DEGREE PROJECT IN TECHNOLOGY AND LEARNING,
SECOND CYCLE, 30 CREDITS
*STOCKHOLM, SWEDEN 2017*

# Fraud detection in online payments using Spark ML

Master thesis performed in collaboration with Qliro AB

**IGNACIO AMAYA DE LA PEÑA**

**KTH ROYAL INSTITUTE OF TECHNOLOGY**
**SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY**

# Fraud detection in online payments using Spark ML

Master thesis performed in collaboration with Qliro AB

Ignacio Amaya de la Peña

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

11 September 2017

Examiner: Sarunas Girdzijauskas

# Abstract

Frauds in online payments cause billions of dollars in losses every year. To reduce them, traditional fraud detection systems can be enhanced with the latest advances in machine learning, which usually require distributed computing frameworks to handle the big size of the available data.

Previous academic work has failed to address fraud detection in real-world environments. To fill this gap, this thesis focuses on building a fraud detection classifier on Spark ML using real-world payment data.

Class imbalance and non-stationarity reduced the performance of our models, so experiments to tackle those problems were performed. Our best results were achieved by applying undersampling and oversampling on the training data to reduce the class imbalance. Updating the model regularly to use the latest data also helped diminishing the negative effects of non-stationarity.

A final machine learning model that leverages all our findings has been deployed at Qliro, an important online payments provider in the Nordics. This model periodically sends suspicious purchase orders for review to fraud investigators, enabling them to catch frauds that were missed before.

# Sammanfattning

Bedrägerier vid online-betalningar medför stora förluster, så företag bygger bedrägeribekämpningssystem för att förhindra dem.

I denna avhandling studerar vi hur maskininlärning kan tillämpas för att förbättra dessa system.

Tidigare studier har misslyckats med att hantera bedrägeribekämpning med verklig data, ett problem som kräver distribuerade beräkningsramverk för att hantera den stora datamängden.

För att lösa det har vi använt betalningsdata från industrin för att bygga en klassificator för bedrägeridetektering via Spark ML. Obalanserade klasser och icke-stationäritet minskade träffsäkerheten hos våra modeller, så experiment för att hantera dessa problem har utförts.

Våra bästa resultat erhålls genom att kombinera undersampling och oversampling på träningsdata. Att använda bara den senaste datan och kombinera flera modeller som ej har tränats med samma data förbättrar också träffsäkerheten.

En slutgiltig modell har implementerats hos Qliro, en stor leverantör av online betalningar i Norden, vilket har förbättrat deras bedrägeribekämpningssystem och hjälper utredare att upptäcka bedrägerier som tidigare missades.

# Acknowledgements

First, I would like to thank my KTH supervisor Vladimir Vlassov, who gave me good advices to improve this thesis. Thanks also to my examiner Sarunas Girdzijauskas who was very helpful and solved all my doubts.

Thanks to my supervisor at Qliro, Xerxes Almqvist, for giving me the opportunity to start such a cool project and for supporting me in everything I needed to successfully finish it. I also owe a great deal of gratitude to my fellow workers at Qliro and my colleagues from the Business Intelligence team, who have made working here very pleasant. Thanks to Nils Sjögren for keeping the Hadoop cluster healthy and for translating the abstract to Swedish. Thanks to Nhan Nguyen for helping me in the data ingestion. Thanks to Qliro's head of fraud, John Blackburn, for his continuous feedback, and to the rest of the fraud team, specially Anna Lemettilä and Albin Sjöstrand, that were always willing to answer all my questions. Also, thanks to Olof Wallstedt for letting me peer review the investigators' daily activities.

I would like to thank also Andrea Dal Pozzolo for inspiring me with his Phd thesis, which has been very helpful.

Thanks to all my friends from the EIT Data Science program for our great machine learning discussions, particularly to Adrián Ramírez, who gave me great ideas while doing pull ups in Lappis.

Special thanks to my girlfriend Yoselin García, which has been very supportive during all these months.

Finally, this thesis wouldn't have been possible without the support from my parents, Jaime and M$^{a}$ Pilar, who has always encouraged me to travel abroad and follow my dreams.

