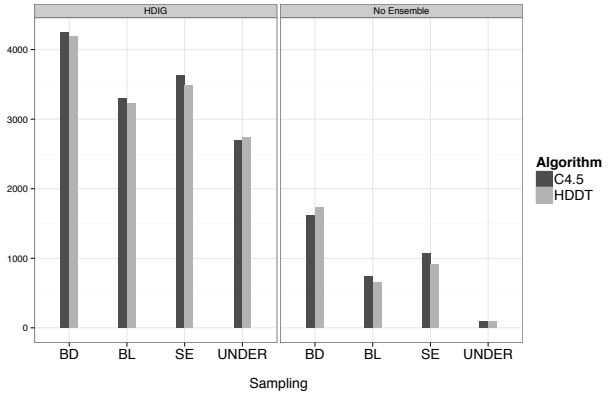


FIGURE 5.12: Batch average results in terms of computational time (lower is better) using different sampling strategies and batch-ensemble weighting methods with C4.5 and HDDT over all UCI datasets.

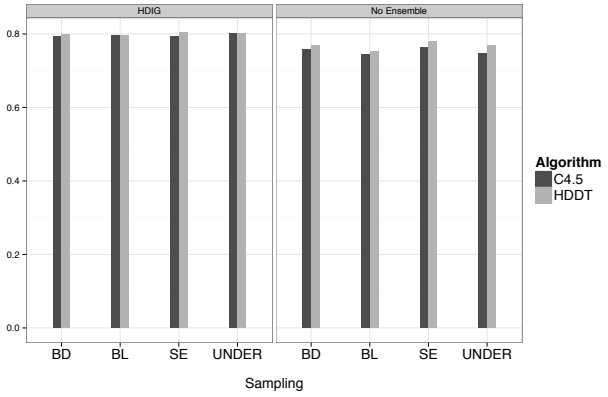


FIGURE 5.13: Batch average results in terms of computational AUC (higher is better) using different sampling strategies and batch-ensemble weighting methods with C4.5 and HDDT over all drifting MOA datasets.

Figure 5.14 displays the results on the Creditcard dataset. This dataset is a good example of an unbalanced data stream with concept drift. Once again HDDT is always better than C4.5, however the increase in performance given by the ensemble is less important than the one registered with the UCI datasets. From Figure 5.14, it is hard to discriminate the best strategy, as many of them have comparable results.

Figure 5.15 shows the sum of the ranks for each strategy over all the batches. As explained in Section 5.1.3, for each batch, we assign the highest rank to the most accurate strategy and then sum the ranks over all batches. The higher the sum, the higher the number of times one strategy is superior to the others.

The strategy with the highest sum of ranks (BL_HDIG_HDDT) combines BL with HDIG ensembles of HDDTs. BL method leaves the batches unbalanced, which means that the best strategy is actually the one avoiding instance propagation/sampling. A

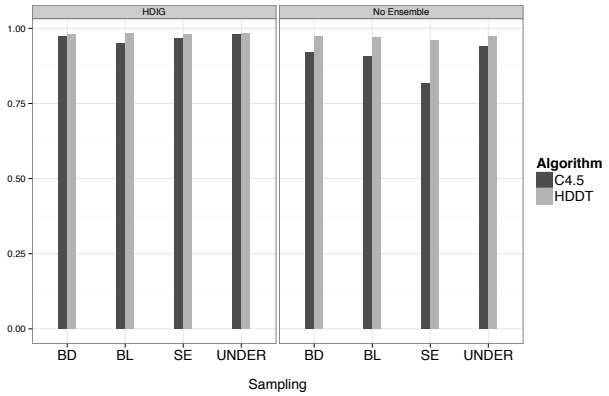


FIGURE 5.14: Batch average results in terms of AUC (higher is better) using different sampling strategies and batch-ensemble weighting methods with C4.5 and HDDT over the Credit Card dataset.

paired t-test on the ranks was then used to compare each strategy with the best. Based on this test, we saw that the strategy with the second-highest sum of ranks (*UNDER_HDIG_HDDT*) is not significantly worse than the first. Compared to the first, this strategy implements undersampling at each batch instead of BL. Figure 5.15 confirms that HDDT is better than C4.5. The C4.5 implementation of the winning strategy (*BL_HDIG_C4.5*) is significantly worse than the best (*BL_HDIG_HDDT*). The same happens for the second best ranking strategy (*UNDER_HDIG_HDDT* ranks higher than *UNDER_HDIG_C4.5*).

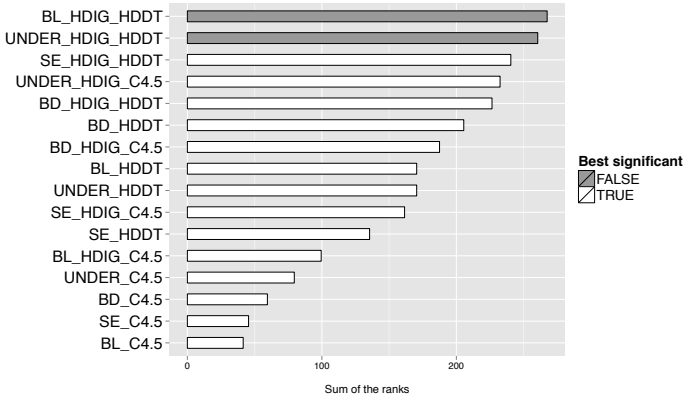


FIGURE 5.15: Comparison of different strategies using the sum of ranks in all batches for the Credit Card dataset in terms of AUC. In gray are the strategies that are not significantly worse than the best having the highest sum of ranks.

5.2.4 Discussion

To our knowledge, our work was the first to evaluate the use of the HDDT tree algorithm for streaming data. Many of the state-of-the-art techniques use the C4.5 algorithm

combined with sampling or instance propagation to balance the batches before training. We have shown that when used in data streams, HDDT without sampling typically leads to better results than C4.5 with sampling. Thus, HDDT can offer better performance than C4.5, while actually removing sampling from the process.

The removal of the propagation/sampling step in the learning process has several benefits:

- It allows a single-pass approach (the observations are processed as soon as they arrive, avoiding several passes throughout the batches for instance propagation).
- It reduces the computational cost/resources needed (this is important since with massive amounts of data it may no longer be possible to store/retrieve old instances).
- It avoids the problem of finding previous minority instances from the same concept (in the case of a new concept in the minority class, it may not be possible to find previous observations to propagate).

We have used artificial datasets to test how different strategies work under concept drift. The use of HDIG as an ensemble weighting strategy has increased the performances of the single classifiers, not only in artificial datasets with known drift (MOA datasets), but even in datasets whose distribution is assumed to be more or less stable (UCI datasets).

Finally, we tested our framework on a proprietary dataset containing credit card transactions from online payment. This is a particularly interesting dataset, as it is extremely unbalanced and exhibits concept drift within the minority class. HDDT performs very well when combined with BL (no sampling) and undersampling. An important feature of these basic sampling strategies is the fact that frameworks implementing them are much faster (see Figure 5.12) since no observations are stored from previous batches. When these two sampling strategies give comparable results, the practitioner could prefer undersampling as it is more memory efficient since it uses a reduced part of the batch for training. By using undersampling, however, a lot of instances from the majority class are not considered.

5.3 Conclusion

In the first part of the chapter (Section 5.1) we have first formalized the problem of fraud detection and then presented three different approaches for unbalanced data streams. We have compared a static approach against two dynamic ways to learn in the presence of non-stationarity and unbalanced distribution. From our experimental assessment the

static approach is consistently the worst in terms of predictive accuracy, a strong signal that the data stream is evolving and updating the detection strategy is beneficial. From the experiments it emerges also that it is important to use strategies to rebalance the batches of the stream before training a classification algorithm, either by resampling (e.g. undersampling, SMOTE) or by propagating instances from the minority class along the stream.

However, it is not always possible to store or revisit previous transactions in high frequency data streams. For this reason, in the second part of the chapter (Section 5.2) we have presented a classification algorithm (HDDT) optimized for unbalanced datasets and used it in a data stream environment in order to avoid the propagation of instances between batches. HDDT showed good performances without the need of rebalancing the two classes (e.g. no undersampling required) and proved to perform better than standard decision trees (e.g. C4.5).

Chapter 6

A real-world Fraud Detection Systems: Concept Drift Adaptation with Alert-Feedback Interaction

Part of the results presented in this chapter have been published in the following paper:

- Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. *Credit Card Fraud Detection and Concept-Drift Adaptation with Delayed Supervised Information.* In Neural Networks (IJCNN), The 2015 International Joint Conference on. IEEE, 2015.

Most FDSs monitor streams of credit card transactions by means of classifiers returning alerts for the riskiest payments. Fraud detection differs from conventional classification because, in a first phase, human investigators who have time to assess only a reduced number of alerts providing a small set of supervised samples denoted as *feedbacks*. Labels of the vast majority of transactions are made available only several days later, when customers have possibly reported unauthorized transactions. These transactions define an additional set of supervised samples called *delayed samples*. The delay in obtaining accurate labels and the interaction between alerts and supervised information has to be carefully taken into consideration when learning in a concept-drifting environment.

In this chapter we address a realistic fraud detection setting and we show that feedbacks and delayed samples have to be handled separately. We design two prototypes of FDS on the basis of an ensemble and a sliding-window approach (Section 6.3.1) and we show that the winning strategy consists in training two separate classifiers (on feedbacks and delayed samples, respectively), and then aggregating the outcomes. Experiments on large

datasets show that the alert precision, which is the primary concern of investigators, can substantially be improved by the proposed approach.

In order to obtain precise alerts, feedbacks samples have to receive larger weights than non-alerted transactions and methods that diminish their role in the learning process lead to loss of predictive accuracy.

6.1 Realistic working conditions

As discussed in Section 3.4, FDSs typically relay on classification algorithms to identify transactions at risk of fraud that generate alerts, but are not able to integrate the feedback that investigators provide on the alerts raised by the FDS. As a consequence, most FDSs available in the literature (e.g. [11, 180, 181]) ignore Alert-Feedback Interaction (AFI), making the unrealistic assumption that all transactions are correctly labeled by a supervisor.

With a limited number of investigators only a restricted quantity of alerts can be checked, which means a small set of labeled transactions returned as feedback. Non-alerted transactions are a large set of unsupervised samples that can be either fraud or genuine. Additional labeled observations are obtained by means of cardholders that report unauthorized transactions [20, 61]. The number of customers reporting frauds not detected by the FDS is usually small and hard to model since cardholders have different habits when it comes to check the transcript of credit card transactions given by the bank. Then, every day in addiction to investigators' feedback, we have historical supervised samples for which the labels can safely assumed to be correct after some time. In summary, we can distinguish between two types of supervised samples: i) *feedbacks* provided by investigators and ii) historical transactions whose labels are received with a large delay. We will call the latter *delayed samples* to stress the fact that their label is available only after a while.

In this formulation we assume that the FDS is updated everyday at midnight and the detection model is then applied to all the transactions occurring the following day. Feedbacks of alerted transactions are given by investigators during the day and by the end of the day we have all the feedbacks for the alerts generated by the FDS. In these settings, the ML algorithm learns from the batch of feedbacks available at the end of the day and is not trained from each transaction incrementally, i.e. the algorithm is trained only when a sufficient batch of supervised samples is provided.

6.2 Fraud Detection with Alert-Feedback Interaction

As in Section 5.1.1, we formulate the fraud detection problem as a binary classification task where each transaction is associated to a feature vector x and a label y. Features in x could be the transaction amount, the shop id, the card id, the timestamp or the country, as well as features extracted from the customer profile. Because of the time-varying nature of the transactions' stream, typically, FDSs train (or update) a classifier $\mathcal{K}_t$ every day (t). In general, FDSs receive a continuous stream of transactions and they have to score each transaction online (i.e. in few milliseconds), however, the classifier is updated once a day, to gather a sufficient amount of supervised transactions. The set of transactions arriving at day t, denoted as B_t, is processed by the classifier $\mathcal{K}_{t-1}$ trained in the previous day $(t-1)$. The k riskiest transactions of B_t are reported to the investigators, where $k > 0$ represents the number of alerts the investigators are able to validate. The reported alerts A_t are determined by ranking the transactions of B_t according to the posterior probability $\hat{\mathcal{P}}_{\mathcal{K}_{t-1}}(+|x)$, which is the estimate, returned by $\mathcal{K}_{t-1}$, of the probability for x to be a fraud. More formally, A_t is defined as:

$$\mathrm{A}_t = \{x \in \mathrm{B}_t \text{ s.t. } r(x) \leq k\} \tag{6.1}$$

where $r(x) \in \{1, \ldots, |\mathrm{B}_t|\}$ is the rank of the transaction x according to $\mathcal{P}_{\mathcal{K}_{t-1}}(+|x)$. In other terms, the transaction with the highest probability ranks first ($r(x) = 1$) and the one with the lowest probability ranks last ($r(x) = |\mathrm{B}_t|$).

Once the alerts A_t are generated, investigators provide feedbacks F_t, defining a set of k supervised couples: $\mathrm{F}_t = \{(x, y),\ x \in \mathrm{A}_t\}$, which represents the most recent information that the FDS receives. At day t, we also receive the labels of all the transactions processed at day $t-\delta$, providing a set of delayed supervised couples $\mathrm{D}_{t-\delta} = \{(x, y),\ x \in \mathrm{B}_{t-\delta}\}$, see Figure 6.1. Though investigators have not personally checked these transactions, they are by default assumed to be genuine after δ days, as far as customers do not report frauds.[1] As a result, the labels of all the transactions older than δ days are provided at day t. The problem of receiving delayed labels is also referred to as *verification latency* [239].

Feedbacks F_t can either refer to frauds (correct alerts) or genuine transactions (false alerts): correct alerts are then True Positives (TPs), while false alerts are FPs. Similarly, $\mathrm{D}_{t-\delta}$ contains both fraud (false negative) and genuine transactions (true negatives), although the vast majority of transactions belongs to the genuine class.

The goal of a FDS is to return accurate alerts: when too many FPs are reported, investigators might decide to ignore forthcoming alerts. Thus, what actually matters is

[1] Investigators typically assume that frauds missed by the FDS are reported by customers themselves (e.g. after having checked their credit card balance), within a maximum time-interval of δ days.

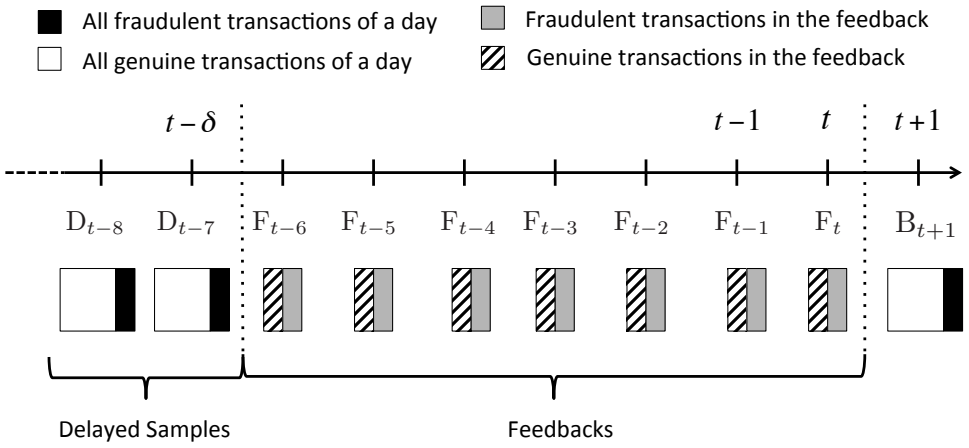


FIGURE 6.1: The supervised samples available at day t include: i) feedbacks of the first δ days and ii) delayed couples occurred before the δ^{th} day. In this Figure we set $\delta = 7$.

to achieve the highest precision in A_t. As shown in Section 2.2.3, this precision can be measured by the quantity

$$\mathrm{P}_k(t) = \frac{|\mathrm{TP}_k(t)|}{k} \tag{6.2}$$

where $\mathrm{TP}_k(t) = \{(x, y) \in \mathrm{A}_t \text{ s.t. } y = +\}$.[2] $\mathrm{P}_k(t)$ is then the proportion of frauds in the top k transactions with the highest estimated likelihood of being frauds [64].

6.3 Learning strategy

The fraud detection scenario described in Section 6.2 suggests that learning from feedbacks F_t is a different problem than learning from delayed samples in $\mathrm{D}_{t-\delta}$. The first difference is evident: F_t provides recent, up-to-date, information while $\mathrm{D}_{t-\delta}$ might be already obsolete once it is available. The second difference concerns the percentage of frauds in F_t and $\mathrm{D}_{t-\delta}$. While it is clear that the class distribution in $\mathrm{D}_{t-\delta}$ is always skewed towards the genuine class, the number of frauds in F_t actually depends on the performance of classifier $\mathcal{K}_{t-1}$: when $\mathrm{P}_k(t) = 0.5$ we have feedbacks F_t with a balanced distribution, while for $\mathrm{P}_k(t) > 0.5$ we have more frauds than genuine transactions in F_t. The third, and probably the most subtle, difference is that supervised couples in F_t are not independently drawn, but are instead selected by $\mathcal{K}_{t-1}$ among those transaction that are more likely to be frauds. As such, a classifier trained on F_t learns how to label transactions that are most likely to be fraudulent, and might be in principle not precise on the vast majority of genuine transactions. Therefore, beside the fact that F_t and $\mathrm{D}_{t-\delta}$ might require different resampling methods, F_t and $\mathrm{D}_{t-\delta}$ are also representative of two

[2]Note that in this formulation $|\mathrm{F}_t| = |\mathrm{A}_t|$ and $|\mathrm{A}_t| = k$.

different classification problems and, as such, they have to be separately handled. In the following, two traditional fraud detection approaches are presented (Section 6.3.1), and further developed to handle separately feedbacks and delayed supervised couples (Section 6.3.2). Experiments in Section 6.5 show that this is a valuable strategy, which substantially improves the alert precision.

6.3.1 Conventional Classification Approaches in FDS

During the FDS operation, both feedbacks F_t and delayed supervised samples $\mathrm{D}_{t-\delta}$ can be exploited for training or updating the classifier $\mathcal{K}_t$. In particular, we train the FDS considering the feedbacks from the last δ days (i.e. $\{\mathrm{F}_t, \mathrm{F}_{t-1}, \ldots, \mathrm{F}_{t-(\delta-1)}\}$) and the delayed supervised pairs from the last M days before the feedbacks, i.e. $\{\mathrm{D}_{t-\delta}, \ldots, \mathrm{D}_{t-(\delta+M-1)}\}$.

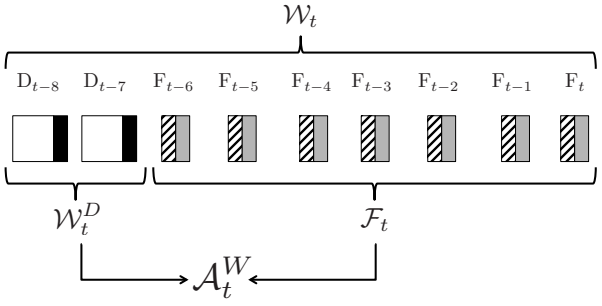
In the following we present two conventional solutions for concept-drift adaptation [171, 182] built upon a classification algorithm proving an estimate of the probability $\mathcal{P}(+|x)$.

- $\mathcal{W}_t$: a *sliding window* classifier that is daily updated over the supervised samples received in the last $\delta + M$ days, i.e. $\{\mathrm{F}_t, \ldots, \mathrm{F}_{t-(\delta-1)}, \mathrm{D}_{t-\delta}, \ldots, \mathrm{D}_{t-(\delta+M-1)}\}$ (see Figure 6.2(a)).
- $\mathcal{E}_t$: an *ensemble* of classifiers $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_M, \mathcal{F}\}$, where $\mathcal{M}_i$ is trained on $\mathrm{D}_{t-(\delta+i-1)}$ and $\mathcal{F}_t$ is trained on all the feedbacks of the last δ days $\{\mathrm{F}_t, \ldots, \mathrm{F}_{t-(\delta-1)}\}$ (see Figure 6.2(b)). The posterior probability $\mathcal{P}_{\mathcal{E}_t}(+|x)$ is estimated by averaging the posterior probabilities of the individual classifiers, $\mathcal{P}_{\mathcal{M}_i}(+|x), i = 1, \ldots, M$ and $\mathcal{P}_{\mathcal{F}_t}(+|x)$. Note that we use a single classifier to learn from the set of feedbacks since their size is typically small. Everyday, $\mathcal{F}_t$ is re-trained considering the new feedbacks, while a new classifier is trained on the new delayed supervised couples provided ($\mathrm{D}_{t-\delta}$) and included in the ensemble. At the same time, the most obsolete classifier is removed from the ensemble.

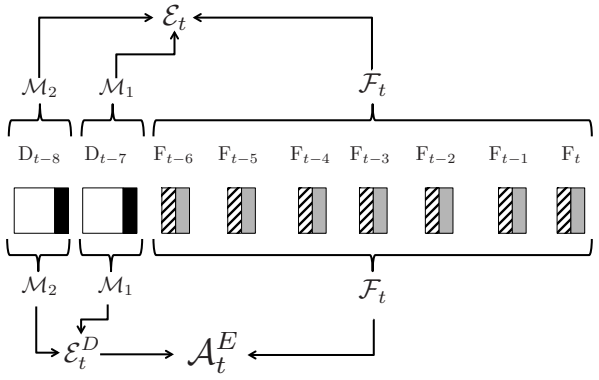
These solutions implement two basic approaches for handling concept drift that can be further improved by adopting dynamic sliding windows or adaptive ensemble sizes [240].

6.3.2 Separating delayed Supervised Samples from Feedbacks

As explained in Section 6.3, our intuition is that feedbacks and delayed transactions have to be treated separately. Therefore, at day t we train a specific classifier $\mathcal{F}_t$ on the feedbacks of the last δ days $\{\mathrm{F}_t, \ldots, \mathrm{F}_{t-(\delta-1)}\}$ and denote by $\mathcal{P}_{\mathcal{F}_t}(+|x)$ its posterior probability. We then train a second classifier on the delayed samples by means either of



(a) **Sliding window** strategies



(b) **Ensemble** strategies

FIGURE 6.2: Learning strategies for feedbacks and delayed transactions occurring in the two days ($M = 2$) before the feedbacks ($\delta = 7$).

a sliding-window or an ensemble mechanism (see Figure 6.2): Let us denote by $\mathcal{W}_t^D$ the classifier trained on a sliding window of delayed samples $\{\mathrm{D}_{t-\delta}, \ldots, \mathrm{D}_{t-(\delta+M-1)}\}$ and by $\mathcal{P}_{\mathcal{W}_t^D}(+|x)$ its posterior probabilities, while $\mathcal{E}_t^D$ denotes the ensemble of M classifiers $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_M\}$ where each individual classifier $\mathcal{M}_i$ is trained on $\mathrm{D}_{t-\delta-i}, i = 1, \ldots, M$. Then, the posterior probability $\mathcal{P}_{\mathcal{E}_t^D}(+|x)$ is obtained by averaging the posterior probabilities of the individual classifiers.

Both "delayed" classifiers $\mathcal{W}_t^D$ and $\mathcal{E}_t^D$ have to be aggregated with $\mathcal{F}_t$ to exploit information provided by feedbacks. However, to raise alerts, we are not interested in aggregation methods at the label level but rather at the posterior probability level. For the sake of simplicity we adopt the most straightforward approach based on a linear combination of the posterior probabilities of the two classifiers ($\mathcal{F}_t$ and one among $\mathcal{W}_t^D$ and $\mathcal{E}_t^D$). Let us denote by $\mathcal{A}_t^E$ the aggregation of $\mathcal{F}_t$ and $\mathcal{E}_t^D$ where $\mathcal{P}_{\mathcal{A}_t^E}(+|x)$ is defined as:

$$\mathcal{P}_{\mathcal{A}_t^E}(+|x) = \alpha_t \mathcal{P}_{\mathcal{F}_t}(+|x) + (1 - \alpha_t)\mathcal{P}_{\mathcal{E}_t^D}(+|x) \tag{6.3}$$

A similar definition holds for $\mathcal{A}_t^W$, the aggregation of $\mathcal{F}_t$ and $\mathcal{W}_t^D$:

$$\mathcal{P}_{\mathcal{A}_t^W}(+|x) = \alpha_t \mathcal{P}_{\mathcal{F}_t}(+|x) + (1 - \alpha_t)\mathcal{P}_{\mathcal{W}_t^D}(+|x) \tag{6.4}$$

Note that $\mathcal{F}_t$ and $\mathcal{W}_t^D$ jointly use the training set of $\mathcal{W}_t$ and, similarly, the two classifiers $\mathcal{F}_t$ and $\mathcal{E}_t^D$ jointly use the same training samples of $\mathcal{E}_t$ (see Figure 6.2).

Feedbacks represent a small portion of the supervised samples used for training classifier $\mathcal{W}_t$, hence they have little influence on $\mathcal{P}_{\mathcal{W}_t}(+|x)$. Similarly, $\mathcal{F}_t$ represents one of the classifiers of the ensemble $\mathcal{E}_t$, hence feedbacks influence $\mathcal{P}_{\mathcal{E}_t}(+|x)$ only for $\frac{1}{M+1}$, while delayed samples for $\frac{M}{M+1}$. By aggregating according to (6.3) and (6.4) we are able to give a larger weight to feedbacks via the parameter α_t, e.g. a larger α_t give greater influence to $\mathcal{F}_t$ in the aggregated posteriori probability $\mathcal{P}_{\mathcal{A}_t^W}$ and $\mathcal{P}_{\mathcal{A}_t^E}$.

Experiments in Section 6.5 show that handling feedbacks separately from delayed supervised samples provides much more precise alerts, and by aggregating with $\alpha_t = 0.5$ we are able to outperform standard FDSs trained on feedbacks and delayed supervised samples pooled together (like $\mathcal{W}_t$ and $\mathcal{E}_t$).[3]

Table 6.1 summarizes the classifiers used in this chapter.

TABLE 6.1: Classifiers used in the chapter.

$\mathcal{F}_t$	A *feedback* classifier daily updated using feedback from t to $t-\delta$.
$\mathcal{W}_t$	A *sliding window* classifier daily updated using $\delta + M$ days.
$\mathcal{W}_t^D$	A *delayed sliding window* classifier daily updated using M days.
$\mathcal{A}_t^W$	An *aggregation* of $\mathcal{W}_t^D$ and $\mathcal{F}_t$ according to (6.3).
$\mathcal{E}_t$	An *ensemble* of classifiers daily updated using $\delta + M$ days.
$\mathcal{E}_t^D$	A *delayed ensemble* of classifiers daily updated using M days.
$\mathcal{M}_i$	i^{th} member of an *ensemble* of classifiers.
$\mathcal{A}_t^E$	An *aggregation* of $\mathcal{E}_t^D$ and $\mathcal{F}_t$ according to (6.4).
α_t	Weight assigned to $\mathcal{F}_t$ in aggregations $\mathcal{A}_t^E$ and $\mathcal{A}_t^W$.

6.3.3 Two Specific FDSs based on Random Forest

The FDSs presented in the previous section has been implement using a Random Forest [43] with 100 trees. In particular, for $\mathcal{W}_t^D$, $\mathcal{W}_t$ and for all $\mathcal{M}_i, i = 1, \ldots, M$, we used a Balanced Random Forest (BRF) [50] where each tree is trained on a balanced bootstrap sample, obtained by randomly undersampling the majority class while preserving all the minority class samples in the corresponding training set. Each tree of BRF receives a different random sample of the genuine transactions and the same samples from the fraud class in the training set, yielding a balanced training set. This undersampling strategy allows one to learn trees with balanced distribution and to exploit many subsets of the majority class. At the same time, this resampling method reduces training time. A

[3] In the specific case of the ensemble approach and $\alpha_t = 0.5$, the aggregation of $\mathcal{F}_t$ and $\mathcal{E}_t^D$ ($\mathcal{A}_t^E$) corresponds to assigning a larger weight to $\mathcal{F}_t$ in $\mathcal{E}_t$.

drawback of undersampling is that we are potentially removing relevant training samples form the dataset, however this problem is mitigated by the fact that we learn 100 different trees. Using undersampling allows us to rebalance the batches without propagating minority class observations along the streams as in [171]. In contrast, for $\mathcal{F}_t$ that is trained on feedbacks we adopted a standard RF where no resampling is performed.

6.4 Selection bias and Alert-Feedback Interaction

In this section we discuss the problem of training a classifier on feedback samples. A conventional assumption in ML is that the learning algorithm receives training and test samples drawn according to the same distribution. However, this assumption does not hold in our case, since Alert-Feedback Interaction (AFI) provides a recent biased subset of supervised transactions (F_t), which is not representative of all the transactions occurring in a day (B_t). As a consequence, classifier $\mathcal{F}_t$ learns under a selection bias governed by AFI.

Let us define a random selection variable $\mathbf{s}$ that associates to each sample in B_t value 1 if the transaction is in F_t and 0 otherwise. When training a classifier $\mathcal{F}_t$ on F_t instead of B_t we get an estimate of $\mathcal{P}_{\mathcal{F}_t}(+|x, \mathbf{s}=1)$ rather than $\mathcal{P}_{\mathcal{F}_t}(+|x)$. As shown in Section 3.2.1 (see (3.1)), a standard solution to SSB consist into training a weight-sensitive algorithm, where a training sample (x, y) receives as weight w:

$$w = \frac{\mathcal{P}(\mathbf{s}=1)}{\mathcal{P}(\mathbf{s}=1|x,y)} \tag{6.5}$$

Alternatively, if the learning algorithm is not able to accept weighted training samples, it suffices to resample the training set with replacement and probability equal to the weights [133].

In equation (6.5), $\mathcal{P}(\mathbf{s}=1)$ can be easily estimate by calculating the proportion of feedbacks in a day. However, $\mathcal{P}(\mathbf{s}=1|x,y)$ is not easy to estimate since we know the label of only few feedback samples. An assumption often made in literature [135–138] is that the selection variable $\mathbf{s}$ is independent of the class y given the input x (*covariate shift*): $\mathcal{P}(\mathbf{s}|y,x) = \mathcal{P}(\mathbf{s}|x)$. Then equation (6.5) becomes:

$$w = \frac{\mathcal{P}(\mathbf{s}=1)}{\mathcal{P}(\mathbf{s}=1|x)} \tag{6.6}$$

We can get an estimate of $\mathcal{P}(\mathbf{s}=1|x)$ by building a classifier that predicts $\mathbf{s}$ as the class label, which is a classifier that learn to discriminate feedbacks and non-feedbacks sample between t and $t-\delta$.

The problem with all these corrections is that, since feedbacks are selected according to $\mathcal{P}(+|x)$, we have that $\mathcal{P}(\mathbf{s} = 1|x)$ and $\mathcal{P}(+|x)$ are highly correlated. Moreover, we expect $\mathcal{P}(\mathbf{s} = 1|x)$ and $\mathcal{P}(\mathbf{s} = 1|x, y)$ to be larger for feedbacks samples, leading to smaller weights in (6.5) and (6.6). This means that importance weighting techniques are expected to lower the influence of feedbacks in the learning process. As shown in our previous work [20], strategies reducing the weight of feedback samples are often returning less precise alerts (lower P_k). For this reason, re-weighting is not expected to bring improvement with AFI. Additionally, the covariate shift assumption is hard to defend, because the percentage of fraud in the feedback is much higher than the one registered in typical day, i.e. the probability of selecting a feedback cannot be said to be independent of the class of the transaction ($\mathcal{P}(\mathbf{s}|y, x) \neq \mathcal{P}(\mathbf{s}|x)$).

6.5 Experiments

In this work we used three large datasets containing credit card transactions made by European cardholders via on-line websites. The first, referred to as 2013, has transactions from September 2013 to January 2014, the second one, referred to as 2014, has transactions from August to December 2014 and the third, referred to as 2015, from January 2015 to end of May 2015. In Table 6.2 we have reported additional information about the data. The datasets contain both original and aggregated features calculated as shown in Section 5.1.3. It is easy to notice that these datasets are highly unbalanced, i.e. frauds account for about 0.2% of all transactions. The number of frauds is also varying significantly over the year (see Figure 6.3).

Table 6.2: Datasets

Id	Start day	End day	# Days	# Transactions	# Features	% Fraud
2013	2013-09-05	2014-01-18	136	21'830'330	51	0.19%
2014	2014-08-05	2014-12-31	148	27'113'187	51	0.22%
2015	2015-01-05	2015-05-31	146	27'651'197	51	0.26%

In the first experiments we process all datasets to assess the importance of separating feedbacks from delayed supervised samples. Though we expect these streams to be affected by Concept Drift (CD), since they span a quite long time range, we do not have any ground truth to investigate the reaction to CD of the proposed FDS. To this purpose, we design the second experiment where we juxtapose batches of transactions acquired in different times of the year to artificially introduce CD in a specific day in the transaction stream.

In both experiments we test FDSs built on Random Forests as presented in Section 6.3.3. We considered both the sliding window and ensemble approaches and compared the

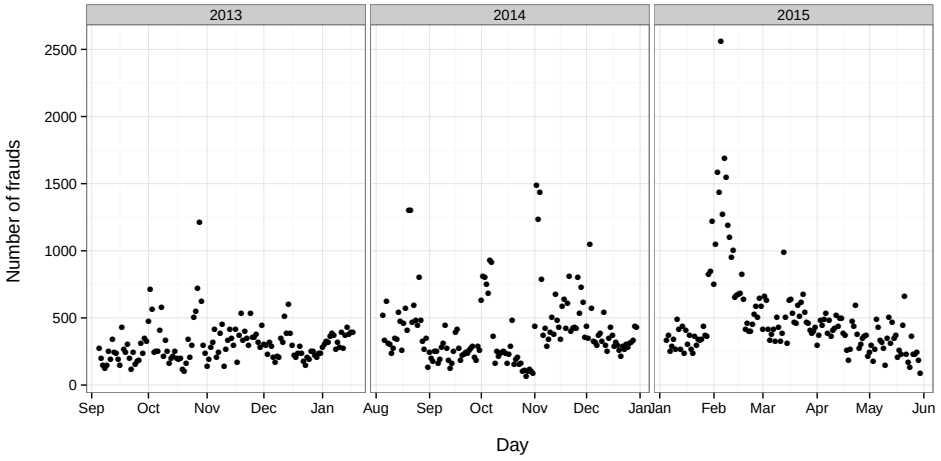


FIGURE 6.3: Number of daily frauds for datasets in Table 6.2.

accuracy of pooling feedbacks and delayed supervised samples together ($\mathcal{W}_t$ and $\mathcal{E}_t$) against learning separate classifiers ($\mathcal{F}_t$, $\mathcal{W}_t^D$ and $\mathcal{E}_t^D$) that are then aggregated ($\mathcal{A}_t^W$ and $\mathcal{A}_t^E$). Let us recall that each tested classifier raises alerts differently. This means that also the feedbacks returned to the classifiers might be different. This has to be considered when comparing several classifiers, for instance, when comparing $\mathcal{W}_t$ and $\mathcal{W}_t^D$, the supervised information provided is not the same because, in the first case alerts are raised by $\mathcal{W}_t$ while in the second by $\mathcal{W}_t^D$.

At first we assume that after $\delta = 7$ days all the transactions labels are provided (delayed supervised information) and that we have a budget of $k = 100$ alerts that can be checked by the investigators: thus, $\mathcal{F}_t$ is trained on a window of 700 feedbacks. Then in the experiments of Section 6.5.3 we use $\delta = 15$ and we allow $\mathcal{F}_t$ to be trained on a larger number of feedbacks per day ($|\mathrm{F}_t| \geq 100$) to see how these parameters influence the performance of the detection. In the same section we test some SSB correction techniques to see whether they can actually improve the performance of $\mathcal{F}_t$.

Experiments from Section 6.5.4 compare the proposed classifiers' aggregation $\mathcal{A}_t^W$ against a classifier $\mathcal{R}_t$ trained on all recent transactions that makes the unrealistic assumption that all daily transactions can be checked by investigators. This experiment aims to compare a classifier $\mathcal{R}_t$ ignoring AFI with the aggregation proposed using different accuracy measures, P_k, AUC and AP.

In Section 6.5.5 we run experiments to study whether adapting the weight α_t given to $\mathcal{F}_t$ in (6.3) and (6.4) has an impact on the performances of $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$. It emerges that combining classifiers with a standard mean ($\alpha_t = 0.5$ for all the data stream) is often competitive to more complicated weighting strategies.

For all experiments, we set $M = 16$ so that $\mathcal{W}_t^D$ is trained on a window of 16 days and $\mathcal{E}_t^D$ (resp. $\mathcal{E}_t$) is an ensemble of 16 (resp. 17) classifiers.[4] Each experiments is repeated 10 times to reduce the results' variability due to bootstrapping of the training sets in the random forests. The FDS performance is assessed by means of the average P_k over all the batches (the higher the better) and use a paired t-test to assess whether the performance gaps between each pair of tested classifiers is significant or not. We compute the paired t-test on the ranks resulting from the Friedman test [232] as recommended by Demsar [241] (see Section 5.1.3).

6.5.1 Separating feedbacks from delayed supervised samples

In order to evaluate the benefit of learning on feedbacks and delayed samples separately, we first compare the performance of classifier $\mathcal{W}_t$ against $\mathcal{F}_t$, $\mathcal{W}_t^D$ and the aggregation $\mathcal{A}_t^W$. The aggregations $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$ are computed using $\alpha_t = 0.5$ over all the data stream. In addition to the classifiers presented in Section 6.3, we consider also a static classifier called $\mathcal{S}_t$ that is trained once on the first M day and never updated. In absence of concept drift we expect $\mathcal{S}_t$ to perform similarly to $\mathcal{W}_t^D$ ($\mathcal{W}_t^D$ is just $\mathcal{S}_t$ updated every day). Table 6.3(a) shows the average P_k over all the batches for the three datasets separately.

TABLE 6.3: Average P_k for the sliding and ensemble strategies ($\delta = 7$, $M = 16$ and $\alpha_t = 0.5$).