# Contents

# List of Figures

# List of Tables



# Listings

# List of Notations

| | |
|---|---|
| **AI** | Artificial Intelligence |
| **AP** | Average Precision |
| **API** | Application User Interface |
| **ASUM** | Analytics Solutions Unified Method |
| **AU-PR** | Area under the PR curve |
| **AU-ROC** | Area under the ROC curve |
| **BI** | Business Intelligence |
| **BRF** | Balanced Random Forest |
| **CD** | Concept Drift |
| **CNN** | Condensed Nearest Neighbors |
| **CRISP-DM** | Cross Industry Standard Process for Data Mining |
| **CS** | Computer Science |
| **CV** | Cross-validation |
| **DAG** | Directed Acyclical Graph |
| **DF** | DataFrame |
| **ETL** | Extract, Transform and Load |
| **FDS** | Fraud Detection System |
| **FN** | False Negative |
| **FP** | False Positive |

| | |
|---|---|
| **FPR** | False Positive Rate |
| **GBT** | Gradient Boosting Trees |
| **GFS** | Google File System |
| **HDDT** | Hellinger Distance Decision Tree |
| **HDFS** | Hadoop Distributed File System |
| **HDP** | Hortonworks Data Platform |
| **HPL/SQL** | Hive Hybrid Procedural SQL On Hadoop |
| **ID** | Identity Document |
| **IDE** | Integrated Development Environment |
| **JSON** | JavaScript Object Notation |
| **JVM** | Java Virtual Machine |
| **KNN** | $K$-nearest Neighbors |
| **ML** | Machine Learning |
| **MSEK** | Million Swedish Kronor |
| **NN** | Neural Network |
| **OOS** | One Sided Selection |
| **ORC** | Optimized Row Columnar |
| **OS** | Operating System |
| **PCA** | Principal Component Analysis |
| **PPV** | Positive Predicted Value |
| **PR** | Precision-recall |
| **RDD** | Resilient Distributed Dataset |
| **RF** | Random Forest |
| **ROC** | Receiving Operating Characteristics |
| **SMOTE** | Synthetic Minority Over-sampling Technique |

**SMOTE-N** Synthetic Minority Over-sampling Technique Nominal

**SMOTE-NC** Synthetic Minority Over-sampling Technique Nominal Continuous

**SQL** Structured Query Language

**SSB** Sample Selection Bias

**SSH** Secure Shell

**SVM** Support Vector Machine

**TP** True Positive

**TPR** True Positive Rate

**TN** True Negative

**UDF** User Defined Function

**UI** User Interface

**VCS** Version Control System

**WRF** Weighted Random Forest

**XML** Extensible Markup Language

**YARN** Yet Another Resource Negotiator

# 1

# Introduction

*"Nothing behind me, everything ahead of me, as is ever so on the road."*

– Jack Kerouac, *On the Road*

In this chapter the targeted problem is introduced and the aim of this thesis is detailed by defining the objectives and delimitations that shape our research question.

The chapter starts with Section 1.1 providing the motivation for this work. Then, Section 1.2 presents the research question. Section 1.3 focus on the contributions made in this project, while Section 1.4 highlights the delimitations of this work. In Section 1.5 some ethical and sustainability considerations are discussed. Finally, Section 1.6 gives an overview of how this document is structured.

## 1.1 Motivation

Technological revolution has disrupted the finance industry during the last years. A lot of startups, the so-called *fintechs*, have brought innovation into banking. Payments have been simplified and mobile access to all our financial operations is now a reality. User friendly and more transparent financial services are being created, improving the way customers manage their finances. However, securing the increasing number of transactions can become a problem if the fintechs fail to scale up their data processes.

As payments increase, the number of frauds start to be significant enough to translate into important losses for the companies. Bank transaction frauds cause over $13B annual losses [1], which affect not only banks and fintechs, but also

their clients*. Regulations regarding fraud detection also need to be complied, so this becomes a very important task that has to be handled adequately. When the number of transactions is small, fraud detection can be worked around with some hand-crafted rules, but as the company grows, their complexity increases. Adding more rules and changing existing ones is cumbersome and validating the correctness of the new rules is usually hard. That is one of the reasons why expert systems based on rules are highly inefficient and pose scalability issues. One way to solve this is to combine these systems with automated approaches that leverage the data from previous frauds. By doing that, only basic rules, which are easier to maintain, need to be implemented. These automatic systems can detect difficult cases more accurately than complex rules and don't present scalability problems.