(a) **sliding** approach

dataset	classifier	mean	sd
2013	$\mathcal{F}$	0.62	0.25
2013	$\mathcal{W}^D$	0.54	0.22
2013	$\mathcal{W}$	0.57	0.23
2013	$\mathcal{S}$	0.48	0.25
2013	$\mathcal{A}^W$	0.70	0.21
2014	$\mathcal{F}$	0.59	0.29
2014	$\mathcal{W}^D$	0.58	0.26
2014	$\mathcal{W}$	0.60	0.26
2014	$\mathcal{S}$	0.54	0.24
2014	$\mathcal{A}^W$	0.69	0.24
2015	$\mathcal{F}$	0.67	0.25
2015	$\mathcal{W}^D$	0.66	0.21
2015	$\mathcal{W}$	0.68	0.21
2015	$\mathcal{S}$	0.58	0.23
2015	$\mathcal{A}^W$	0.75	0.20

(b) **ensemble** approach

dataset	classifier	mean	sd
2013	$\mathcal{F}$	0.62	0.25
2013	$\mathcal{E}^D$	0.50	0.24
2013	$\mathcal{E}$	0.58	0.24
2013	$\mathcal{S}$	0.47	0.24
2013	$\mathcal{A}^E$	0.69	0.21
2014	$\mathcal{F}$	0.60	0.30
2014	$\mathcal{E}^D$	0.49	0.28
2014	$\mathcal{E}$	0.56	0.27
2014	$\mathcal{S}$	0.53	0.25
2014	$\mathcal{A}^E$	0.68	0.26
2015	$\mathcal{F}$	0.66	0.25
2015	$\mathcal{E}^D$	0.61	0.19
2015	$\mathcal{E}$	0.67	0.20
2015	$\mathcal{S}$	0.58	0.23
2015	$\mathcal{A}^E$	0.74	0.20

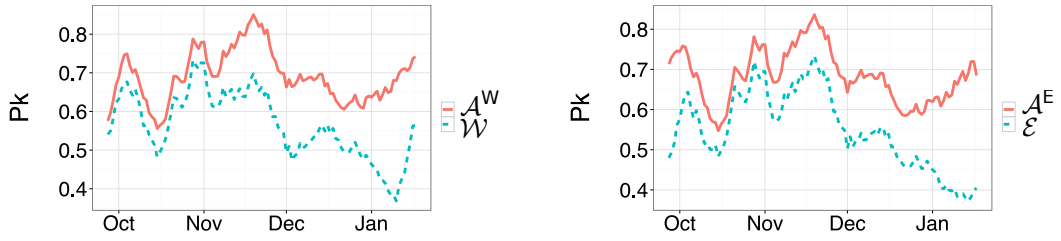
In all datasets, $\mathcal{A}_t^W$ outperforms the other FDSs in terms of P_k. The barplots of Figure 6.5 show the sum of ranks for each classifier and the results of the paired t-tests. Figure 6.5 indicates that in all datasets (Figures 6.5(a), 6.5(b) and 6.5(c)) $\mathcal{A}_t^W$ is significantly

[4] We ran several experiments with $M = 1, 8, 16, 24$ and found $M = 16$ as a good trade-off between performance, computational load, and the number of days that can be used for testing in each stream.

better than all the other classifiers. $\mathcal{F}_t$ achieves higher average P_k and higher sum of ranks than $\mathcal{W}_t^D$ and $\mathcal{W}_t$: this confirms that feedbacks are very important to increase P_k.

Figure 6.4(a) displays the value of P_k for $\mathcal{A}_t^W$ and $\mathcal{W}_t$ in each day, averaged in a neighborhood of 15 days. After December there is a substantial performance drop, which can be seen as a CD due to a change in cardholder behaviour in the period before and after Christmas. However, $\mathcal{A}_t^W$ dominates $\mathcal{W}_t$ along the whole 2013 dataset, which confirms that a classifier $\mathcal{A}_t^W$ that learns on feedbacks and delayed transactions separately outperforms a classifier $\mathcal{W}_t$ trained on all the supervised information pooled together (feedbacks and delayed transactions).

Table 6.3(b) and Figures 6.5(d), 6.5(e) and 6.5(f) confirm this claim also when the FDS implements an ensemble of classifiers.[5] In particular, Figure 6.4(b) displays the smoothed average P_k of classifiers $\mathcal{A}_t^E$ and $\mathcal{E}_t$. For the whole dataset $\mathcal{A}_t^E$ has better P_k than $\mathcal{E}_t$.



(a) **Sliding window** strategies

(b) **Ensemble** strategies

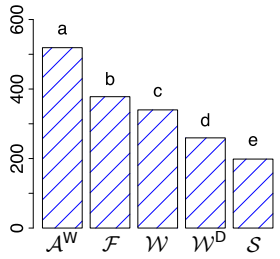
FIGURE 6.4: Average P_k per day (the higher the better) for classifiers on dataset 2013 smoothed using moving average of 15 days. In the sliding window approach classifier $\mathcal{A}_t^W$ has higher P_k than $\mathcal{W}_t$, and in the ensemble approach $\mathcal{A}_t^E$ is superior than $\mathcal{E}_t$.

6.5.2 Artificial dataset with Concept Drift

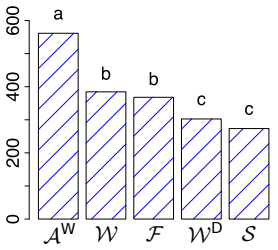
The rationale of this experiment is to test the reaction of the proposed FDS to abrupt CD. To this purpose, in this section we artificially introduce an abrupt CD in specific days by juxtaposing transactions acquired in different times of the year. Table 6.4 reports the three datasets that have been generated by concatenating batches of the dataset 2013 with batches from 2014. The number of days after CD is set such that the FDS has the time to forget the information from the previous concept.

Table 6.5(a) shows the values of P_k averaged over all the batches in the month before the change for the sliding window approach, while Table 6.5(b) shows P_k in the month after the CD. $\mathcal{A}_t^W$ reports the highest P_k before and after CD. Similar results are obtained with

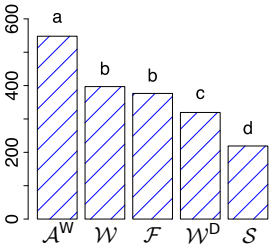
[5] Please note that classifier $\mathcal{F}_t$ returns different results between 6.3(a) and 6.3(b) because of the stochastic nature of RF.



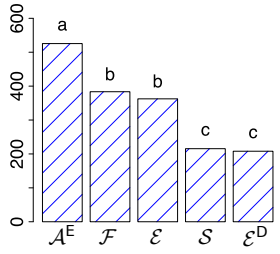
(a) **Sliding window** strategies on dataset **2013**



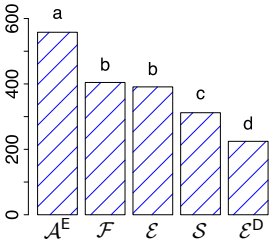
(b) **Sliding window** strategies on dataset **2014**



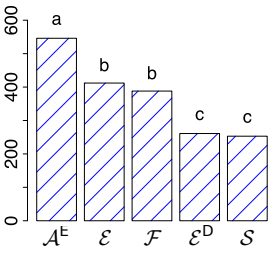
(c) **Sliding window** strategies on dataset **2015**



(d) **Ensemble** strategies on dataset **2013**



(e) **Ensemble** strategies on dataset **2014**



(f) **Ensemble** strategies on dataset **2015**

FIGURE 6.5: Comparison of classification strategies using sum of ranks in all batches and paired t-test based upon on the ranks of each batch (classifiers having the same label on their bar are not significantly different with a confidence level of 0.95). In all datasets (2013, 2014 and 2015), classifiers aggregation $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$ are significantly better that the others.

TABLE 6.4: Datasets with Artificially Introduced CD

Id	Start 2013	End 2013	Start 2014	End 2014
CD1	2013-09-05	2013-09-30	2014-08-05	2014-08-31
CD2	2013-10-01	2013-10-31	2014-09-01	2014-09-30
CD3	2013-11-01	2013-11-30	2014-08-05	2014-08-31

TABLE 6.5: Average P_k in the month before and after CD for the sliding window

(a) **Before** CD

	CD1		CD2		CD3	
classifier	mean	sd	mean	sd	mean	sd
$\mathcal{F}$	0.411	0.142	0.754	0.270	0.690	0.252
$\mathcal{W}^D$	0.291	0.129	0.757	0.265	0.622	0.228
$\mathcal{W}$	0.332	0.215	0.758	0.261	0.640	0.227
$\mathcal{A}^W$	0.598	0.192	0.788	0.261	0.768	0.221

(b) **After** CD

	CD1		CD2		CD3	
classifier	mean	sd	mean	sd	mean	sd
$\mathcal{F}$	0.635	0.279	0.511	0.224	0.599	0.271
$\mathcal{W}^D$	0.536	0.335	0.374	0.218	0.515	0.331
$\mathcal{W}$	0.570	0.309	0.391	0.213	0.546	0.319
$\mathcal{A}^W$	0.714	0.250	0.594	0.210	0.675	0.244

the ensemble approach (Tables 6.6(a), 6.6(b)). In all these experiments, $\mathcal{A}_t^E$ is also faster than standard classifiers $\mathcal{E}_t$ and $\mathcal{W}_t$ to react in the presence of a CD (see Figure 6.6). The large variation of P_k over the time reflects the non-stationarity of the data stream. Expect for dataset CD1, we have on average lower P_k after Concept Drift.

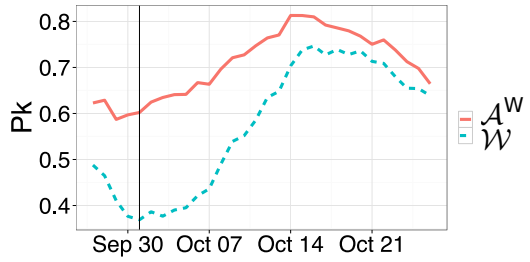
TABLE 6.6: Average P_k in the month before and after CD for the ensemble

(a) **Before** CD

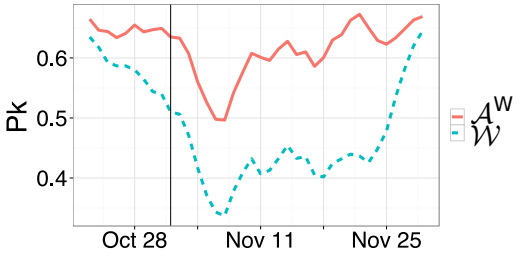
	CD1		CD2		CD3	
classifier	mean	sd	mean	sd	mean	sd
$\mathcal{F}$	0.585	0.183	0.731	0.267	0.706	0.245
$\mathcal{E}^D$	0.555	0.318	0.563	0.217	0.562	0.223
$\mathcal{E}$	0.618	0.313	0.696	0.276	0.648	0.245
$\mathcal{A}^E$	0.666	0.222	0.772	0.272	0.751	0.221

(b) **After** CD

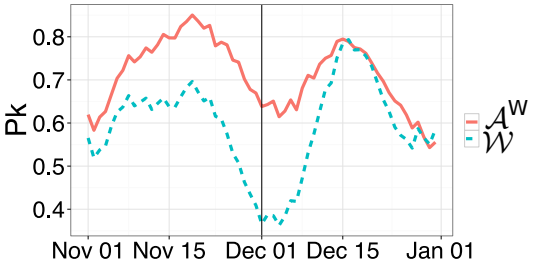
	CD1		CD2		CD3	
classifier	mean	sd	mean	sd	mean	sd
$\mathcal{F}$	0.696	0.270	0.477	0.235	0.610	0.270
$\mathcal{E}^D$	0.551	0.298	0.286	0.182	0.486	0.265
$\mathcal{E}$	0.654	0.266	0.373	0.235	0.581	0.268
$\mathcal{A}^E$	0.740	0.232	0.575	0.227	0.659	0.245



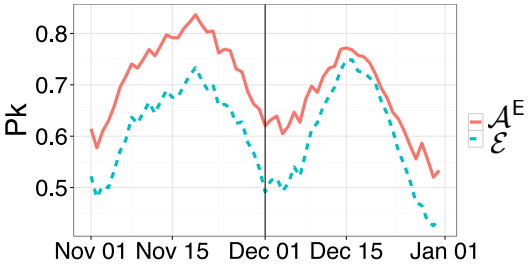
(a) **Sliding window** strategies on dataset **CD1**



(b) **Sliding window** strategies on dataset **CD2**



(c) **Sliding window** strategies on dataset **CD3**



(d) **Ensemble** strategies on dataset **CD3**

FIGURE 6.6: Average P_k per day (the higher the better) for classifiers on datasets with artificial concept drift (CD1, CD2 and CD3) smoothed using moving average of 15 days. In all datasets $\mathcal{A}_t^W$ has higher P_k than $\mathcal{W}_t$. For the ensemble approach we show only dataset CD3, where $\mathcal{A}_t^E$ dominates $\mathcal{E}_t$ for the whole dataset (similar results are obtained on CD1 and CD2, but they are not included for compactness). The vertical bar denotes the date of the concept drift.

6.5.3 Improving the performance of the feedback classifier

The goal of these experiments is to see how the performance of $\mathcal{F}_t$ are influenced by: i) the number of days of feedback available (defined by δ), ii) methods correcting the selection bias, and iii) the number of feedbacks available everyday. To this purpose we first test strategies with $\mathcal{F}_t$ trained on 15 days ($\delta = 15$). In Table 6.7 we see that, with $\delta = 15$, $\mathcal{F}_t$ has higher P_k than when it is trained with $\delta = 7$ (see Table 6.3). The aggregations $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$ also improve their performances with $\delta = 15$.

The rationale of the second experiment is to see where or not the methods for SSB would improve the performance of the feedback classifier $\mathcal{F}_t$. In Table 6.8 we test two types of methods for correcting the selection bias: importance weighting with weights

TABLE 6.7: Average P_k for the sliding and ensemble approach when $\delta = 15$.

(a) **sliding** approach

dataset	classifier	mean	sd
2013	$\mathcal{F}$	0.67	0.24
2013	$\mathcal{W}^D$	0.54	0.22
2013	$\mathcal{A}^W$	0.72	0.20
2014	$\mathcal{F}$	0.67	0.27
2014	$\mathcal{W}^D$	0.56	0.27
2014	$\mathcal{A}^W$	0.71	0.25
2015	$\mathcal{F}$	0.71	0.21
2015	$\mathcal{W}^D$	0.62	0.21
2015	$\mathcal{A}^W$	0.76	0.19

(b) **ensemble** approach

dataset	classifier	mean	sd
2013	$\mathcal{F}$	0.67	0.23
2013	$\mathcal{E}^D$	0.48	0.23
2013	$\mathcal{A}^E$	0.72	0.21
2014	$\mathcal{F}$	0.67	0.28
2014	$\mathcal{E}^D$	0.49	0.27
2014	$\mathcal{A}^E$	0.70	0.25
2015	$\mathcal{F}$	0.71	0.22
2015	$\mathcal{E}^D$	0.56	0.18
2015	$\mathcal{A}^E$	0.75	0.20

provided by (6.5) and Joint Probability Averaging [242]. In the first case we used as weight-sensitive algorithm an implementation of the Random Forest based on conditional inference trees [243] available in the `party` package [244]. In the second case we used the semi-supervised version (SJA) of the algorithms proposed by Fan et al. [242] for SSB correction. These results are obtained with the sliding approach when $\delta = 15$, but equivalent one can be obtained with the ensemble since $\mathcal{F}_t$ is the same for both approaches. If we compare the results of Table 6.8 with the one of $\mathcal{F}_t$ in Table 6.7 we see that techniques for SSB do not improve the performance of $\mathcal{F}_t$, and weighting techniques can actually deteriorate its performance.

TABLE 6.8: Average P_k of $\mathcal{F}_t$ with methods for SSB correction when $\delta = 15$.

dataset	SSB correction	mean	sd
2013	SJA	0.66	0.24
2013	weigthing	0.64	0.25
2014	SJA	0.66	0.27
2014	weigthing	0.65	0.28
2015	SJA	0.71	0.22
2015	weigthing	0.68	0.23

In the remainder of the section we investigate how requesting a larger number of feedbacks (i.e. generating a larger number of alerts) influences the performances of $\mathcal{F}_t$ and its aggregation. If having more feedbacks, leads to better performances, then the company should hire more investigators. In Table 6.9 we allow $\mathcal{F}_t$ to be trained on a larger number of feedbacks ($|\mathrm{F}_t| \geq 100$) and evaluate the performances with P_k using $k = 100$ for all experiments. It emerges that, the company would obtain a better detection by increasing the number of investigators. However, the increase in accuracy should be evaluated against the cost of hiring more investigators, perhaps better accuracy is not sufficient to justify the higher cost of investigation.

TABLE 6.9: Average P_k for the sliding approach with more than 100 feedbacks per day when $\delta = 15$ and $k = 100$.

(a) dataset **2013**

$\lvert F_t \rvert$	classifier	mean	sd
100	$\mathcal{F}$	0.68	0.23
100	$\mathcal{A}^W$	0.72	0.20
300	$\mathcal{F}$	0.74	0.22
300	$\mathcal{A}^W$	0.78	0.19
500	$\mathcal{F}$	0.76	0.20
500	$\mathcal{A}^W$	0.79	0.18

(b) dataset **2014**

$\lvert F_t \rvert$	classifier	mean	sd
100	$\mathcal{F}$	0.68	0.27
100	$\mathcal{A}^W$	0.71	0.24
300	$\mathcal{F}$	0.73	0.25
300	$\mathcal{A}^W$	0.76	0.22
500	$\mathcal{F}$	0.75	0.23
500	$\mathcal{A}^W$	0.78	0.21

(c) dataset **2015**

$\lvert F_t \rvert$	classifier	mean	sd
100	$\mathcal{F}$	0.71	0.21
100	$\mathcal{A}^W$	0.76	0.19
300	$\mathcal{F}$	0.80	0.18
300	$\mathcal{A}^W$	0.82	0.17
500	$\mathcal{F}$	0.82	0.17
500	$\mathcal{A}^W$	0.83	0.17

6.5.4 Standard accuracy measures and classifiers ignoring AFI

In this section we compare the performance of the feedback classifier $\mathcal{F}_t$ and its aggregation $\mathcal{A}_t^W$ against a classifier $\mathcal{R}_t$ that is trained on all recent transactions occurred between t and $t - \delta$ (feedbacks and non-alerted transactions, see Figure 6.7). In this experiment we want to see how a classifier $\mathcal{R}_t$ ignoring Alert-Feedback Interaction compares to $\mathcal{F}_t$ (trained only on feedbacks).[6] It is important to remark than $\mathcal{R}_t$ and $\mathcal{F}_t$ are trained on the same number of days (defined by δ), while $\mathcal{W}_t^D$ is trained on a window of delayed samples. We assess the results using P_k and two other detection measures presented in Section 2.2.3, namely AUC and AP. Note that these last figures of merit can be considered as global ranking measures, because they assess the ranking quality over all the instances, not only in the top k as P_k does. As explained in Section 6.1, most works in fraud detection use AUC as performances measure and train a classifier using all available information like $\mathcal{R}_t$ without considering investigators' feedback.

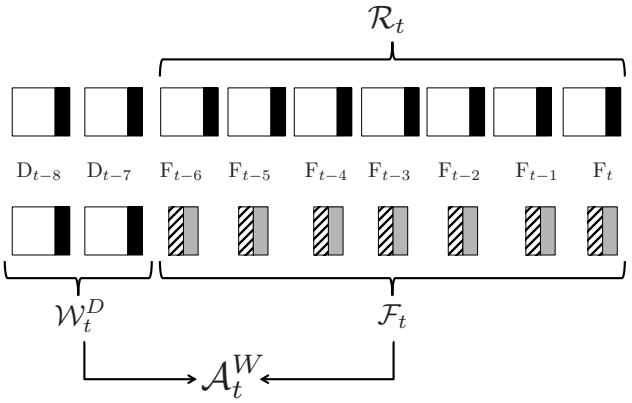


FIGURE 6.7: A classifier $\mathcal{R}_t$ trained on all recent transactions occurred between t and $t - \delta$ makes the unrealistic assumption that all these transactions have been checked and labeled by investigators. In this figure we use $\delta = 7$ and $M = 2$.

Table 6.10 reports the results for the three datasets. We notice that, when using global ranking measure such as AUC and AP, $\mathcal{R}_t$ always outperforms $\mathcal{F}_t$ and the latter is

[6] Note that in a real scenario we cannot compute $\mathcal{R}_t$, since labels are available only for feedback samples between t and $t - \delta$.

often the worst between all classifiers considered. On the contrary, when the accuracy is evaluated in terms of P_k, then $\mathcal{F}_t$ is a better alternative to $\mathcal{R}_t$. These results can be interpreted in the following way: when the objective is to get accurate ranking of the most suspicious transactions (maximize P_k) we should use a classifier trained on transactions that are as risky as the one we want to predict, hence $\mathcal{F}_t$ should be favored because it contains only risky samples (feedbacks). On the contrary, a classifier trained on all daily transactions (which are mostly genuine), like $\mathcal{R}_t$, will be a better choice if we want to obtain a good ranking on all samples (e.g. maximize AUC).

From Table 6.10 and Figure 6.8 we see also that $\mathcal{R}_t$ has in general better accuracy than $\mathcal{W}_t^D$, which confirms the presence of a non-stationary environment and training on more recent transactions allows one to react faster to CD. The aggregation of $\mathcal{F}_t$ and $\mathcal{W}_t^D$ in $\mathcal{A}_t^W$ returns higher P_k than $\mathcal{R}_t$, but it is beaten by $\mathcal{R}_t$ in terms of AUC. In a nutshell, to have good prediction on the k transactions with the largest fraud risk it is better to train a specific classifier on the feedbacks (previous risky transactions). When the objective is instead to obtain good overall accuracy (not only on the top k) we should perhaps use a standard classifier that use all information available.

TABLE 6.10: Average AP, AUC and P_k for the sliding approach ($\delta = 15$, $M = 16$ and $\alpha_t = 0.5$).

(a) dataset **2013**

metric	classifier	mean	sd
AP	$\mathcal{F}$	0.31	0.13
AP	$\mathcal{W}^D$	0.28	0.15
AP	$\mathcal{R}$	0.33	0.15
AP	$\mathcal{A}^W$	0.40	0.14
AUC	$\mathcal{F}$	0.83	0.06
AUC	$\mathcal{W}^D$	0.94	0.03
AUC	$\mathcal{R}$	0.96	0.01
AUC	$\mathcal{A}^W$	0.94	0.03
P_k	$\mathcal{F}$	0.67	0.24
P_k	$\mathcal{W}^D$	0.54	0.22
P_k	$\mathcal{R}$	0.60	0.22
P_k	$\mathcal{A}^W$	0.72	0.20

(b) dataset **2014**

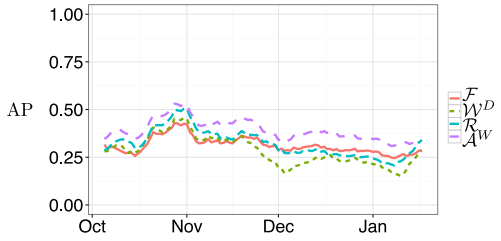
metric	classifier	mean	sd
AP	$\mathcal{F}$	0.30	0.15
AP	$\mathcal{W}^D$	0.29	0.17
AP	$\mathcal{R}$	0.35	0.19
AP	$\mathcal{A}^W$	0.39	0.16
AUC	$\mathcal{F}$	0.81	0.08
AUC	$\mathcal{W}^D$	0.94	0.03
AUC	$\mathcal{R}$	0.96	0.02
AUC	$\mathcal{A}^W$	0.93	0.03
P_k	$\mathcal{F}$	0.67	0.27
P_k	$\mathcal{W}^D$	0.56	0.27
P_k	$\mathcal{R}$	0.63	0.25
P_k	$\mathcal{A}^W$	0.71	0.25

(c) dataset **2015**

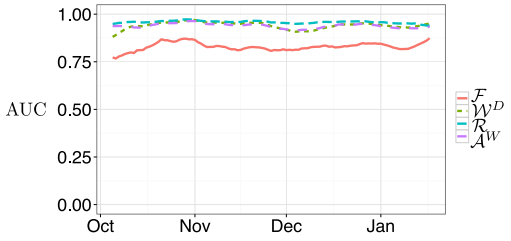
metric	classifier	mean	sd
AP	$\mathcal{F}$	0.27	0.11
AP	$\mathcal{W}^D$	0.30	0.13
AP	$\mathcal{R}$	0.39	0.16
AP	$\mathcal{A}^W$	0.37	0.12
AUC	$\mathcal{F}$	0.81	0.07
AUC	$\mathcal{W}^D$	0.95	0.02
AUC	$\mathcal{R}$	0.97	0.01
AUC	$\mathcal{A}^W$	0.94	0.02
P_k	$\mathcal{F}$	0.71	0.21
P_k	$\mathcal{W}^D$	0.62	0.21
P_k	$\mathcal{R}$	0.68	0.20
P_k	$\mathcal{A}^W$	0.76	0.19

6.5.5 Adaptive aggregation

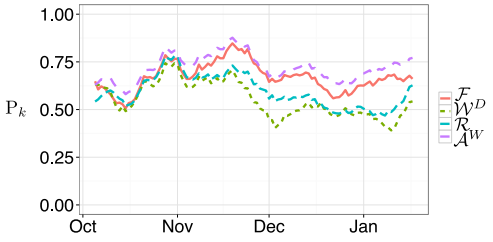
The rational of following experiments is to investigate how to adapt the aggregations $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$ in the presence of non-stationary environments. Let us recall that the posterior probability of $\mathcal{A}_t^W$ is defined as follow: $\mathcal{P}_{\mathcal{A}_t^W}(+|x) = \alpha_t \mathcal{P}_{\mathcal{F}_t} + (1-\alpha_t)\mathcal{P}_{\mathcal{W}_t^D}$, where we have introduced the compact notation $\mathcal{P}_{\mathcal{F}_t}$, $\mathcal{P}_{\mathcal{W}_t^D}$ to denote $\mathcal{P}_{\mathcal{F}_t}(+|x)$, $\mathcal{P}_{\mathcal{W}_t^D}(+|x)$. Similar definition holds for $\mathcal{A}_t^E$ (aggregation of $\mathcal{F}_t$ and $\mathcal{E}_t^D$, see (6.4)). Perhaps the most straightforward adaptation strategy is to set the weight at day $t+1$ (α_{t+1}) depending on the performances of $\mathcal{F}_t$ and $\mathcal{W}_t^D$ (or $\mathcal{E}_t^D$). Operating with different values of α_t



(a) Ranking measured in terms of **AP**



(b) Ranking measured in terms of **AUC**



(c) Ranking measured in terms of $\mathbf{P}_k$

FIGURE 6.8: Detection accuracy of $\mathcal{A}_t^W$, $\mathcal{F}_t$, $\mathcal{W}_t^D$ and $\mathcal{R}_t$ measured using different performance measures on the 2013 dataset. AUC and AP are measures of global ranking while P_k is a measure of ranking in the top k transactions with the largest probability of being fraudulent.

corresponds to cutting the plane $(\mathcal{P}_{\mathcal{F}_t}, \mathcal{P}_{\mathcal{W}_t^D})$ using straight lines having different angular coefficient (see Figure 6.10).

In our aggregations we would like to give a larger weight to the probability that is more accurate, so if $\mathcal{P}_{\mathcal{F}_t}$ is more accurate than $\mathcal{P}_{\mathcal{W}_t^D}$ (or $\mathcal{P}_{\mathcal{E}_t^D}$), at day $t+1$ we want $\alpha_{t+1} > 0.5$. In order to define the weight α_{t+1} we have first to decide how to measure the accuracy of $\mathcal{F}_t$ when the classifier generating the alerts (and requesting feedbacks) is $\mathcal{A}_t^W$.[7]

Let $\mathrm{F}_{\mathcal{A}_t^W}$ be the feedbacks requested by $\mathcal{A}_t^W$ and $\mathrm{F}_{\mathcal{F}_t}$ be the feedbacks requested by $\mathcal{F}_t$ (see Figure 6.9). Let Y_t^+ (resp. Y_t^-) be the subset of fraudulent (resp. genuine) transactions in day t, we define the following sets: $\mathrm{CORR}_{\mathcal{F}_t} = \mathrm{F}_{\mathcal{A}_t^W} \cap \mathrm{F}_{\mathcal{F}_t} \cap \mathrm{Y}_t^+$, $\mathrm{ERR}_{\mathcal{F}_t} = \mathrm{F}_{\mathcal{A}_t^W} \cap \mathrm{F}_{\mathcal{F}_t} \cap \mathrm{Y}_t^-$, and $\mathrm{MISS}_{\mathcal{F}_t} = \mathrm{F}_{\mathcal{A}_t^W} \cap \{\mathrm{B}_t \setminus \mathrm{F}_{\mathcal{F}_t}\} \cap \mathrm{Y}_t^+$. In the example of Figure 6.9 we have $|\mathrm{CORR}_{\mathcal{F}_t}| = 4$, $|\mathrm{ERR}_{\mathcal{F}_t}| = 2$ and $|\mathrm{MISS}_{\mathcal{F}_t}| = 3$. We can now compute some accuracy measures of $\mathcal{F}_t$ in the feedbacks requested by $\mathcal{A}_t^W$:[8]

- $acc_{\mathcal{F}_t} = 1 - \frac{|\mathrm{ERR}_{\mathcal{F}_t}|}{k}$
- $accMiss_{\mathcal{F}_t} = 1 - \frac{|\mathrm{ERR}_{\mathcal{F}_t}| + |\mathrm{MISS}_{\mathcal{F}_t}|}{k}$
- $precision_{\mathcal{F}_t} = \frac{|\mathrm{CORR}_{\mathcal{F}_t}|}{k}$

[7] We cannot compute P_k of $\mathcal{F}_t$ and $\mathcal{W}_t^D$ because feedbacks are requested by $\mathcal{A}_t^W$.
[8] These accuracy measures are inspired by standard classification metrics presented in Section 2.1.4

- $recall_{\mathcal{F}_t} = \frac{|\mathrm{CORR}_{\mathcal{F}_t}|}{|\mathrm{F}_{\mathcal{A}_t^W} \cap \mathrm{Y}_t^+|}$
- $auc_{\mathcal{F}_t}$ = probability that fraudulent feedbacks rank higher than genuine feedbacks in $\mathrm{F}_{\mathcal{A}_t^W}$ according to $\mathcal{P}_{\mathcal{F}_t}$.
- $probDif_{\mathcal{F}_t}$ = difference between the mean of $\mathcal{P}_{\mathcal{F}_t}$ calculated on the frauds and the mean on the genuine in $\mathrm{F}_{\mathcal{A}_t^W}$.

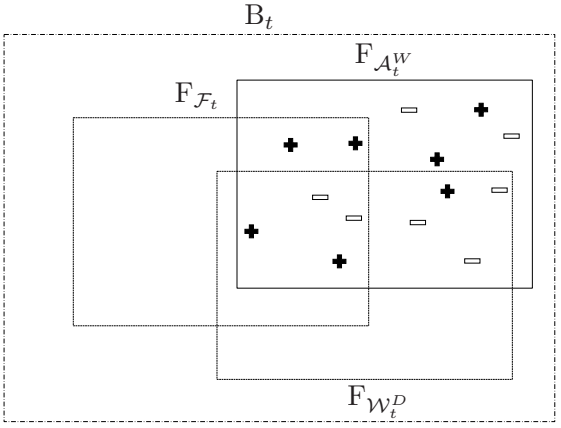


FIGURE 6.9: Feedbacks requested by $\mathcal{A}_t^W$ ($\mathrm{F}_{\mathcal{A}_t^W}$) are a subset of all the transactions of day t (B_t). $\mathrm{F}_{\mathcal{F}_t}$ ($\mathrm{F}_{\mathcal{W}_t^D}$) denotes the feedbacks requested by $\mathcal{F}_t$ ($\mathcal{W}_t^D$). The symbol + is used for frauds and − for genuine transactions.

Similarly, we can compute the accuracy of classifier $\mathcal{W}_t^D$ (or $\mathcal{E}_t^D$) in the feedbacks requested by $\mathcal{A}_t^W$. For example, in Figure 6.9 we have $|\mathrm{CORR}_{\mathcal{W}_t^D}| = 3$, $|\mathrm{ERR}_{\mathcal{W}_t^D}| = 5$ and $|\mathrm{MISS}_{\mathcal{W}_t^D}| = 4$. By choosing one of the previous accuracy measures we obtain different ways to combine the probabilities and adapt to CD at the aggregation level. Let's imagine we choose $auc_{\mathcal{F}_t}$ as metric for $\mathcal{F}_t$ and similarly $auc_{\mathcal{W}_t^D}$ for $\mathcal{W}_t^D$, then we compute α_{t+1} as follows:

$$\alpha_{t+1} = \frac{auc_{\mathcal{F}_t}}{auc_{\mathcal{F}_t} + auc_{\mathcal{W}_t^D}} \tag{6.7}$$

Using (6.7) ensures that $\alpha_{t+1} \in [0, 1]$. In Table 6.11 we present the average P_k of classifier $\mathcal{A}_t^W$ when $\delta = 15$ with adaptive α_t, i.e. α_t changes everyday according to one of the accuracy measures presented before.

We have also considered the ideal case in which we could use everyday the value of α_t returning the best results, by testing different values $\alpha_t \in \{0.1, \ldots, 0.9\}$ and selecting α_t^* as the one allowing the aggregation to have the highest P_k. Note that in a real scenario it is not possible to compute α_t^* since we cannot ask feedbacks for more than one classifier and all the feedbacks are available only at the end of the day. Table 6.11 reports results obtained with α_t^* under the name *best* for α adaptation. In general, the performances increase with α_t^* is marginal w.r.t. all the other strategies considered. Also, it appears that adapting the weight is not significantly better than keeping everyday $\alpha_t = 0.5$.

TABLE 6.11: Average P_k of $\mathcal{A}_t^W$ with adaptive α_t ($\delta = 15$).

(a) dataset **2013**

α adaptation	mean	sd
$best(\alpha_t^*)$	0.73	0.21
$probDif$	0.70	0.22
acc	0.72	0.21
$accMiss$	0.72	0.21
$precision$	0.72	0.21
$recall$	0.71	0.21
auc	0.72	0.21
$none(\alpha_t = 0.5)$	0.72	0.20

(b) dataset **2014**

α adaptation	mean	sd
$best(\alpha_t^*)$	0.72	0.24
$probDif$	0.68	0.26
acc	0.71	0.24
$accMiss$	0.70	0.24
$precision$	0.71	0.24
$recall$	0.70	0.25
auc	0.71	0.24
$none(\alpha_t = 0.5)$	0.71	0.24

(c) dataset **2015**

α adaptation	mean	sd
$best(\alpha_t^*)$	0.76	0.19
$probDif$	0.73	0.21
acc	0.76	0.19
$accMiss$	0.75	0.19
$precision$	0.75	0.19
$recall$	0.74	0.20
auc	0.75	0.19
$none(\alpha_t = 0.5)$	0.76	0.19

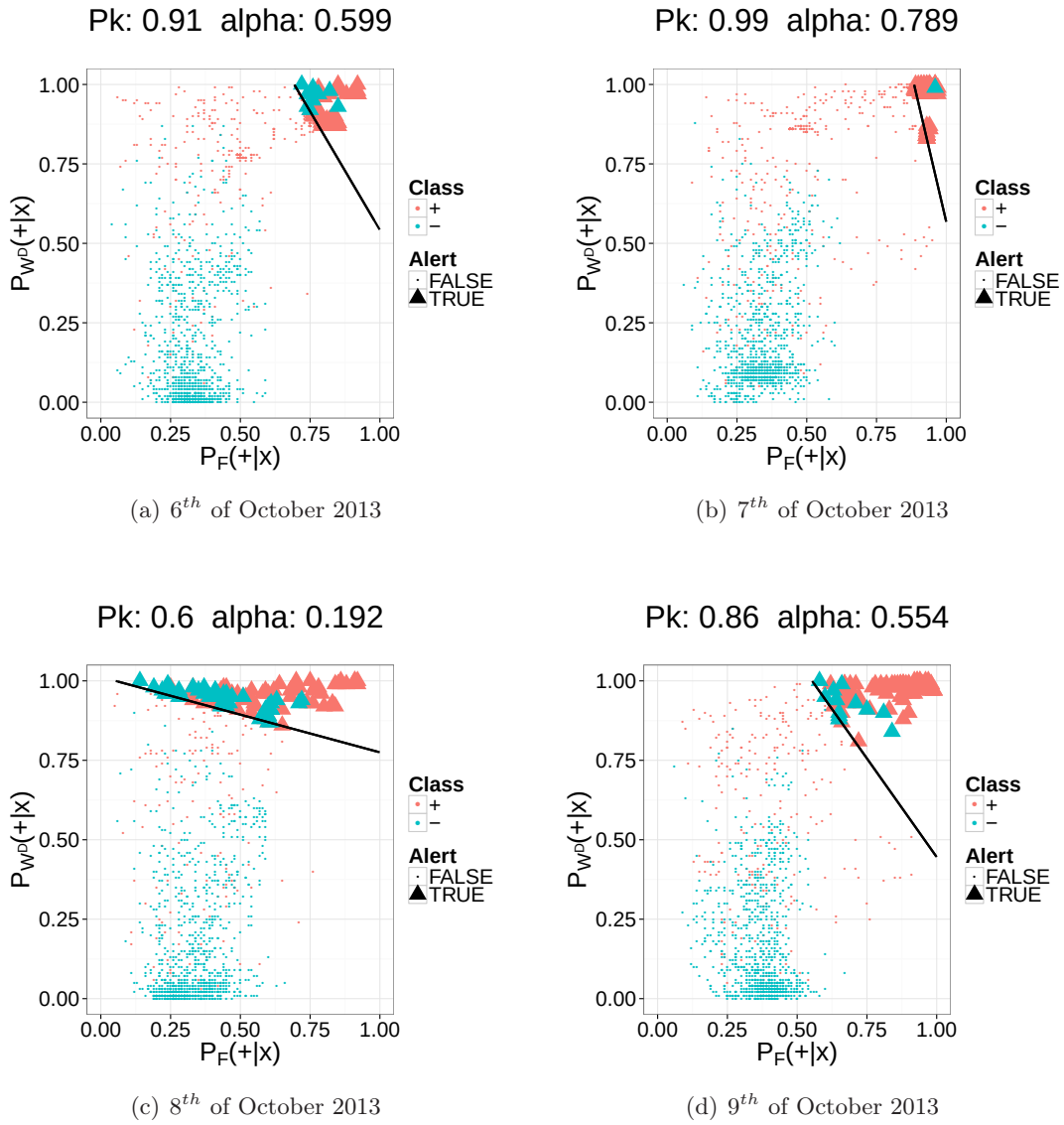
6.5.6 Final strategy selection and classification model analysis

The aim of this section is to select the final strategy for the FDS, make an analysis of the time complexity of the proposed solution and understand which features of the dataset are the most informative. In the previous experiments we saw that aggregating a feedback classifier ($\mathcal{F}_t$) with a delayed classifier ($\mathcal{W}_t^D$ or $\mathcal{E}_t^D$) using (6.3) or (6.4) is often the best solution. Overall we found the best performances using the configurations of Table 6.7: $\delta = 15$, $M = 16$ and $\alpha_t = 0.5$. If we have to choose between the sliding window or ensemble approach we would recommend the second. As shown in Figure 6.11(b), the ensemble $\mathcal{E}_t^D$ has lower training time than the sliding window, because the first trains everyday a model $\mathcal{M}_t$ using only transactions from D_{t-d}. On the contrary, in the sliding window a model $\mathcal{W}_t^D$ is built using transactions from $\{D_{t-\delta}, \ldots, D_{t-(\delta+M-1)}\}$, hence $\mathcal{W}_t^D$ is a trained on a much larger training set.