In order to implement an efficient Fraud Detection System (FDS), multidisciplinary teams are needed. Data scientists, data engineers and domain experts are required to work together. Data engineers create data pipelines that fetch the information needed from production systems and transform it into an usable format. Then, data scientists can use that data to create fraud prediction models. Fraud investigators are also important, as they have extensive knowledge about the fraudsters' behavior, which can save a lot of time of data exploration to the data scientists. Finally, the data engineers need to put the models created by the data scientists into production. Small companies don't usually have the data pipeline processes already in place, increasing the complexity of implementing the FDS.

University research projects often focus on comparing different techniques and algorithms using toy datasets, so they don't have into account data level problems or deployment issues, which are usually present in real world cases. Dealing with financial data is sensible, so publicly available datasets in this study area are scarce. This slows down further advances because researchers use different datasets, which most of the times can't be made public, so comparing the results is hard.

Another problem is that most of the academic work is not thought to be implemented and productionized in real environments. Conversely, companies aim to create models that can be deployed and integrated into their FDSs, but they rarely share their findings.

During the last years, the amount of information that is being generated has increased at a big pace. Although this is very promising for analytics, it might also turn into a problem because no insights can be extracted from them without the right infrastructures and technologies. Companies' data pipelines that correctly handle these volume of data require more technical complexity in order to scale well. One of the main reasons for this big complexity is the frequent bad quality of that data, which hinders analytics.

* The interest rates and membership fees usually increase to compensate fraud losses.

A wide number of studies using small datasets have been performed, but very few of them use distributed systems to deal with larger amounts of unclean data, which is the most common scenario in real life problems. Toy datasets are handy to compare different techniques and algorithms because using big real-world datasets for this purpose is cumbersome. Nevertheless, some very promising techniques that work locally for small datasets become really hard to distribute and impossible to utilize when dealing with big datasets. Hence, there is an increasing need to explore these problems in detail.

## 1.2 Research question

Our main research question is how can we tackle the main problems faced in fraud detection in a real-world environment, specially when dealing with large amounts of unclean data.

The approach taken to solve that problem is to create an automatic FDS using Spark [2], a distributed computing framework that can handle big data volumes. The work realized has been done in collaboration with Qliro, a fintech company that focuses on providing online payment solutions for e-commerce stores in the Nordics. Therefore, we can use real-world data with quality issues, something very common in other datasets in the industry. Special focus on deployment is taken into account to reduce the gap between data scientists and data engineers when creating a real FDS. The predictions obtained have been evaluated and compared against the current system in-place at Qliro.

This work follows up on some open issues highlighted in Dal Pozzolo's Phd thesis [3]:

- **Big data solutions regarding fraud detection systems:** Dal Pozzolo studies the fraud detection problem in detail, but his findings are not applied into leveraging distributed systems to deal with huge amounts of data. This thesis tries to create an automatic FDS in a distributed framework, tackling the problems he discusses in [3].

- **Modeling an alert-feedback interaction system:** Dal Pozzolo suggests splitting the fraud detection problem in two parts: one that deals with fraud investigators' feedbacks and other that focuses on the customers' claims. This way, the fraud detection problem is modeled as an alert-feedback interaction system. Experiments have been performed to check whether or not this approach improves the results in our dataset.

- **Defining a good performance measure:** although there is a consensus about missing frauds being worse than generating false alerts, no agreement

on how to measure fraud detection performance has been reached. Different metrics have been explored in this thesis and some conclusions regarding their problems have been presented.

## 1.3 Contributions

The contributions of this thesis project address the challenges presented in Section 1.1. They cover three different areas that are usually treated separately:

- **Data science:** this is the most academical part of this thesis, and includes the experiments performed to deal with the machine learning problems present in fraud detection. Different techniques and algorithms from the literature have been studied, and some of them have been compared using a real-world dataset. The main contributions extracted from these experiments are:
  - **Learning from skewed data:** solutions to tackle the class imbalance problem are proposed. Unbalanced datasets are explored using a combination of different techniques based on previous literature solutions.
  - **Dealing with non-stationary distributions:** different approaches to learn from evolving datasets are compared.
  - **Alert-feedback interaction:** the fraud detection problem has been also studied from the perspective of alert-feedback interaction systems.
  - **Metrics analysis in fraud detection:** study of the different metrics to identify their suitability to select the best fraud detection models.
- **Data engineering:** this part covers the technical implementation of a fraud detection classifier using Spark ML. The design, implementation and deployment are included here. Our implementation in Spark ML is highly scalable, which is essential to handle the big size of real-world datasets. The process of obtaining the best model to implement has been described, and all the reasons behind the different choices in the data mining pipeline are backed up with previous studies or empirical facts based on experiments. The main contributions regarding this part are:
  - **Deployment design:** a deployment pipeline is presented, which explains how to productionize our fraud detection model, periodically sending possible frauds to the investigators and retraining the model automatically.