Despite the large difference in the training set size between the ensemble and sliding window for the delayed classifier (see Figure 6.11(a)), the difference in the training time is smaller (see Figure 6.11(b)), because in $\mathcal{W}_t^D$ and $\mathcal{E}_t^D$ each tree of the Random Forest uses a balanced bootstrap sample of the original training set (see Section 6.3.3). The training time of $\mathcal{F}_t$ is negligible since it is trained on few feedbacks samples. As a consequence, the training time of aggregations $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$ are essentially equivalent to the one of the delayed classifier $\mathcal{W}_t^D$ and $\mathcal{E}_t^D$.

The RF algorithm can be easily parallelized by distributing the training of each decision tree on multiple cores/machines. In this way it is possible to drastically reduce the computing time requested for training the FDS. In the ideal case of no overhead due to parallelization, the theoretical speedup is equivalent to the number of cores/machines used for training the RF.

Louppe [245] derives three bounds for the time complexity of RF training procedure: i) best $O(qmN_b \log^2 N_b)$, ii) worst $O(qmN_b^2 \log N_b)$ and iii) average $O(qmN_b \log^2 N_b)$, where N_b is the number of boostrap samples in each tree, m the number of trees in the forest



(a) 6^{th} of October 2013

(b) 7^{th} of October 2013

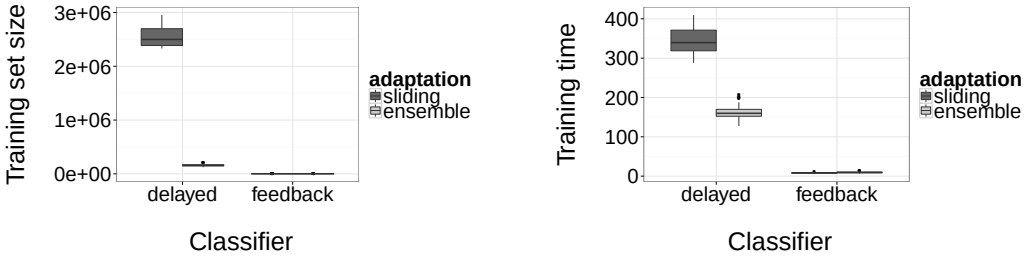
(c) 8^{th} of October 2013

(d) 9^{th} of October 2013

FIGURE 6.10: Posterior probabilities $\mathcal{P}_{\mathcal{F}_t}(+|x)$ and $\mathcal{P}_{\mathcal{W}_t^D}(+|x)$ for different days. Feedback transactions are denoted with triangles and red color is used for frauds. In this example, feebacks are requested by $\mathcal{A}_t^W$ with α_t computed as in (6.7). Operating with different values of α_t corresponds to cutting the plane $(\mathcal{P}_{\mathcal{F}_t}, \mathcal{P}_{\mathcal{W}_t^D})$ using straight lines having as angular coefficient $\frac{-\alpha_t}{(1-\alpha_t)}$.

and q the number of features used to split each tree of the RF ($q \leq n$). The best case corresponds to the case when samples are always partitioned at the tree node into two balanced subsets of $\frac{N}{2}$ samples. The worst corresponds to the case of splits that create a subset with only one sample and the other subset with the remaining $N-1$ instances. The average corresponds to the average time complexity. In all our experiments only the term N_b is changing between the different classifiers, while $m = 100$ and $q = 7$.[9] In the

[9]In the `randomForest` package [222], the default value of q is obtained as $q = \sqrt{n}$.



(a) Training set size

(b) Training time

FIGURE 6.11: Training set size and time (in seconds) to train a RF for the feedback ($\mathcal{F}_t$) and delayed classifiers ($\mathcal{W}_t^D$ and $\mathcal{E}_t^D$) in the 2013 dataset. All the simulations were run using a single core in order to minimize the computing resources requested to the university cluster as in Section 5.1.3.

case of BRF, $N_b = 2N^+$ where N^+ is the number of frauds available for training (see Section 6.3.3).

Finally, in Figure 6.12 we plot a measure of feature relevance extracted from the BRF model of $\mathcal{W}_t^D$. The most informative feature is RISK_TERM_MIDUID which measures the risk associated to a terminal identifier (MIDUID). Within the top 10 most informative features we see variables measuring the amount of the transaction (e.g. SUM_AMT_HIS) and features measuring the risk associated to the country and continent of the terminal (e.g. RISK_TERM_COUNTRY). All variables with the name starting with RISK are features that were originally provided as categorical variables and that have been converted into numerical one by means of the transform presented in Section 5.1.3.

6.6 Discussion

Let us now discuss the accuracy improvements achieved by classifiers $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$ proposed in Section 6.3.2. First of all, we notice that the classifier learned on recent feedbacks is more accurate that those trained on delayed samples. This is made explicit by Table 6.3 showing that $\mathcal{F}_t$ often outperforms $\mathcal{W}_t^D$ (and $\mathcal{E}_t^D$), and $\mathcal{W}_t$ (and $\mathcal{E}_t$). We deem that $\mathcal{F}_t$ outperforms $\mathcal{W}_t^D$ (resp. $\mathcal{E}_t^D$) since $\mathcal{W}_t^D$ (resp. $\mathcal{E}_t^D$) are trained on less recent supervised couples. As far as the improvement with respect to $\mathcal{W}_t$ (and $\mathcal{E}_t$) is concerned, our interpretation is that this is due to the fact that $\mathcal{W}_t$ (and $\mathcal{E}_t$) are trained on the entire supervised dataset, then weakening the specific contribution of feedbacks.

Our results instead show that aggregation prevents the large amount of delayed supervised samples to dominate the small set of immediate feedbacks. This boils down to assign larger weights to the most recent than to the old samples, which is a golden rule

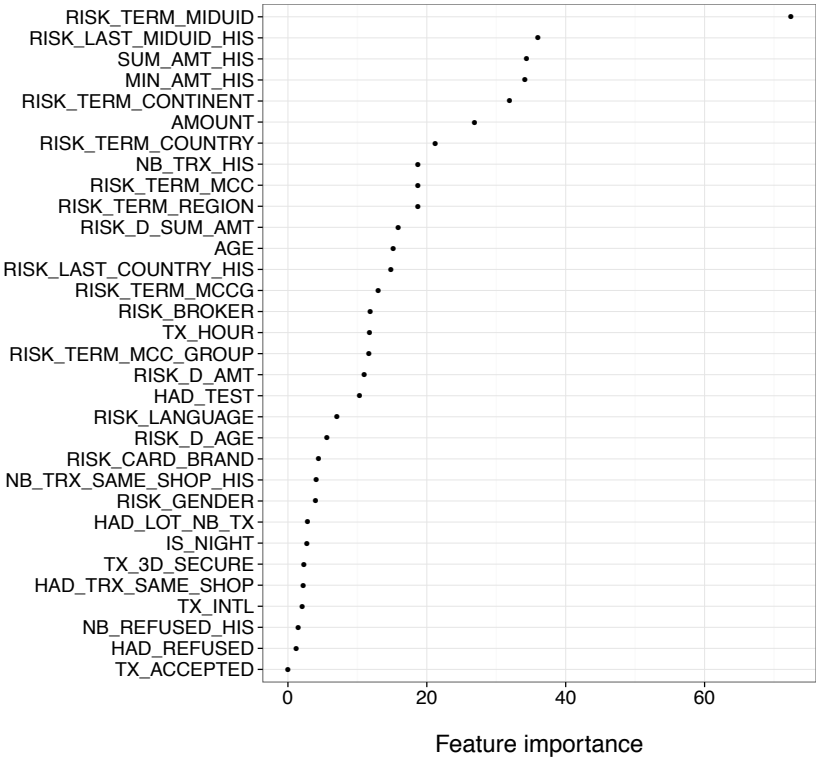


FIGURE 6.12: Average feature importance measured by the mean decrease in accuracy calculated with the Gini index in the RF model of $\mathcal{W}_t^D$ in the 2013 dataset. The `randomForest` package [222] available in R use the Gini index as splitting criterion. As stated in the package documentation: "The mean decrease in accuracy measure is computed from permuting the Out Of Bag (OOB) data: For each tree, the error rate on the OOB portion of the data is recorded. Then the same is done after permuting each predictor variable. The difference between the two are then averaged over all trees, and normalized by the standard deviation of the differences".

when learning in non-stationary environments. The aggregation $\mathcal{A}_t^W$ with $\alpha_t = 0.5$ is indeed an effective way to attribute higher importance to the information included in the feedbacks. At the same time $\mathcal{A}_t^E$ with $\alpha_t = 0.5$ is a way to balance the contribution of $\mathcal{F}_t$ and the remaining M models of $\mathcal{E}_t$.

Another motivation of the accuracy improvement with the aggregations is that classifiers trained on feedbacks and delayed samples address two different classification tasks (see Section 6.3). In particular $\mathcal{F}_t$ learns to discriminate those samples that have high risk of being frauds, i.e. those transactions yielding a large value of the posterior probability of the classifier used to generate alerts. On the contrary, classifiers trained on delayed samples rely on heterogeneous samples that were not selected depending on the posterior of the classifier, and includes both frauds and genuine transactions that are not considered risky. For this reason too, it is not convenient to pool the two types of supervised samples together. Finally, we notice that a constant and equal weight ($\alpha_t = 0.5$) in the aggregations $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$ is often performing as well as adaptive weighting strategies presented in Section 6.5.5.

The interaction between the FDSs (raising alerts) and the investigators (providing true labels) recalls solutions where the classifier interact with an oracle for additional labels typical of Active Learning [246]. The goal of Active Learning is to minimize the request of labeled data asked to oracles, by determining which are the most informative instances to query. Unfortunately, in a FDS, we have a small budget devoted to fraud alert investigation, i.e. the money that companies dedicate to fraud investigations is limited. The few investigators available have to focus on the most suspicious transactions with the goal of detecting the largest number of frauds. We cannot demand the investigators to check genuine transactions for the sake of obtaining informative patterns. Validation of transactions with low risk would come at the cost of not controlling highly risky transactions with a consequent impact on the detection accuracy. In the ideal case of perfect detection all transactions labeled by investigators should be of class fraud.

In our formulation (Section 6.2) we select transactions to alert using the probability of the samples to be fraudulent. Alternatively, Baesens et al. [10] recommend generating alerts for transactions with *expected fraud loss* (fraud probability $\times$ transaction amount) greater than a certain threshold. Similarly, Fan et al. [126] suggest to alert only transactions having the expected fraud loss higher than the overheads (cost of reviewing an alert, i.e. cost of investigation). In this work we define the alerts without considering the transactions amount, because we want to give equal importance to frauds of small and large amount. If we detect frauds of small amounts, we can block the card and prevent larger ones before they occur, because fraudsters typically try to steal money with small amounts first and then, if successful, with larger ones.

It is worth to remark that the formulation proposed in Section 6.2 is still a simplified description of the processes regulating companies analyzing credit cards transactions. For instance, it is typically not possible to extract the alerts A_t by ranking the whole set B_t, since transactions have to be immediately passed to investigators; similarly, delayed supervised couples $\mathrm{D}_{t-\delta}$ do not come all at once, but are provided over time. Notwithstanding, we deem that the most important aspects of the problem (i.e. the Alert-Feedback Interaction and the time-varying nature of the stream) are already contained in our formulation and that further details would unnecessarily make the problem setting complex.

A limitation of the current study is that we report as feedbacks only those transactions generating an alert. However, when the investigators call a cardholder, they typically check the status of also previous transactions made by the same card. This means that one alert can generate multiple supervised transactions ($|\mathrm{F}_t| \geq |\mathrm{A}_t|$). In this setting, it is more interesting measuring alert precision at the card level instead of the transaction level, i.e. measuring how many fraudulent cards are detected in the k cards that

investigators are able to check. Preliminary results, not included in this chapter, seem to confirm that, also in terms of alert precision measured at the card level, the best performances are obtained by aggregating two classifiers one trained on feedbacks and the other on delayed samples.

6.7 Conclusion

In this chapter we formalize a framework that reproduces the working conditions of real-world FDSs. In a real-world fraud-detection scenario, the only recent supervised-information is provided on the basis of the alerts generated by the FDS and feedbacks provided by investigators, i.e. Alert-Feedback Interaction (AFI). All the other supervised samples are available with a much larger delay.

Our intuition is that: i) AFI has to be explicitly considered in order to improve alert precision and ii) feedbacks and delayed samples have to be separately handled when training a realistic FDS. To this purpose, we have considered two general approaches for fraud detection: a sliding window and an ensemble of classifiers. We have then compared FDSs that separately learn on feedbacks and delayed samples against FDSs that pool all the available supervised information together. Experiments run on real-world streams of transactions show that the former strategy provides much more precise alerts than the latter, and that it also adapts more promptly in concept-drifting environments.

The majority of the works presented in the literature (e.g. [18, 61]) assume that we have the labels all transactions (ignoring AFI) and use AUC as accuracy measure. We claim that in a real world scenario, the main goal of a FDS is to return accurate alerts, i.e. high P_k (precision within the k transactions that we report as alert to the investigators). In this chapter we have showed that in order to get precise alerts (high P_k) it is mandatory to give large importance to feedbacks samples. Strategies lowering their influence in the learning process (e.g. SSB correction techniques and $\mathcal{W}_t$ or $\mathcal{E}_t$ classifiers) are often returning less precise alerts (lower P_k).

The feedback classifier $\mathcal{F}_t$ provides accurate ranking of the most suspicious transactions (high P_k), but it is not the best option when the goal is to achieve a good ranking of all transactions (low AUC). Classifier trained on everyday transactions (e.g. $\mathcal{W}_t^D$, $\mathcal{E}_t^D$, $\mathcal{R}_t$) have lower P_k, but return a better global ranking (higher AUC). By increasing δ, the number of days in which we receive feedbacks, $\mathcal{F}_t$ has better performances in terms of P_k. When $\mathcal{F}_t$ has higher P_k also its aggregation $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$ have higher accuracy.

Adaptation techniques are important in the presence of CD, this is made clear by the poor performances of the static classifier $\mathcal{S}_t$ which use the same model along the stream.

Aggregating $\mathcal{F}_t$ and $\mathcal{W}_t^D$ (or $\mathcal{E}_t^D$) with $\mathcal{A}_t^W$ (or $\mathcal{A}_t^E$) is often the best solution to achieve high P_k even in the presence of abrupt CD (see results from Section 6.5.2). We have also tested several methods to compute the weight α_t used in the aggregations, but it appears that a simple average ($\alpha_t = 0.5$) is often the best solution. In the future works we want to study non-linear aggregation of classifiers used in $\mathcal{A}_t^W$ and $\mathcal{A}_t^E$, perhaps better performances can be achieve by using a ranking algorithm receiving as input the posterior probabilities and the class of the transaction.

Currently, unsupervised (non-feedback) transactions between t and $t-\delta$ are not used in the learning process. We think that is worth testing semi-supervised learning algorithms to exploit both supervised (feedbacks) and unsupervised samples occurring between t and $t-\delta$. Another direction worth investigating is to use more than k transactions for training $\mathcal{F}_t$ if the ranking is accurate (when $\mathrm{P}_k \approx 1$). When $\mathcal{F}_t$ has high P_k we expect that the ranking produced by the posterior probability $\mathcal{P}_{\mathcal{F}_t}$ is accurate not only in the top k transactions, but also let's say in the top $k+\gamma$. In this case we could train $\mathcal{F}_t$ using $k+\gamma$ samples, where transactions ranked between k and $k+\gamma$ are labeled as fraudulent.

Finally, we can say that the results presented in this chapter are in line with the performance of our industrial partner (sometimes even better). However, for confidentiality reason, we are not allowed to disclose figures regarding the performance of our partner. Appendix B presents the software modules of the proposed FDS.

Chapter 7

Conclusions and Future Perspectives

Fraud detection is a particularly challenging and complex task. Fraudulent activities are rare events that are hard to model and in constant evolution. The large volume of transactions happening everyday demands automatic tools to support investigation, and the human resources devoted to investigations have to concentrate on the most suspicious cases. This thesis investigated how machine learning algorithms could be used to address some of these issues. In particular, we focused on the design of a framework that is able to report the riskiest transactions to investigators by means of algorithms that can deal with unbalanced and evolving data streams. This chapter summarizes the main results of the thesis, discusses open issues and presents future research directions.

7.1 Summary of contributions

A standard solution to deal with classification problems having unbalanced class distribution (like fraud detection) is to rebalance the classes before training a model. A popular rebalancing technique in the machine learning community is undersampling. In Chapter 4 we show the conditions under which undersampling is expected to improve the ranking of fraudulent and genuine transactions given by the posterior probability of a classifier. We also study the effect of class-separability on probability calibration and how to set a threshold to make predictions. It emerges that, without control on the data distribution, it is not possible to know beforehand whether undersampling is beneficial. For this reason in Section 4.3 we propose a racing algorithm to select rapidly the best sampling technique when multiple solutions are available. The racing algorithm was made available open source as a software package for the R language [25] (see package `unbalanced` [24] presented in Appendix A).

In order to deal with the non-stationarity distribution of credit cards transactions, Chapter 5 investigates multiple strategies for concept drift adaptation in the presence of skewed distribution. We notice that updating regularly the FDS is a much better alternative than using the same model over all the year. When choosing the samples to train a classifier it is important to retain historical transactions as well as forget outdated samples for the model to be precise. Also, resampling methods (notably undersampling) significantly improve the performances of a FDS. Propagation of fraudulent transactions along the stream is another effective way to rebalance the class distribution. However, the latter solution leads to computational overheads and it can be avoided without loss of accuracy by adopting a FDS based on Hellinger Distance Decision Tree.

Typically, fraud alerts generated by a FDS are checked by human investigators that annotate alerted transactions as genuine or fraudulent. Feedbacks from investigators provide recent supervised samples that are highly informative. In Chapter 6 we present a prototype of a FDS that is able to include investigators' feedbacks in the learning process. We show that, for the FDS to produce accurate alerts, feedbacks have to receive larger weights than the other supervised samples available. Combining two classifiers, one trained on feedbacks and one trained on delayed samples, is often the best way to provide accurate alerts in the presence of concept drift.

With this thesis we also made available to the machine learning community a dataset containing observations of credit card transactions. This dataset has been used in [28] and it includes examples of fraudulent samples, information that is rarely available.[1]

7.2 Learned lessons

In the following we summarize what we have learned during this PhD project with the intent of providing the reader and practitioners with some take home messages:

- As shown in Chapter 4, rebalancing a training set with undersampling is not guaranteed to improve performances, several factors influence the final effectiveness of undersampling and most of them cannot be controlled (e.g. the variance of a classifier and the samples where the conditions (4.20) is satisfied).
- The optimal degree of sampling depends on the datasets considered. In Section 4.1 we show that the right amount of sampling with undersampling (defined by β) is dataset specific. When sampling is too aggressive we have less points for which undersampling is beneficial. Characteristics of the classification task, such as class

[1] Dataset available at `http://www.ulb.ac.be/di/map/adalpozz/data/creditcard.Rdata`.

separability and imbalance ratio, also influence the performance of the classifier and the effectiveness of sampling methods.

- The best technique for unbalanced classification does not exist and the best for a given dataset can vary with the algorithm and accuracy measure adopted (see Section 4.3). Adaptive selection strategies (e.g. F-Race [26]) can be effective to rapidly provide an answer on which technique to use.

- The data stream defined by credit card transactions has non-stationary distributions, i.e. change in fraudulent and genuine behavior severely affect the performances of a FDS. This is made clear by the poor performances of static approaches as shown in Section 5.1.3. Updating the FDS is often a good idea and retaining historical transactions can improve predictive accuracy.

- For a real-world FDS it is mandatory to produce precise alerts, i.e. provide an accurate ranking of transactions with the highest risk of being fraudulent. As explained in Section 6.2, investigators trust and follow the alerts of a Data Driven Model as long as it does not generate too many false alerts. Poor alert precision (low P_k) means also few recent fraudulent transactions and the FDS has to relay only on old fraudulent pattern for the detection.

- The best learning algorithm for a FDS depends on the accuracy measure that we want to maximize. As shown in Chapter 6, a classifier trained only on feedback samples ($\mathcal{F}_t$) can provide more precise alerts (higher P_k) than one trained on all recent transactions ($\mathcal{R}_t$). However, the latter becomes a much better choice when using standard classification measures such as AUC.

7.3 Open issues

In this section we want to discuss some open issues in fraud detection that we believe are worth investigating, such as: i) defining a good performance measure, ii) modeling Alert-Feedback Interaction and iii) using the supervised and unsupervised information available with AFI.

Despite everybody agreeing on the fact that missing a fraud is much worse than generating a false alert, there is no agreement on which is the best way to measure fraud detection performance. Indeed, the machine learning community has proposed several cost measures, some are transaction-dependent [35, 70, 71], others are class-dependent [2, 247]. Other works avoid using cost-based accuracy measure by making the implicit assumption that it is more important to provide correct predictions [18, 64].

We think that there is not a correct and wrong way to measure detection performances, companies have different ideas about what is the best figure of merit. However, if a cost-based measure is preferred then we recommend using *normalized cost* [63] instead of *savings* [70], because the second can be negative which is counter intuitive.[2] Alternatively one should use a benefit matrix as proposed by Elkan [35]. When using standard classification metrics, we suggest AUC estimation based on the Mann-Whitney (Wilcoxon) statistics [18] (see Section 2.2.3). In this thesis we deem that, from an investigator perspective, the most relevant measure is the alert precision, denoted as P_k, i.e. precision within the k most risky transaction [20].

Chapter 6 has proposed a framework that is able to exploit investigators' feedbacks to improve detection performance. In our formulation, feedbacks are available at the end of the day altogether, while in reality, the feedback mechanism is much more complex. Feedbacks are provided during the day as soon as investigators call the cardholder. Moreover, investigators check all previous transactions of one card once it is found to be victim of fraud. This means that we could have more the one feedback transaction per alert generated and the model trained on feedbacks should learn *online*, i.e. as soon as feedbacks are received. Additionally, feedbacks can be provided with a delay of more than 24 hours and historical transactions could receive their labels after months. Despite these practical constraints we deem that the framework proposed in Chapter 6 is able to meet essential working conditions of a real-world FDS presented in Section 2.2.1.

Improvements of the FDS presented in Chapter 6 could come from using all available transactions, not only supervised samples. For example, semi-supervised strategies could exploit also unlabeled data (non-feedback transactions in the first δ days) to improve the detection. Alternatively, even if investigators require the FDS to provide accurate alerts, we could use few alerts (e.g. 5%) to query unrisky, but interesting samples for the sake of obtaining more precise alerts. This last option would allow exploring a small part of unsupervised samples as in active learning.

7.4 The Future: going towards Big Data solutions

In this section we will discuss future research directions that in the meantime led to the definition of a new research project called *BruFence* sponsored by Innoviris.[3] BruFence is a three years joint project between three research groups of two universities and three companies based in Brussels, Belgium. The partners of the project are: Machine Learning

[2] In some cases, given the small number of frauds, the cost of predicting all transactions as genuine can be lower than the cost registered by a classification algorithm, leading to negative savings.

[3] Innoviris is the Brussels institute for the encouragement of scientific research and innovation.

Group from Université Libre de Bruxelles, Machine Learning Group from Université Catholique de Louvain, QualSec from Université Libre de Bruxelles, Worldline S.A., Steria and NViso.

The goal of the project is the design of mechanisms based on machine learning and big data mining techniques that allow to automatically detect attacks and fraudulent behaviors in large amounts of transactions. BruFence aims at developing a real-time framework that is able to compare in parallel a large number of alternative models in terms of nature (expert-based or data driven), features (e.g. history or customer related), data (e.g. supervised or unsupervised), predictive methods (e.g. decision trees, neural networks), scalability (e.g. conventional versus Map/Reduce implementation) and quality criteria (e.g. readability vs. accuracy). The framework is expected to be scalable with respect to the amount of data and resources available. The project will also investigate the exploitation of network data (social networks, communication networks, etc.) for fraud, privacy and security purposes. The use of network data is currently a highly studied field, subject of much recent work in various areas of science [10].

Currently the IT infrastructure used for the FDS is based on classical data warehouse architectures. These architectures are well designed for business reporting but not for applying analytics to big volume of data. In this context, it is infeasible to do analytics directly on the whole historical data set with standard machine learning algorithms. Though an easy solution consists in using only a portion of data for training the algorithms, this has detrimental effects on the resulting predictive accuracy.

Companies that want to stay at the cutting edge of security technology (like the sponsors of this project) are more and more interested to enter the big data paradigm. However, though the introduction of big data technologies (e.g. Hadoop[4], Spark[5]) in the everyday business process is claimed to be straightforward by many vendors, in practice it demands a major redesign of existing functionalities. This is particularly true for analytics and business intelligence applications, where the number of off-the-shelf solutions is still limited and the required data processing is not trivial. A major goal of the project will be to adapt and, when necessary, rewrite machine learning and adaptive functionalities to make them scalable for huge amounts of transactional and log data. In particular we will target problems characterized by large amounts of noise, large dimensionality, non-stationarity and demanding a rapid and accurate identification of threatening or fraudulent configurations.

As shown by Van Vlasselaer et al. [61], network data is a powerful source of information in order to improve the detection of fraudulent behaviors. The rationale is that network

[4] `http://hadoop.apache.org/`
[5] `http://spark.apache.org/`

connectivity provides information that can improve the accuracy of the prediction model. For example, it is well known that fraudulent activities are linked to each other. Knowing that a merchant is targeted by many fraudsters may provide useful information on the likelihood that a transaction on the same shop is fraudulent. The project will investigate, develop, and compare different predictive models in order to determine to which extent prediction accuracy can be improved by using internal and external network data. The project will also investigate and compare different predictive models (graph-based semi-supervised classification [248]) for private information discovery (information not publicly available), in order to determine to which extent, hidden information can be inferred from the network (e.g. age and sex of the cardholder if not available). Eventually, we will develop new measures identifying the most critical or vulnerable connected nodes whose removal results in splitting the network. This can be done with traffic information or without traffic data.

Credit card fraud detection has traditionally focused on looking for factors such as transaction amount, point of sales, location, etc. available inside the organization. As shown in Section 2.2.2, from these basic variables it is possible to compute new aggregate features to model the behavior of the cardholder. Typically, companies use a small sample of historical transactions for each cardholder to build account-level variables. Because it is computationally demanding to compute aggregates, these features are usually calculated offline and then added to the feature vector representing the transaction when it is authorized. Using a small part of the information available may translate into a loss of predictive accuracy. The introduction of big data technologies allows overcoming these issues, i.e. computing aggregates in real time and using a larger set of historical transactions.

The Big Data solution that we envisage will be able to process massive amounts of structured and unstructured data from a hybrid of sources as well. This will enable the exploitation of both existing in-house and public data (i.e. social media, websites, blogs). Models and algorithms will take advantage of these richer sources to build more accurate detection models. For example, social media can used to check whether a cardholder is traveling and validate a transaction from an unusual location.

The project will deliver an online learning framework that is able to process the data stream where the account-level information and the network of transactions are considered. The learning process will have to handle the unbalanced nature of the data in real time without having to store/recall previous transactions as in Dal Pozzolo et al. [19]. The algorithms will be implemented using scalable architecture that will allow the integration of existing and external sources of information.

7.5 Added value for the company

In this section we want to discuss how the results of *Doctiris* project (presented in Section 1.7) could be valorized by the industrial partner Worldline S.A.

The FDS in production at Worldline is currently adopting a static approach (see Section 5.1.2), i.e. a classification model updated once/twice a year. The results of Chapter 5 show to the company that there is a clear performance gain when the model is updated more frequently, e.g. once a week or every 15 days. The work done in Chapter 5 allows Worldline to assess the performance of several learning strategies in the presence of CD and to test different CD adaptation methods.

We also used different classification algorithms not yet explored by Worldline. In particular, Random Forest has emerged as the best algorithm in many simulations and now the company has it in production. During the project we also investigated new ways to create aggregated features in order to include user behavior at the transaction level (see the transformation proposed in Section 5.1.3).

The unbalanced problem had never been theoretically studied before in Worldline. The results of Chapter 4 showed that, in case of fraud detection, the performance of a classifier can be significantly improved when sampling methods are used to rebalance the two classes. Given the large imbalance ratio and number of transactions, undersampling should perhaps be favored w.r.t. oversampling techniques. At the same time it is important to calibrate the probability in order to provide an accurate ranking of the transactions after sampling the dataset (see Section 4.2). Additionally, the racing strategy proposed in Section 4.3 gives Worldline a new tool for selecting efficiently the unbalanced method that best fits the data.

Finally, in Chapter 6 we provide evidence that alerted transactions can be very informative for obtaining accurate FDS. Currently, the FDS in production at Worldline is not able to trace if an historical transaction has been checked in a fraud investigation or not. In this setting, it is not possible to distinguish between feedback and non-feedback samples. As shown in Section 6.5, higher performances can be achieved by combining classifiers separately trained on feedback and delayed transactions.

7.6 Concluding remarks

The *Doctiris* PhD project was an unique opportunity to work on real-world fraud detection data that, because of its high sensitivity, is scarcely available. Moreover, this type of data is very interesting because it combines several challenges such as class overlap, class

imbalance and mislabeled samples among others. The collaboration with the industrial partner was particularly fruitful because in the company we had a supervisor, Dr. Olivier Caelen, who carefully guided our work.

As shown in Chapter 6, in real working conditions we have constraints that define new challenges. For example, the limited number of transactions verified by investigators allows only few recent supervised samples, while in the literature most works assume to know the labels of all transactions. Also, the figure of merit that is interesting for a company may differ from standard accuracy measures.

Typically, companies are interested in more practical than theoretical results, e.g. "as long as the algorithm works well there is no need to question its design or implementation". On the contrary, the academic world is sometimes addressing complex problems that have few practical applications. We believe that both worlds, industry and university, should look at each other and exchange ideas in order to have a much larger impact on society. For all these reasons, we hope that the Doctiris initiative will be followed by many others and will pave the way of more collaborations between companies and universities in the Brussels region.

Appendix A

The unbalanced package

This appendix presents a new software package called `unbalanced` [24] available for the R language [25]. It implements some techniques for unbalanced classification tasks presented in Section 3.1.1 and provides a racing strategy [227] to adaptively select the best methods for a given dataset, classification algorithms and accuracy measure adopted.

A.1 Methods for unbalanced classification

The `unbalanced` package includes some of the most well-known sampling and distance-based methods for unbalanced classification task. Within the family of sampling methods, we have functions for random undersampling (*ubUnder*) and oversampling (*ubOver*) [91]. The package contains also a function called *ubSMOTE* that implements SMOTE [92]. Other distance-based methods available in `unbalanced` are OSS [100] (*ubOSS*), CNN [98] (*ubCNN*), ENN [101] (*ubENN*), NCL [89] (*ubNCL*) and Tomek Link [96] (*ubTomek*). All these methods can be called by a wrapper function *ubBalance* that allows testing all these strategies by simpling changing the argument *type*.

The package includes the *ubIonosphere* datasets, which is a modification of the Ionosphere dataset contained in `mlbench` package. It has only numerical input variables, i.e. the first two variables are removed. The *Class* variable, originally taking values *bad* and *good*, has been transformed into a factor where 1 denotes the minority (bad) and 0 the majority class (good). This variable is our target and it is in the last column of the dataset. In the following we will also called the minority class as positive and the majority as negative.

For example, let's apply oversampling to the Ionosphere dataset to have a balanced dataset.

```
library(unbalanced)
data(ubIonosphere)
n <- ncol(ubIonosphere)
output <- ubIonosphere[ ,n]
input <- ubIonosphere[ ,-n]

set.seed(1234)

# 1-option using ubOver function
data <- ubOver(X=input, Y=output, k=0)

# 2-option using ubBalance function
#data <- ubBalance(X=input, Y=output, type="ubOver", k=0)

#oversampled dataset
overData <- data.frame(data$X, Class=data$Y)
#check the frequency of the target variable after oversampling
summary(overData$Class)

##   0   1
## 225 225
```

In this case we replicate the minority class until we have as many positive as negative instances. Alternativelly, we can balance the dataset using undersampling (i.e. removing observations from the majority class):

```
# using ubUnder function
data <- ubUnder(X=input, Y=output, perc=50,  method="percPos")
#undersampled dataset
underData <- data.frame(data$X, Class=data$Y)
#check the frequency of the target variable after oversampling
summary(underData$Class)

##   0   1
## 126 126
```

Another well-know method for unbalanced distribution is SMOTE, which oversample the minority class by creating new synthetic observations. Let's compare the performances

of two `randomForest` classifiers, one trained on the original unbalanced dataset and a second trained on a dataset obtained after applying SMOTE.

```
set.seed(1234)

#keep half for training and half for testing
N <- nrow(ubIonosphere)
N.tr <- floor(0.5*N)
X.tr  <- input[1:N.tr, ]
Y.tr <- output[1:N.tr]
X.ts <- input[(N.tr+1):N, ]
Y.ts <- output[(N.tr+1):N]

#use the original unbalanced training set to build a model
unbalTrain <- data.frame(X.tr, Class=Y.tr)
library(randomForest)
model1 <- randomForest(Class ~ ., unbalTrain)

#predict on the testing set
preds <- predict(model1, X.ts, type="class")
confusionMatrix1 <- table(prediction=preds, actual=Y.ts)
print(confusionMatrix1)

##           actual
## prediction   0   1
##          0 131   2
##          1   6  37

#rebalance the training set before building a model
#balanced <- ubBalance(X=X.tr, Y=Y.tr, type="ubOver", k=0)
balanced <- ubBalance(X=X.tr, Y=Y.tr, type="ubSMOTE", percOver=250)
balTrain <- data.frame(balanced$X, Class=balanced$Y)
#use the balanced training set
model2 <- randomForest(Class ~ ., balTrain)

#predict on the testing set
preds <- predict(model2, X.ts, type="class")
confusionMatrix2 <- table(prediction=preds, actual=Y.ts)
print(confusionMatrix2)
```

```
##           actual
## prediction   0   1
##          0 128   0
##          1   9  39

#we can now correctly classify more minority class instances
```

Using SMOTE we alter the original class distribution and we are able to increase the number of minority instances correctly classified. After smoting the dataset we have no false negatives, but a larger number of false positives. In unbalanced classification, it often desired to correctly classify all minority instances (reducing the number of false negatives), because the cost of missing a positive instances (a false negative) is much higher than the cost of missing a negative instance (a false positive).

A.2 Racing for strategy selection

The variety of approaches available in the `unbalanced` package allows the user to test multiple unbalanced methods. In a real situation where we have no prior information about the data distribution, it is difficult to decide which unbalanced strategy to use. In this case testing all alternatives is not an option either because of the associated computational cost.

As shown in Section 4.3, a possible solution comes from the adoption of the Racing algorithm. Racing is available in `unbalanced` with the *ubRacing* function and its implementation is a modification of the *race* function available in the `race` package. The function *ubRacing* compares the 8 unbalanced methods (*ubUnder*, *ubOver*, *ubSMOTE*, *ubOSS*, *ubCNN*, *ubENN*, *ubNCL*, *ubTomek*) against the unbalanced distribution, so we have 9 candidates starting the race. In the following we will use a highly unbalanced dataset containing credit card transactions used in [28].

```
set.seed(1234)

# load the dataset
load(url("http://www.ulb.ac.be/di/map/adalpozz/data/creditcard.Rdata"))

#configuration of the sampling method used in the race
ubConf <- list(type="ubUnder", percOver=200, percUnder=200,
               k=2, perc=50, method="percPos", w=NULL)
```

```
# Race with 10 trees in the Random Forest to speed up results
results <- ubRacing(Class ~., creditcard, "randomForest", positive=1,
                    metric="auc", ubConf=ubConf, ntree=10)

##  Racing for unbalanced methods selection in 10 fold CV
##  Number of candidates.....................................9
##  Max number of folds in the CV...........................10
##  Max number of experiments...............................100
##  Statistical test.............................Friedman test
##
##                              Markers:
##                                 x No test is performed.
##                                 - The test is performed and
##                                   some candidates are discarded.
##                                 = The test is performed but
##                                   no candidate is discarded.
##
##  +-+-----------+-----------+-----------+-----------+-----------+
##  | |       Fold|      Alive|       Best|  Mean best| Exp so far|
##  +-+-----------+-----------+-----------+-----------+-----------+
##  |x|          1|          9|          4|     0.9541|          9|
##  |=|          2|          9|          4|      0.954|         18|
##  |-|          3|          2|          4|     0.9591|         27|
##  |=|          4|          2|          4|      0.963|         29|
##  |=|          5|          2|          4|     0.9651|         31|
##  |-|          6|          1|          4|     0.9646|         33|
##  +-+-----------+-----------+-----------+-----------+-----------+
## Selected candidate: ubSMOTE   metric: auc   mean value: 0.9646
```

The best method according to the F-race is SMOTE. Please note that it is possible to change the type of statistical test used to remove candidates in the race with the argument *stat.test*. When we set *stat.test* = *"no"*, no statistical test is performed and the race terminates when all the folds of the cross validation are explored.

A.3 Summary

With the `unbalanced` package we have made available some of the most well-known methods for unbalanced distribution. All these methods can be called from *ubBalance* that is a wrapper to the method-specific functions. Depending on the type of dataset, classification algorithm and accuracy measure adopted, we may have different strategies that return the best accuracy.

This consideration has led us to adopt the racing strategy where different candidates (unbalanced methods) are tested simultaneously. This algorithm is implemented in the *ubRacing* function which selects the best candidate without having to explore the whole dataset.

Appendix B

FDS software modules

This appendix presents the software modules of the FDS prototype presented in Chapter 6. The software is divided in three main modules (see Figure B.1): i) Model training, ii) Scoring and iii) Review. The following sections present the role of each module. These modules are all implemented in the R language [25].

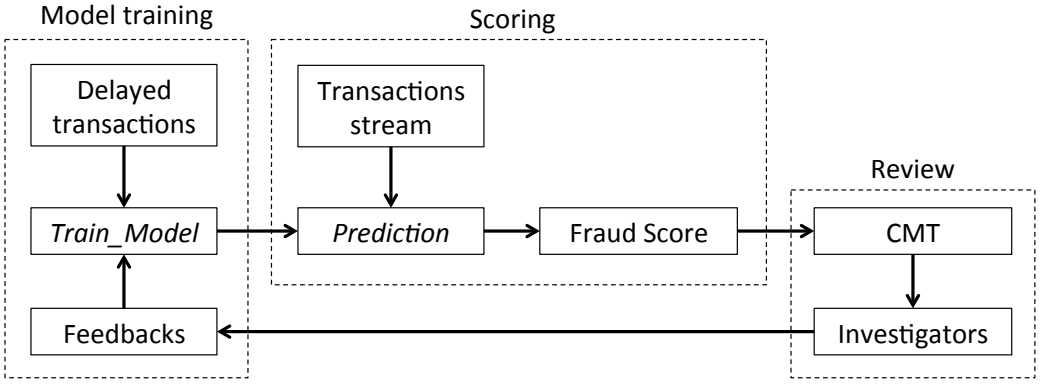


FIGURE B.1: Software modules of the FDS prototype presented in Chapter 6.