  - **Open sourced spark-imbalance project:** a SMOTE-NC [4] implementation on top of Spark Datasets, a decision tree JSON visualizer, and stratified cross validation have been implemented and have been made available to the research community. More information about this project is presented in Appendix B. This is a useful contribution, as Spark packages to work with unbalanced data are scarce. An implementation of weighted random forest is still under development by the Spark community, which will be an option for dealing with unbalanced datasets in future Spark releases.

- **Business:** this project has been carried on following the CRISP-DM methodology, which gives a lot of importance to the business evaluation of the final solution. Also, as this work has been done in collaboration with Qliro, its business application was always present.

## 1.4 Delimitations

The main delimitations in this work are related with the problems described in Section 1.1.

- **Not extensive methods and techniques comparison:** this work focus on implementing solutions for big data in Spark, not in comparing a wide variety of methods. Algorithmic comparisons have been previously studied extensively in small and medium sized dataset, but comparing different distributed algorithms is not the focus of this work.

- **No public dataset:** the data used in this project is private and very sensitive to Qliro. This is why specific details about the dataset are not disclosed in the thesis. This limitation could be overcome if a real-world dataset for fraud detection was available for the research community as mentioned in Section 1.1.

- **Feature engineering not treated in detail:** hundreds of data sources were available at Qliro to be used and sometimes the data was very unclean. Some work has been done collecting tables useful to detect frauds and preprocessing them. However, proper feature engineering considering all the sources has not been performed, as it would have been very time consuming. Advanced Extract, Transform and Load (ETL) efforts are out of the scope of this work, as the focus is on solving the fraud detection problem, instead of very specific data collection and preprocessing issues.

- **Spark ML limitations:** the Machine Learning (ML) development is done using Spark ML, so all the algorithms and features not present in the current version are not contemplated (the versions used are shown in Appendix A).

## 1.5 Ethics and sustainability

Developing an automated fraud detection system raises some ethical concerns. When potential fraudsters are detected, the associated transactions are blocked. However, non-fraudsters can be also rejected due to mistakes in the FDS. It is important not to use features that could bias the model against specific parts of the population, such as gender or ethnicity. Tracking back the reasons for the model decisions is hard, difficulting the understanding of why a specific person's transaction was denied. Therefore, assuring that the outcome of the models is ethical is important before putting them into production.

Regarding sustainability, we can find social issues related with the ML field advancing at a very fast pace. In a near future, no human labor might be needed to perform fraud detection. If that transition is not performed gradually and in a responsible way, investigators could lose their jobs. Hence, special care is needed when implementing new processes that leverage ML in order to avoid social problems derived from an increase of the unemployment rate. Environmental sustainability is also a concern, due to the big consumption of distributed systems. Therefore, small increases in the fraud detection performance might not be desirable if their associated footprint is much higher.

## 1.6 Outline

The thesis is structured in the following chapters:

**Chapter 2** briefly covers the different areas and concepts used in the rest of the thesis.

**Chapter 3** presents the techniques found in the literature to solve the fraud detection challenges introduced in Section 1.1. First, Section 3.1 analyzes the class imbalance problem. Then, Section 3.2 focus on problems regarding time evolving data. Section 3.3 studies both previous problems combined. Finally, Section 3.4 investigates fraud detection from the perspective of alert-feedback interaction systems.

**Chapter 4** describes the hardware and software used to carry on the work

presented in Chapter 5 and Chapter 6.

**Chapter 5** describes the tasks performed to build an automatic FDS layer and the related experiments performed. The different techniques used are explained and justified.

**Chapter 6** presents the the outcome of the experiments previously introduced.

**Chapter 7** summarizes the findings extracted from the results presented in Chapter 6. Lessons learned and open issues are highlighted. Future lines of work to extend this work are also suggested.

**Chapter 8** concludes this thesis highlighting the implications of this work in the existing academic community and how it relates to real world problems outside academia.

# 2

# Background

*"The past is never dead. It's not even past."*
– William Faulkner, *Requiem for a Nun*

This chapter comprises the theoretical background used in the rest of the thesis.

First, Section 2.1 explains the business of Qliro, the company providing the data and the infrastructure used in this thesis. Then, Section 2.2 describes CRISP-DM, the methodology framework employed. Section 2.3 introduces the ML field, focusing in the binary classification task. Different performance metrics and algorithms used are explained here. Section 2.4 introduces distributed systems, which are necessary to handle large amounts of data. Finally, Section 2.5 explains the fraud detection problem, covering the definition of a FDS.

## 2.1 Qliro

Qliro offers financial products to simplify payments in e-commerce stores. It is currently operating in some of the biggest online retailers of the Nordic countries. They focus on simplifying the purchase process by reducing to a minimum the amount of data the user has to input, thus providing seamlessly shopping experiences.

The company was founded in 2014 as part of Qliro Group, which comprises some of the most important e-commerce stores of Sweden, such as CDON or Nelly. Since then, it has been growing at a fast pace, transitioning from a small startup mindset to a medium sized scaling company.

Different countries regulations make the whole payment flow differ from one country to another, so in this thesis the work will be constraint to detecting frauds

for the payments in Sweden.

Swedish citizens have a personal number, which is used to identify them for social security purposes, as well as other educational or health services. Using Qliro as a payment method, customers only need to provide their personal number, which is used to obtain the banking information associated.

Qliro currently implements different fraud detection methods to prevent fraudsters from impersonating other customers. As the company grows bigger, ensuring that the fraud detection is keeping credit losses to a minimum becomes increasingly important.

## 2.2 CRISP-DM

Cross Industry Standard Process for Data Mining (CRISP-DM) [5] is a process model that serves as a guide to perform data mining projects. It was conceived in 1996 and published by IBM in 1999. It is the most popular methodology in data science projects [6]. Some companies use their own methodologies, but they are similar to CRISP-DM, which captures the essential challenges in the data mining process.

This methodology breaks down the data mining process in six different phases, which make emphasis in keeping the business objectives in mind.

- **Business understanding:** it focuses on understanding the business needs and creating a data mining plan aligned with them.
- **Data understanding:** it comprises the data collection and the first analysis of the data to detect quality problems, and to acquire the first insights.
- **Data preparation:** activities that transform the raw data into the dataset that will be used for modeling. Data cleaning and data selection are included here. This step is usually very time consuming and usually runs almost in parallel with the modeling phase.
- **Modeling:** different modeling techniques are applied and their parameters are selected. The input required by different algorithms usually varies, so going back to the previous stage to prepare the data differently or select other attributes is usually unavoidable.
- **Evaluation:** once models achieve good performance, reviewing the steps taken and making sure they are aligned with the business rules is important to decide whether the models should be deployed into production. In case the business objectives are not met, it is necessary to redefine them, going back to business understanding.

- **Deployment:** normally, models created can't be directly applied into the company's flow to generate immediate value. Integrating the models into the company's infrastructure or generating reports and visualizations that summarize the findings isn't usually straightforward. This phase is often underestimated, so most models never reach production, even if their results are good.

All these steps are interconnected. We can see how the flow of CRISP-DM project looks like in Figure 2.1.

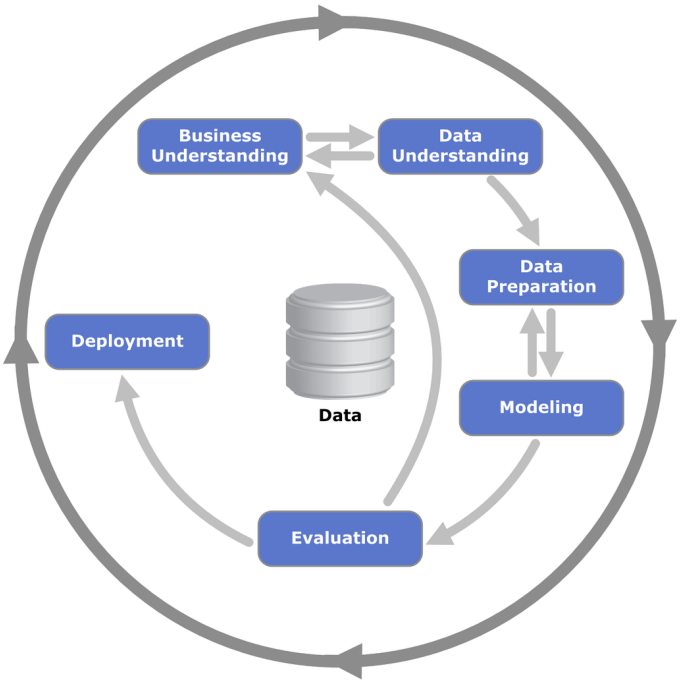



Figure 2.1: CRISP-DM methodology diagram with the relationship between the different phases (diagram created by Kenneth Jensen[†]).