B.1 Model training

This module is used to train predictive models that are capable of estimating the probability of a transaction to be fraudulent. In particular, everyday it learns two models, one on delayed transactions $D_{t-\delta}$ and one on feedbacks F_t (see Section 6.3.2). The module implements two standard CD adaptation techniques (see Section 6.3.1): i) a sliding window classifier and ii) an ensemble of classifiers. Regardless of the CD adaptation strategy adopted, the module runs at midnight so that the resulting predictive models ($\mathcal{F}_t$ and $\mathcal{W}_t^D$ or $\mathcal{E}_t^D$) are used to score transactions occurring the next day $(t+1)$. The main function of this module is *Train_Model* which receives in input some supervised

transactions (feedbacks or delayed samples) and returns a predictive model built using the `randomForest` [222] package as explained in Section 6.3.3.

B.2 Scoring

In the scoring module all transactions arriving at day $t+1$ are scored by the *Prediction* function. In the *Prediction* function, each transaction is first scored by both the feedback classifier (with $\mathcal{P}_{\mathcal{F}_t}(+|x)$) and the delayed classifier (with $\mathcal{P}_{\mathcal{W}_t^D}(+|x)$ or $\mathcal{P}_{\mathcal{E}_t^D}(+|x)$). Then, the final fraud score is computed using (6.3) or (6.4) depending on the CD adaptation strategy used in module B.1. This module is the most critical because transactions arrive as a continuous stream and have to be scored in Near Real Time (see Section 2.2.1), i.e. the *Prediction* function receives in input transactions that have been authorized and has to score them within a little time span. Note that the feature vector of transactions entering this module contains already the aggregates features presented in Section 2.2.2. Aggregated feature are computed offline in a distinct module.

B.3 Review

In this module, transactions are made available to the investigators in a dashboard called Case Management Tool (CMT), where for each transaction they can see the fraud score returned by the *Prediction* function, and the feature vector containing the original variable such as $CARD\_ID$, the shop, currency, and also the aggregated variable. In the CMT transactions are sorted according to their fraud score so that investigators can review the most suspicious one at the time they access the CMT. After verification a transaction is flagged as checked and it becomes a feedback. Transactions that are not reviewed by the investigators remain unlabeled for δ days and then, if not reported as fraudulent by the cardholders, are labeled as genuine. Alter δ days, transactions that have not been checked are used as delayed supervised samples in module B.1.

Appendix C

Bias and Variance of an estimator

This appendix presents two measures to assess the quality of an estimator, namely Bias and Variance. Then we show a Bias–Variance decomposition for the mean squared error.

Definition 1 (Bias of an estimator)**.** *An estimator $\hat{\boldsymbol{\theta}}$ of θ has bias*

$$Bias[\hat{\boldsymbol{\theta}}] = E[\hat{\boldsymbol{\theta}}] - \theta$$

In particular, $\hat{\boldsymbol{\theta}}$ is said to be unbiased *if $E[\hat{\boldsymbol{\theta}}] = \theta$ and* biased *otherwise.*

Definition 2 (Variance of an estimator)**.** *The variance of an estimator $\hat{\boldsymbol{\theta}}$ is the variance of its sampling distribution*

$$Var[\hat{\boldsymbol{\theta}}] = E[(\hat{\boldsymbol{\theta}} - E[\hat{\boldsymbol{\theta}}])^2]$$

For any random variable $\mathbf{x}$ we can write

$$Var[\mathbf{x}] = E[\mathbf{x}^2] - E[\mathbf{x}]^2 \Longleftrightarrow E[\mathbf{x}^2] = Var[\mathbf{x}] + E[\mathbf{x}]^2 \tag{C.1}$$

Since θ is deterministic $E[\theta] = \theta$ and $Var[\theta] = 0$. Therefore, using (C.1) we can decompose the Mean Squared Error (MSE) in terms of Bias and Variance of $\hat{\boldsymbol{\theta}}$:

$$\begin{aligned}
\text{MSE} = E\big[(\theta - \hat{\boldsymbol{\theta}})^2\big] &= E[\theta^2 + \hat{\boldsymbol{\theta}}^2 - 2\theta\hat{\boldsymbol{\theta}}] && \text{(C.2)}\\
&= E[\theta^2] + E[\hat{\boldsymbol{\theta}}^2] - E[2\theta\hat{\boldsymbol{\theta}}] && \text{(C.3)}\\
&= Var[\theta] + E[\theta]^2 + Var[\hat{\boldsymbol{\theta}}] + E[\hat{\boldsymbol{\theta}}]^2 - 2\theta E[\hat{\boldsymbol{\theta}}] && \text{(C.4)}\\
&= 0 + Var[\hat{\boldsymbol{\theta}}] + (\theta - E[\hat{\boldsymbol{\theta}}])^2 && \text{(C.5)}\\
&= Var[\hat{\boldsymbol{\theta}}] + E[\theta - \hat{\boldsymbol{\theta}}]^2 && \text{(C.6)}\\
&= Var[\hat{\boldsymbol{\theta}}] + Bias[\hat{\boldsymbol{\theta}}]^2 && \text{(C.7)}
\end{aligned}$$

Bibliography

[1] D.J. Newman A. Asuncion. UCI machine learning repository, 2007. URL `http://archive.ics.uci.edu/ml/`.

[2] R.J. Bolton and D.J. Hand. Statistical fraud detection: A review. *Statistical Science*, pages 235–249, 2002.

[3] Piotr Juszczak, Niall M Adams, David J Hand, Christopher Whitrow, and David J Weston. Off-the-peg and bespoke classifiers for fraud detection. *Computational Statistics & Data Analysis*, 52(9):4521–4532, 2008.

[4] Sam Maes, Karl Tuyls, Bram Vanschoenwinkel, and Bernard Manderick. Credit card fraud detection using bayesian and neural networks. In *Proceedings of the 1st international naiso congress on neuro fuzzy technologies*, 2002.

[5] Jon TS Quah and M Sriganesh. Real-time credit card fraud detection using computational intelligence. *Expert Systems with Applications*, 35(4):1721–1732, 2008.

[6] Tej Paul Bhatla, Vikram Prabhu, and Amit Dua. Understanding credit card frauds. *Cards business review*, 1(6), 2003.

[7] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[8] Linda Delamaire, HAH Abdou, and John Pointon. Credit card fraud and detection techniques: a review. *Banks and Bank Systems*, 4(2):57–68, 2009.

[9] Raymond Anderson. *The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management and Decision Automation.* Oxford University Press, 2007.

[10] Bart Baesens, Veronique Van Vlasselaer, and Wouter Verbeke. *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques: A Guide to Data Science for Fraud Detection.* John Wiley & Sons, 2015.

[11] Richard J Bolton and David J Hand. Unsupervised profiling methods for fraud detection. *Credit Scoring and Credit Control VII*, pages 235–255, 2001.

[12] The nilson report, August 2013. URL `http://www.nilsonreport.com/`. [Issue 1023 | Aug 2013].

[13] Lexis Nexis. True cost of fraud 2014 study. URL `http://www.lexisnexis.com/risk/insights/true-cost-fraud.aspx`.

[14] European Central Bank. Report on card fraud available: `https://www.ecb.europa.eu/press/pr/date/2014/html/pr140225.en.html`.

[15] Cybersource. 2015 uk fraud report series: Part 2, 2015. URL `http://www.cybersource.com/`. [Online; accessed July-2015].

[16] Wikipedia. 3-d secure, 2015. URL `http://en.wikipedia.org/wiki/3-D_Secure`. [Online; accessed July-2015].

[17] Jose M Pavía, Ernesto J Veres-Ferrer, and Gabriel Foix-Escura. Credit card incidents and control systems. *International Journal of Information Management*, 32 (6):501–503, 2012.

[18] Andrea Dal Pozzolo, Olivier Caelen, Yann-Ael Le Borgne, Serge Waterschoot, and Gianluca Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications*, 41(10):4915–4928, 2014.

[19] Andrea Dal Pozzolo, Reid A. Johnson, Olivier Caelen, Serge Waterschoot, Nitesh V Chawla, and Gianluca Bontempi. Using hddt to avoid instances propagation in unbalanced and evolving data streams. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 588–594. IEEE, 2014.

[20] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection and concept-drift adaptation with delayed supervised information. In *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 2015.

[21] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, 2009.

[22] G. Batista, A. Carvalho, and M. Monard. Applying one-sided selection to unbalanced datasets. *MICAI 2000: Advances in Artificial Intelligence*, pages 315–325, 2000.

[23] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

[24] Andrea Dal Pozzolo, Olivier Caelen, and Gianluca Bontempi. *unbalanced: Racing For Unbalanced Methods Selection.*, 2015. URL `http://CRAN.R-project.org/package=unbalanced`. R package version 2.0.

[25] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL `http://www.R-project.org/`. ISBN 3-900051-07-0.

[26] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the genetic and evolutionary computation conference*, pages 11–18, 2002.

[27] Andrea Dal Pozzolo, Olivier Caelen, Serge Waterschoot, and Gianluca Bontempi. Racing for unbalanced methods selection. In *Proceedings of the 14th International Conference on Intelligent Data Engineering and Automated Learning*. IDEAL, 2013.

[28] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2015.

[29] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 1997.

[30] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[31] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[32] Lennart Ljung. System identification: Theory for the user. *PTR Prentice Hall Information and System Sciences Series*, 198, 1987.

[33] Andrew R Webb. *Statistical pattern recognition*. John Wiley & Sons, 2003.

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

[35] C. Elkan. The foundations of cost-sensitive learning. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 973–978, 2001.

[36] Trevor. Hastie, Robert. Tibshirani, and J Jerome H Friedman. *The elements of statistical learning*, volume 1. Springer New York, 2001.

[37] Pedro Domingos. A unified bias-variance decomposition for zero-one and squared loss. *AAAI/IAAI*, 2000:564–569, 2000.

[38] Robert C Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[39] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130, 1997.

[40] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1-2):105–139, 1999.

[41] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[42] Robert E Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of statistics*, pages 1651–1686, 1998.

[43] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[44] Anders Krogh, Jesper Vedelsby, et al. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, 7:231–238, 1995.

[45] Hongyu Guo and Herna L Viktor. Learning from imbalanced data sets with boosting and data generation: the databoost-im approach. *ACM SIGKDD Explorations Newsletter*, 6(1):30–39, 2004.

[46] Gary M Weiss. Mining with rarity: a unifying framework. *ACM SIGKDD Explorations Newsletter*, 6(1):7–19, 2004.

[47] David L Olson and Dursun Delen. *Advanced data mining techniques*. Springer Science & Business Media, 2008.

[48] I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[49] Foster Provost. Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 Workshop on Imbalanced Data Sets*, 2000.

[50] Chao Chen, Andy Liaw, and Leo Breiman. Using random forest to learn imbalanced data. *University of California, Berkeley*, 2004.

[51] Nitesh V Chawla, David A Cieslak, Lawrence O Hall, and Ajay Joshi. Automatically countering imbalance and its empirical relationship to cost. *Data Mining and Knowledge Discovery*, 17(2):225–252, 2008.

[52] Gregory Ditzler, Robi Polikar, and Nitesh Chawla. An incremental learning algorithm for non-stationary environments and class imbalance. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2997–3000. IEEE, 2010.

[53] Tom Fawcett. Roc graphs: Notes and practical considerations for researchers. *Machine learning*, 31:1–38, 2004.

[54] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8): 861–874, 2006.

[55] Nitesh V Chawla. C4.5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure. In *Proceedings of the ICML*, volume 3, 2003.

[56] Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 853–867. Springer, 2005.

[57] Wei Liu, Sanjay Chawla, David A Cieslak, and Nitesh V Chawla. A robust decision tree algorithm for imbalanced data sets. In *SDM*, volume 10, pages 766–777. SIAM, 2010.

[58] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[59] Ira Cohen and Moises Goldszmidt. Properties and benefits of calibrated classifiers. In *Knowledge Discovery in Databases: PKDD 2004*, pages 125–136. Springer, 2004.

[60] Glenn W Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950.

[61] Véronique Van Vlasselaer, Cristian Bravo, Olivier Caelen, Tina Eliassi-Rad, Leman Akoglu, Monique Snoeck, and Bart Baesens. Apate: A novel approach for automated credit card transaction fraud detection using network-based extensions. *Decision Support Systems*, 2015.

[62] M Krivko. A hybrid model for plastic card fraud detection systems. *Expert Systems with Applications*, 37(8):6070–6076, 2010.

[63] Christopher Whitrow, David J Hand, Piotr Juszczak, D Weston, and Niall M Adams. Transaction aggregation as a strategy for credit card fraud detection. *Data Mining and Knowledge Discovery*, 18(1):30–55, 2009.

[64] Siddhartha Bhattacharyya, Sanjeev Jha, Kurian Tharakunnel, and J Christopher Westland. Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3):602–613, 2011.

[65] Sanjeev Jha, Montserrat Guillen, and J Christopher Westland. Employing transaction aggregation strategy to detect credit card fraud. *Expert systems with applications*, 39(16):12650–12657, 2012.

[66] D.J. Hand. Measuring classifier performance: a coherent alternative to the area under the roc curve. *Machine learning*, 77(1):103–123, 2009.

[67] Donald Bamber. The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *Journal of mathematical psychology*, 12(4):387–415, 1975.

[68] Yusuf Sahin, Serol Bulkan, and Ekrem Duman. A cost-sensitive decision tree approach for fraud detection. *Expert Systems with Applications*, 40(15):5916–5923, 2013.

[69] Nader Mahmoudi and Ekrem Duman. Detecting credit card fraud by modified fisher discriminant analysis. *Expert Systems with Applications*, 42(5):2510–2516, 2015.

[70] Alejandro Correa Bahnsen, Djamila Aouada, and Björn Ottersten. Example-dependent cost-sensitive decision trees. *Expert Systems with Applications*, 2015.

[71] Alejandro Correa Bahnsen, Aleksandar Stojanovic, Djamila Aouada, and Bjorn Ottersten. Cost sensitive credit card fraud detection using bayes minimum risk. In *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, volume 1, pages 333–338. IEEE, 2013.

[72] Ekrem Duman and M Hamdi Ozcelik. Detecting credit card fraud by genetic algorithm and scatter search. *Expert Systems with Applications*, 38(10):13057–13063, 2011.

[73] G. Fan and M. Zhu. Detection of rare items with target. *Statistics and Its Interface*, 4:11–17, 2011.

[74] Mu Zhu. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2, 2004.

[75] David J Hand and Martin J Crowder. Overcoming selectivity bias in evaluating new fraud detection systems for revolving credit operations. *International Journal of Forecasting*, 28(1):216–223, 2012.

[76] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1):1–6, 2004.

[77] B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Data Mining, ICDM*, pages 435–442. IEEE, 2003.

[78] P. Domingos. Metacost: a general method for making classifiers cost-sensitive. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 155–164. ACM, 1999.

[79] Nathalie Japkowicz. Concept-learning in the presence of between-class and within-class imbalances. In *Advances in Artificial Intelligence*, pages 67–77. Springer, 2001.

[80] Taeho Jo and Nathalie Japkowicz. Class imbalances versus small disjuncts. *ACM SIGKDD Explorations Newsletter*, 6(1):40–49, 2004.

[81] Gary M Weiss. Learning with rare cases and small disjuncts. In *ICML*, pages 558–565, 1995.

[82] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[83] Nathalie Japkowicz. Class imbalances: are we focusing on the right issue. In *Workshop on Learning from Imbalanced Data Sets II*, volume 1723, page 63, 2003.

[84] Ronaldo C Prati, Gustavo EAPA Batista, and Maria Carolina Monard. Class imbalances versus class overlapping: an analysis of a learning system behavior. In *MICAI 2004: Advances in Artificial Intelligence*, pages 312–321. Springer, 2004.

[85] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[86] Robert C Holte, Liane E Acker, and Bruce W Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, volume 1, 1989.

[87] Haibo He and Yunqian Ma. *Imbalanced learning: foundations, algorithms, and applications*. John Wiley & Sons, 2013.

[88] Gary M Weiss and Foster Provost. The effect of class distribution on classifier learning: an empirical study. *Rutgers Univ*, 2001.

[89] J. Laurikkala. Improving identification of difficult small classes by balancing class distribution. *Artificial Intelligence in Medicine*, pages 63–66, 2001.

[90] Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence*, 20(1): 18–36, 2004.

[91] C. Drummond and R.C. Holte. C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on Learning from Imbalanced Datasets II*, 2003.

[92] NV Chawla, KW Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research (JAIR)*, 16:321–357, 2002.

[93] BX Wang and N Japkowicz. Imbalanced data set learning with synthetic samples. In *Proc. IRIS Machine Learning Workshop*, page 19, 2004.

[94] 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*, pages 878–887. Springer, 2005.

[95] Haibo He, Yang Bai, Edwardo Garcia, Shutao Li, et al. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 1322–1328. IEEE, 2008.

[96] I. Tomek. Two modifications of cnn. *IEEE Trans. Syst. Man Cybern.*, 6:769–772, 1976.

[97] S. Suman, K. Laddhad, and U. Deshmukh. Methods for handling highly skewed datasets. *Part I-October*, 3, 2005.

[98] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 1968.

[99] D.R. Wilson and T.R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286, 2000.

[100] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *ICML*, volume 97, pages 179–186. Nashville, USA, 1997.

[101] D.L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics*, (3):408–421, 1972.

[102] J.R. Quinlan. *C4.5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[103] David A Cieslak and Nitesh V Chawla. Learning decision trees for unbalanced data. In *Machine Learning and Knowledge Discovery in Databases*, pages 241–256. Springer, 2008.

[104] Sofia Visa and Anca Ralescu. Issues in mining imbalanced data sets-a review paper. In *Proceedings of the sixteen midwest artificial intelligence and cognitive science conference*, pages 67–73. sn, 2005.

[105] Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of Workshop on Learning from Imbalanced Datasets*, 2003.

[106] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20–29, 2004.

[107] Rong Yan, Yan Liu, Rong Jin, and Alex Hauptmann. On predicting rare classes with svm ensembles in scene classification. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 3, pages III–21. IEEE, 2003.

[108] Gang Wu and Edward Y Chang. Class-boundary alignment for imbalanced dataset learning. In *ICML 2003 workshop on learning from imbalanced data sets II, Washington, DC*, pages 49–56, 2003.

[109] Jérôme Callut and Pierre Dupont. F β support vector machines. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 3, pages 1443–1448. IEEE, 2005.

[110] Xuchun Li, Lei Wang, and Eric Sung. Adaboost with svm-based component classifiers. *Engineering Applications of Artificial Intelligence*, 21(5):785–795, 2008.

[111] Wei Liu and Sanjay Chawla. Class confidence weighted knn algorithms for imbalanced data sets. In *Advances in Knowledge Discovery and Data Mining*, pages 345–356. Springer, 2011.

[112] Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple minimum supports. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 337–341. ACM, 1999.

[113] Florian Verhein and Sanjay Chawla. Using significant, positively associated and relatively class correlated rules for associative classification of imbalanced datasets. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 679–684. IEEE, 2007.

[114] Gary M Weiss. Foundations of imbalanced learning. *H. He, & Y. Ma, Imbalanced Learning: Foundations, Algorithms, and Applications*, pages 13–41, 2013.

[115] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[116] X.Y. Liu, J. Wu, and Z.H. Zhou. Exploratory undersampling for class-imbalance learning. *Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(2):539–550, 2009.