Some CRISP-DM projects take some shortcuts to speed up the process, but this often derives into a corrupted CRISP-DM methodology. This can cause projects to fail in bringing value to the business [7].

The main four problems that arise from those corrupted versions are:

- **Skipping the business understanding:** due to the lack of clarity regarding the business problem, some teams decide to jump into the data analysis step

[†] This image is under the license CC BY-SA 3.0, via Wikimedia Commons. More information about them can be found in http://creativecommons.org/licenses/by-sa/3.0 and in https://commons.wikimedia.org/wiki/Main_Page.

without having clear business goals and metrics to measure the success of the project.

- **Evaluation in analytic terms:** some models that yield good predictions from the analytics point of view might not be meeting business objectives. However, if these objectives are not clear this is very difficult to assess. Most teams take the shortcut of iterating back to the data understanding step, instead of going all the way back to business understanding and re-evaluate the business problem in conjunction with the business partners.

- **Gap between analytic teams and deployment:** some teams don't think about deploying the model, so they usually hand the final model to another team in charge of taking it into production. If no deployment considerations are made while developing the models, the time and cost to take them into production are likely to be high. Sometimes operational issues make that deployment impossible. This is why most models never end up providing business value.

- **Lack of model maintenance:** iterating again over the CRISP-DM process is necessary to keep the value of the models from decreasing over time. However, some teams leave models unmonitored. Sometimes this is due to the lack of clarity in the business metrics to track. In other cases, new projects catch the teams' attention, leaving the model maintenance relegated. Monitoring and updating the models is essential to be aligned with the business needs and the fast changing data environments. Feedback loops [8], which are situations where the ML model is indirectly fed into its own input are also important to avoid. This is usually hard, as collaboration across different teams is usually needed.

In 2015, IBM Corporation released Analytics Solutions Unified Method (ASUM), which refines and extends CRISP-DM, providing templates to ease the process at the cost of complexity overhead [9]. However, as the templates are private and the changes in the methodology are minimal, it has not been considered for this thesis. The CRISP-DM methodology has been used instead as a guideline to structure the project. The common errors of this approach have been taken into account to alleviate the pains derived from them.

## 2.3 Machine learning

Machine Learning is a subfield of computer science which focuses on creating algorithms that can learn from previous data and make predictions based on that.

The statistics field also has as a goal learning from data, but ML is a subfield of Computer Science (CS) and Artificial Intelligence (AI), while statistics is a subfield of mathematics. This means that ML takes a more empirical approach and has its focus on optimization and performance. However, both are branches of predictive modeling, so they are interconnected and their convergence is increasing lately. The notion of data science tries to close the gap between them, allowing collaboration between both fields [10].

Regarding the CRISP-DM stages, ML is part of the modeling phase described in Section 2.2. Fraud detection is a supervised ML task because the learning comes from previously tagged samples. In this problem we want to correctly predict a categorical variable (target variable), which can have two possible values: fraud or non-fraud. This type of supervised learning is called binary classification. The ML algorithm learns from the labeled data with the objective of predicting unseen observations with high confidence. The outcome of the modeling step is a classifier model, which receives a set of untagged instances and determines whether they are fraud or not.

In Section 2.3.1 the binary classification task is formally defined. Then, Section 2.3.2 focus on the different datasets divisions used in ML. Section 2.3.3 describes the different performance measures used to evaluate the outcome of the trained model. Afterwards, Section 2.3.4 comments on the problems derived from selecting and creating new features. Finally, Section 2.3.5 explains the ML algorithms used in this thesis.

### 2.3.1 Binary classification

Let $X = (X_1, \cdots, X_p) \in \mathcal{X} = \mathbb{R}^p$ represent the predictors and $Y$ the target variable with a binary categorical outcome $\mathcal{Y} = \{0, 1\}$. Given a set of $n$ observation pairs, we can construct a *training dataset* $D_n = \{(x_i, y_i), i = 1 \cdots, n\}$ where $(x_i, y_i)$ is an independent and randomly distributed sample of $(X, Y)$ taken from some unknown distribution $P_{X,Y}$ [11].