[117] Shuo Wang, Ke Tang, and Xin Yao. Diversity exploration and negative correlation learning on imbalanced data sets. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 3259–3266. IEEE, 2009.

[118] Fernando Vilariño, Panagiota Spyridonos, Jordi Vitrià, and Petia Radeva. Experiments with svm and stratified sampling with an imbalanced problem: Detection of intestinal contractions. In *Pattern Recognition and Image Analysis*, pages 783–791. Springer, 2005.

[119] Pilsung Kang and Sungzoon Cho. Eus svms: Ensemble of under-sampled svms for data imbalance problems. In *Neural Information Processing*, pages 837–846. Springer, 2006.

[120] Yang Liu, Aijun An, and Xiangji Huang. Boosting prediction accuracy on imbalanced datasets with svm ensembles. In *Advances in Knowledge Discovery and Data Mining*, pages 107–118. Springer, 2006.

[121] Benjamin X Wang and Nathalie Japkowicz. Boosting support vector machines for imbalanced data sets. *Knowledge and Information Systems*, 25(1):1–20, 2010.

[122] N. Chawla, A. Lazarevic, L. Hall, and K. Bowyer. Smoteboost: Improving prediction of the minority class in boosting. *Knowledge Discovery in Databases: PKDD 2003*, pages 107–119, 2003.

[123] Mahesh V Joshi, Vipin Kumar, and Ramesh C Agarwal. Evaluating boosting algorithms to classify rare classes: Comparison and improvements. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 257–264. IEEE, 2001.

[124] David Mease, Abraham J Wyner, and Andreas Buja. Boosted classification trees and class probability/quantile estimation. *The Journal of Machine Learning Research*, 8:409–439, 2007.

[125] Yanmin Sun, Mohamed S Kamel, Andrew KC Wong, and Yang Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40 (12):3358–3378, 2007.

[126] Wei Fan, Salvatore J Stolfo, Junxin Zhang, and Philip K Chan. Adacost: misclassification cost-sensitive boosting. In *ICML*, pages 97–105, 1999.

[127] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Risk minimization, probability elicitation, and cost-sensitive svms. In *ICML*, pages 759–766, 2010.

[128] Matjaz Kukar, Igor Kononenko, et al. Cost-sensitive learning with neural networks. In *ECAI*, pages 445–449, 1998.

[129] Charles X Ling, Qiang Yang, Jianning Wang, and Shichao Zhang. Decision trees with minimal costs. In *Proceedings of the twenty-first international conference on Machine learning*, page 69. ACM, 2004.

[130] Jeffrey P Bradford, Clayton Kunz, Ron Kohavi, Cliff Brunk, and Carla E Brodley. Pruning decision trees with misclassification costs. In *Machine Learning: ECML-98*, pages 131–136. Springer, 1998.

[131] C.X. Ling and V.S. Sheng. Cost-sensitive learning and the class imbalance problem. *Encyclopedia of Machine Learning*, 2008.

[132] Marcus A Maloof, Pat Langley, Stephanie Sage, and T Binford. *Learning to detect rooftops in aerial images*. 1997.

[133] Corinna Cortes, Mehryar Mohri, Michael Riley, and Afshin Rostamizadeh. Sample selection bias correction theory. In *Algorithmic learning theory*, pages 38–53. Springer, 2008.

[134] Marco Saerens, Patrice Latinne, and Christine Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure. *Neural computation*, 14(1):21–41, 2002.

[135] Mark G Kelly, David J Hand, and Niall M Adams. The impact of changing populations on classifier performance. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 367–371. ACM, 1999.

[136] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[137] David J Hand et al. Classifier technology and the illusion of progress. *Statistical science*, 21(1):1–14, 2006.

[138] Keisuke Yamazaki, Motoaki Kawanabe, Sumio Watanabe, Masashi Sugiyama, and Klaus-Robert Müller. Asymptotic bayesian generalization error when training and

test distributions are different. In *Proceedings of the 24th international conference on Machine learning*, pages 1079–1086. ACM, 2007.

[139] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. *The Journal of Machine Learning Research*, 8:985–1005, 2007.

[140] James J Heckman. Sample selection bias as a specification error. *Econometrica: Journal of the econometric society*, pages 153–161, 1979.

[141] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530, 2012.

[142] Bianca Zadrozny and Charles Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 204–213. ACM, 2001.

[143] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the twenty-first international conference on Machine learning*, page 114. ACM, 2004.

[144] Wei Fan, Ian Davidson, Bianca Zadrozny, and Philip S Yu. An improved categorization of classifier's sensitivity on sample selection bias. In *Data Mining, Fifth IEEE International Conference on*, pages 4–pp. IEEE, 2005.

[145] Miroslav Dudík, Steven J Phillips, and Robert E Schapire. Correcting sample selection bias in maximum entropy density estimation. In *Advances in neural information processing systems*, pages 323–330, 2005.

[146] Nitesh V Chawla and Grigoris I Karakoulas. Learning from labeled and unlabeled data: An empirical study across techniques and domains. *J. Artif. Intell. Res.(JAIR)*, 23:331–366, 2005.

[147] Masashi Sugiyama. Learning under non-stationarity: Covariate shift adaptation by importance weighting. In *Handbook of Computational Statistics*, pages 927–952. Springer, 2012.

[148] T Ryan Hoens, Robi Polikar, and Nitesh V Chawla. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, 1(1):89–101, 2012.

[149] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

[150] C. Alippi, G. Boracchi, and M. Roveri. Just-in-time classifiers for recurrent concepts. *Neural Networks and Learning Systems, IEEE Transactions on*, 24(4):620–634, April. ISSN 2162-237X. doi: 10.1109/TNNLS.2013.2239309.

[151] Stephen Grossberg. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural networks*, 1(1):17–61, 1988.

[152] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Advances in artificial intelligence–SBIA 2004*, pages 286–295. Springer, 2004.

[153] Kyosuke Nishida and Koichiro Yamauchi. Detecting concept drift using statistical testing. In *Discovery Science*, pages 264–269. Springer, 2007.

[154] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *SDM*, volume 7, page 2007. SIAM, 2007.

[155] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. A just-in-time adaptive classification system based on the intersection of confidence intervals rule. *Neural Networks*, 24(8):791–800, 2011.

[156] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in non-stationary environments: A survey. *Computational Intelligence Magazine, IEEE*, 10(4):12–25, 2015.

[157] Indrė Žliobaitė. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784*, 2010.

[158] Ryan Elwell and Robi Polikar. Incremental learning of concept drift in nonstationary environments. *Neural Networks, IEEE Transactions on*, 22(10):1517–1531, 2011.

[159] Robi Polikar, L Upda, SS Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 31(4):497–508, 2001.

[160] 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*, pages 377–382. ACM, 2001.

[161] Sheng Chen, Haibo He, Kang Li, and Sachi Desai. Musera: multiple selectively recursive approach towards imbalanced stream data mining. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE, 2010.

[162] J Zico Kolter and Marcus A Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, 8:2755–2790, 2007.

[163] Jeffrey C Schlimmer and Richard H Granger Jr. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.

[164] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[165] Ludmila I Kuncheva. Classifier ensembles for changing environments. In *Multiple classifier systems*, pages 1–15. Springer, 2004.

[166] Charu C Aggarwal. *Data streams: models and algorithms*, volume 31. Springer Science & Business Media, 2007.

[167] 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*, pages 71–80. ACM, 2000.

[168] 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*, pages 97–106. ACM, 2001.

[169] João Gama, Ricardo Fernandes, and Ricardo Rocha. Decision trees for mining data streams. *Intelligent Data Analysis*, 10(1):23–45, 2006.

[170] Jing Gao, Wei Fan, Jiawei Han, and S Yu Philip. A general framework for mining concept-drifting data streams with skewed distributions. In *SDM*, 2007.

[171] Jing Gao, Bolin Ding, Wei Fan, Jiawei Han, and Philip S Yu. Classifying data streams with skewed class distributions and concept drifts. *Internet Computing*, 12(6):37–49, 2008.

[172] Gregory Ditzler and Robi Polikar. An ensemble based incremental learning framework for concept drift and class imbalance. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE, 2010.

[173] Gregory Ditzler and Robi Polikar. Incremental learning of concept drift from streaming imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 25(10):2283–2301, 2013.

[174] Ryan Elwell and Robi Polikar. Incremental learning of variable rate concept drift. In *Multiple Classifier Systems*, pages 142–151. Springer, 2009.

[175] Ryan N Lichtenwalter and Nitesh V Chawla. Adaptive methods for classification in arbitrarily imbalanced and drifting data streams. In *New Frontiers in Applied Data Mining*, pages 53–75. Springer, 2010.

[176] Sheng Chen and Haibo He. Towards incremental learning of nonstationary imbalanced data stream: a multiple selectively recursive approach. *Evolving Systems*, 2 (1):35–50, 2011.

[177] Sheng Chen and Haibo He. Sera: selectively recursive approach towards nonstationary imbalanced stream data mining. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 522–529. IEEE, 2009.

[178] Shuo Wang, Leandro L Minku, and Xin Yao. Online class imbalance learning and its application in fault detection. *International Journal of Computational Intelligence and Applications*, 12(04), 2013.

[179] Nikunj C Oza. Online bagging and boosting. In *Systems, man and cybernetics*, volume 3, pages 2340–2345. IEEE, 2005.

[180] R. Brause, T. Langsdorf, and M. Hepp. Neural data mining for credit card fraud detection. In *Tools with Artificial Intelligence, Proceedings*, pages 103–106. IEEE, 1999.

[181] P.K. Chan, W. Fan, A.L. Prodromidis, and S.J. Stolfo. Distributed data mining in credit card fraud detection. *Intelligent Systems and their Applications*, 14(6): 67–74, 1999.

[182] Dimitris K Tasoulis, Niall M Adams, and David J Hand. Unsupervised clustering in streaming data. In *ICDM Workshops*, pages 638–642, 2006.

[183] F. Provost, T. Fawcett, et al. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Proceedings of the third international conference on knowledge discovery and data mining*, pages 43–48. Amer Assn for Artificial, 1997.

[184] D.J. Weston, D.J. Hand, N.M. Adams, C. Whitrow, and P. Juszczak. Plastic card fraud detection using peer group analysis. *Advances in Data Analysis and Classification*, 2(1):45–62, 2008.

[185] Agus Sudjianto, Sheela Nair, Ming Yuan, Aijun Zhang, Daniel Kern, and Fernando Cela-Díaz. Statistical methods for fighting financial crimes. *Technometrics*, 52(1), 2010.

[186] Naeem Siddiqi. *Credit risk scorecards: developing and implementing intelligent credit scoring*, volume 3. Wiley. com, 2005.

[187] Tom Fawcett and Foster Provost. Adaptive fraud detection. *Data mining and knowledge discovery*, 1(3):291–316, 1997.

[188] Corinna Cortes and Daryl Pregibon. Signature-based methods for data streams. *Data Mining and Knowledge Discovery*, 5(3):167–182, 2001.

[189] 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*, pages 226–235. ACM, 2003.

[190] EWT Ngai, Yong Hu, YH Wong, Yijun Chen, and Xin Sun. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3):559–569, 2011.

[191] Sushmito Ghosh and Douglas L Reilly. Credit card fraud detection with a neural-network. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, volume 3, pages 621–630. IEEE, 1994.

[192] Emin Aleskerov, Bernd Freisleben, and Bharat Rao. Cardwatch: A neural network based database mining system for credit card fraud detection. In *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, pages 220–226. IEEE, 1997.

[193] J.R. Dorronsoro, F. Ginel, C. Sgnchez, and CS Cruz. Neural fraud detection in credit card operations. *Neural Networks*, 8(4):827–834, 1997.

[194] D. Sánchez, MA Vila, L. Cerda, and JM Serrano. Association rules applied to credit card fraud detection. *Expert Systems with Applications*, 36(2):3630–3640, 2009.

[195] Tian-Shyug Lee, Chih-Chou Chiu, Yu-Chao Chou, and Chi-Jie Lu. Mining the customer credit using classification and regression tree and multivariate adaptive regression splines. *Computational Statistics & Data Analysis*, 50(4):1113–1130, 2006.

[196] Krishna M Gopinathan, Louis S Biafore, William M Ferguson, Michael A Lazarus, Anu K Pathria, and Allen Jost. Fraud detection using predictive modeling, October 6 1998. US Patent 5,819,226.

[197] B Fryer. Visa cracks down on fraud. *InformationWeek*, 594:87, 1996.

[198] P. Viola and M. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. *Advances in Neural Information Processing System*, 14, 2001.

[199] K. Ting. An empirical study of metacost using boosting algorithms. *Machine Learning: ECML 2000*, pages 413–425, 2000.

[200] G.K.J. Shawe-Taylor. Optimizing classifiers for imbalanced training sets. *Advances in Neural Information Processing Systems 11*, 11:253, 1999.

[201] Andrew Fast, Lisa Friedland, Marc Maier, Brian Taylor, David Jensen, Henry G Goldberg, and John Komoroske. Relational data pre-processing techniques for improved securities fraud detection. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 941–949. ACM, 2007.

[202] Yufeng Kou, Chang-Tien Lu, Sirirat Sirwongwattana, and Yo-Ping Huang. Survey of fraud detection techniques. In *Networking, sensing and control, 2004 IEEE international conference on*, volume 2, pages 749–754. IEEE, 2004.

[203] Clifton Phua, Vincent Lee, Kate Smith, and Ross Gayler. A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*, 2010.

[204] Vladimir Zaslavsky and Anna Strizhak. Credit card fraud detection using self-organizing maps. *Information and Security*, 18:48, 2006.

[205] Dominik Olszewski. Fraud detection using self-organizing map visualizing the user profiles. *Knowledge-Based Systems*, 70:324–334, 2014.

[206] F.E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[207] V. Barnett and T. Lewis. Outliers in statistical data. *Wiley Series in Probability and Mathematical Statistics. Applied Probability and Statistics, Chichester: Wiley, 1984, 2nd ed.*, 1, 1984.

[208] Longin Jan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. Outlier detection with kernel density functions. In *Machine Learning and Data Mining in Pattern Recognition*, pages 61–75. Springer, 2007.

[209] Feng Jiang, Yuefei Sui, and Cungen Cao. Outlier detection based on rough membership function. In *Rough Sets and Current Trends in Computing*, pages 388–397. Springer, 2006.

[210] Michael H Cahill, Diane Lambert, Jost C Pinheiro, and Don X Sun. Detecting fraud in the real world. *Computing Reviews*, 45(7):447, 2004.

[211] Charu C Aggarwal and Philip S Yu. Outlier detection for high dimensional data. In *ACM Sigmod Record*, volume 30, pages 37–46. ACM, 2001.

[212] Zengyou He, Shengchun Deng, and Xiaofei Xu. An optimization model for outlier detection in categorical data. In *Advances in Intelligent Computing*, pages 400–409. Springer, 2005.

[213] Andrea Dal Pozzolo, Olivier Caelen, and Gianluca Bontempi. When is undersampling effective in unbalanced classification tasks? In *Machine Learning and Knowledge Discovery in Databases*. Springer, 2015.

[214] Jerzy Stefanowski. Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data. In *Emerging Paradigms in Machine Learning*, pages 277–306. Springer, 2013.

[215] Gustavo EAPA Batista, Ronaldo C Prati, and Maria C Monard. Balancing strategies and class overlapping. In *Advances in Intelligent Data Analysis VI*, pages 24–35. Springer, 2005.

[216] Vicente García, Jose Sánchez, and Ramon Mollineda. An empirical study of the behavior of classifiers on imbalanced and overlapped data sets. In *Progress in Pattern Recognition, Image Analysis and Applications*, pages 397–406. Springer, 2007.

[217] Vicente García, Ramón Alberto Mollineda, and José Salvador Sánchez. On the k-nn performance in a challenging scenario of imbalance and overlapping. *Pattern Analysis and Applications*, 11(3-4):269–280, 2008.

[218] Jason Van Hulse and Taghi Khoshgoftaar. Knowledge discovery from imbalanced and noisy data. *Data & Knowledge Engineering*, 68(12):1513–1542, 2009.

[219] D Anyfantis, M Karagiannopoulos, S Kotsiantis, and P Pintelas. Robustness of learning techniques in handling class noise in imbalanced datasets. In *Artificial intelligence and innovations 2007: From theory to applications*, pages 21–28. Springer, 2007.

[220] Damien Brain and Geoffrey I Webb. The need for low bias algorithms in classification learning from large data sets. In *Principles of Data Mining and Knowledge Discovery*, pages 62–73. Springer, 2002.

[221] HO Hartley and A Ross. Unbiased ratio estimators. 1954.

[222] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

[223] Alexandros Karatzoglou, Alex Smola, Kurt Hornik, and Achim Zeileis. kernlab-an s4 package for kernel methods in r. 2004.

[224] Jarek Tuszynski. *caTools: Tools: moving window statistics, GIF, Base64, ROC AUC, etc.*, 2013. URL `http://CRAN.R-project.org/package=caTools`. R package version 1.16.

[225] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[226] David H Wolpert and William G Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[227] O. Maron and A.W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. *Robotics Institute*, page 263, 1993.

[228] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2012. URL `http://CRAN.R-project.org/package=e1071`. R package version 1.6-1.

[229] Terry Therneau, Beth Atkinson, and Brian Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2014. URL `http://CRAN.R-project.org/package=rpart`. R package version 4.1-5.

[230] David A Cieslak, T Ryan Hoens, Nitesh V Chawla, and W Philip Kegelmeyer. Hellinger distance decision trees are robust and skew-insensitive. *Data Mining and Knowledge Discovery*, 24(1):136–158, 2012.

[231] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0.

[232] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32 (200):675–701, 1937.

[233] C Radhakrishna Rao. A review of canonical coordinates and an alternative to correspondence analysis using hellinger distance. *Questiió: Quaderns d'Estadística, Sistemes, Informatica i Investigació Operativa*, 19(1):23–63, 1995.

[234] David A Cieslak and Nitesh V Chawla. Detecting fractures in classifier performance. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 123–132. IEEE, 2007.

[235] Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[236] T Ryan Hoens, Nitesh V Chawla, and Robi Polikar. Heuristic updatable weighted random subspaces for non-stationary environments. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 241–250. IEEE, 2011.

[237] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[238] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *The Journal of Machine Learning Research*, 99:1601–1604, 2010.

[239] Georg Krempl and Vera Hofer. Classification in presence of drift and latency. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pages 596–603. IEEE, 2011.

[240] Wei Fan. Systematic data selection to mine concept-drifting data streams. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 128–137. ACM, 2004.

[241] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[242] Wei Fan and Ian Davidson. On sample selection bias and its efficient correction via model averaging and unlabeled examples. In *SDM*, pages 320–331. SIAM, 2007.

[243] Torsten Hothorn, Peter Bühlmann, Sandrine Dudoit, Annette Molinaro, and Mark J Van Der Laan. Survival ensembles. *Biostatistics*, 7(3):355–373, 2006.

[244] Hothorn Torsten, Hornik Kurt, and Achim Zeileis Carolin, Strobl and. *party: A Laboratory for Recursive Partytioning.*, 2015. URL `http://CRAN.R-project.org/package=party`. R package version 1.0-13.

[245] Gilles Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014.

[246] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52:55–66, 2010.

[247] DJ Hand, C Whitrow, NM Adams, P Juszczak, and D Weston. Performance criteria for plastic card fraud detection tools. *Journal of the Operational Research Society*, 59(7):956–962, 2008.

[248] Bertrand Lebichot, Ilkka Kivimaki, Kevin Françoisse, and Marco Saerens. Semisupervised classification through the bag-of-paths group betweenness. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(6):1173–1186, 2014.