The aim of binary classification is to find a function $\phi : \mathcal{X} \to \mathcal{Y}$ based on $D_n$ which can be generalized to future cases obtained from the distribution $P_{X,Y}$.

This definition can be generalized for multi-class classification, where the class labels are $\mathcal{Y} = \{1, \ldots, k\}$.

### 2.3.2 Data sampling

ML algorithms require data adequately preprocessed in order to generate accurate models. Three different datasets that serve different purposes are also needed:

- **Training set:** dataset used for learning the classification model.

- **Test set:** dataset used to determine the performance of the trained model regarding unseen data observations.
- **Validation set:** dataset used to tune the hyperparameters[‡] of the ML algorithms. This can't be done using the test set because we would be creating a bias towards it.

In order to increase the model generalization and predictive performance, Cross-validation (CV) can be used. This is a model validation technique to tune hyperparameters and reduce the bias derived from picking a specific training and validation set. The most popular CV technique is k-fold Cross-validation, where the original data is randomly partitioned into $k$ subsamples of the same size. Each of these samples is a *fold* and it will be used as a validation set for a model trained with the rest of the data. This means that $k$ models are trained with training sets consisting in $k-1$ folds each. Ten-fold CV is usually used as some studies claim that it yields the best results [12]. However, sometimes less folds are employed due to computational time limitations. In [12] different validation techniques are compared against each other using real-world datasets. They conclude that the best approach when dealing with class imbalance is stratified CV, which ensures that each fold has the same number of instances from each class.

### 2.3.3 Performance measures

In this section we first explain the bias-variance trade-off to understand where the errors in our model come from. Then, we define the confusion matrix for a set of predictions, which is very useful for understanding how the model is behaving. Different metrics obtained from the confusion matrix are explained afterwards. Finally, some curves derived from those metrics are also presented.

#### Bias-variance trade-off

When measuring the predicting error of our model, we can decompose it into two main subcomponents [13]:

- **Error due to bias:** this error measures the difference between the expected prediction of our model and the correct value that is trying to predict. Models with high bias don't learn enough from the training data and miss important relationships between features (this is known as under-fitting).
- **Error due to variance:** this error is caused by the prediction variability for a given data point. Models with high variance can perform very well

[‡] Model hyperparameters express properties specific to the training model and are fixed before the training starts.

in the training set, but they are not able to generalize properly to other data samples due to noise in modeling (this is known as over-fitting).

In Figure 2.2 we show an example of how the bias and variance can affect our models. A dartboard is used to exemplify the performance of our model in the test set. Hence, when we have a high bias and low variance, all the darts are very precise, but they target the wrong point because the model presents under-fitting and is not learning enough. When we have high variance and low bias, the model learns how to predict the target variable in the training set, but it doesn't generalize to test set, so the darts are very dispersed around the target due to over-fitting. When the model has low bias and variance, it learns correctly the boundaries between the two classes.

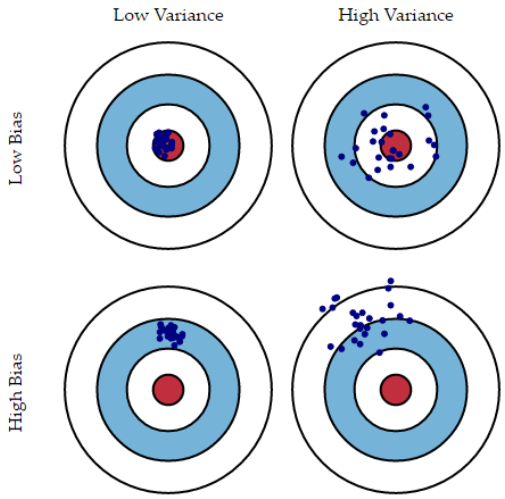



Figure 2.2: Graphical illustration of bias and variance (extracted from [13]).

If $Y$ is our target variable and $X$ represents our features, we may assume that a relationship exists between them. If $Y = f(X) + \varepsilon$ is that relationship, where $\varepsilon \sim N(0, \sigma_\varepsilon)$ with $E(\varepsilon) = 0$ and $Var(\varepsilon) = \sigma_\varepsilon^2$, then we can estimate $f(X)$ with a model $\hat{f}(X)$. For a point $x_0 \in X$ we have the following error [14]:

$$\begin{aligned} Err(x_0) &= \left(E[\hat{f}(x_0) - f(x_0)]\right)^2 + E[\left(\hat{f}(x_0) - E[\hat{f}(x_0)]\right)^2] + \sigma_e^2 \\ &= \text{Bias}^2 + \text{Variance} + \text{Irreducible error} \end{aligned}$$

The first term is the squared bias, which represents the amount by which the average of $\hat{f}(x_0)$ differs from its true mean, $f(x_0)$. The second term is the variance, the expected squared deviation of $\hat{f}(X)$ around its mean. The last term of this

equation, the irreducible error, represents the variance of the target variable around its true mean and it can't be avoided unless $Var(\varepsilon) = \sigma_\varepsilon^2 = 0$.

Although in theory the bias and variance could be reduced to zero, in practice this is not possible, so there is a trade-off between minimizing the bias and minimizing the variance. Typically, using a more complex model $\hat{f}$ reduces the bias, but increases the variance.

### Confusion matrix

Let $y_1$ be the set of positive instances, $y_0$ the set of negative instances, $\hat{y_1}$ the set of predicted positive instance, and $\hat{y_0}$ the set of predicted negative instances. In a binary classification problem, we can create a $2x2$ matrix, which captures the correctness of the labels assigned by our model.

We can see in Figure 2.3 that the correct classified instances are in the diagonal of the matrix, the True Negative (TN) and True Positive (TP) cases. The misclassified instances are the False Negative (FN) and False Positive (FP) ones.

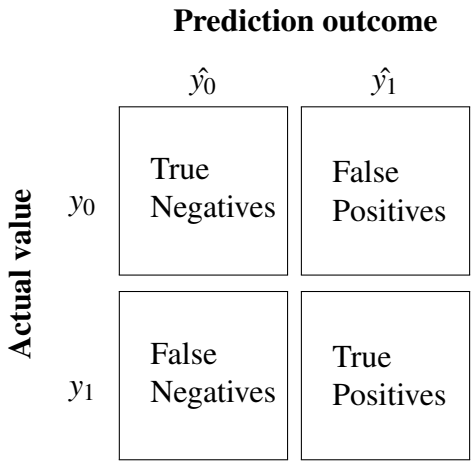



Figure 2.3: Confusion matrix in binary classification.

In the case of fraud detection is worth noticing that the most important value to increase is the number of TPs cases, which correspond with the correctly detected frauds. Metrics involving the TNs are usually not useful because this number is usually much greater than its TP counterpart, as frauds are usually rare. Therefore, our objective is to find the right balance of FNs and FPs, while maximizing the TP observations. Usually, minimizing the FN instances is prioritized over minimizing the FPs due to the higher value in detecting new frauds. As each transaction has a cost associated, matrices based on the cost of every transaction can be derived from the confusion matrix and can be useful to evaluate the business value of the generated models.

### Metrics based on the confusion matrix

Different metrics can be computed from the confusion matrix depending on what we are interested on measuring.

- **Accuracy:** $\frac{TP+TN}{TP+FP+TN+FN}$.
- **Precision:** $\frac{TP}{TP+FP}$, also called Positive Predicted Value (PPV).
- **Recall:** $\frac{TP}{TP+FN}$, also called True Positive Rate (TPR), sensitivity or hit rate.
- **False positive rate (FPR):** $\frac{FP}{FP+TN}$, also called fall-out or probability of false alarm.
- **$F_\beta$-score:** also called $F_\beta$- measure. Usually F-measure refers to the $F_1$ -measure: ):

$$F_\beta = (1+\beta^2)\frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} = \frac{(1+\beta^2)\cdot TP}{(1+\beta^2)\cdot TP + \beta^2 \cdot FN + FP}$$

In an unbalanced classification problem, the metrics calculated using the TNs are misleading, as the majority class outnumbers the minority class. For example, a model that doesn't predict any minority cases can still have a good performance according to metrics that use TPs.

In most of the cases models with high precision suffer from low recall and vice-versa. In order to combine both metrics, the $F_\beta$-score is used, which converts them into a value ranging between 0 and 1. If we want to give the same importance to precision and recall, then $F_1$ is used. However, as we want to prioritize reducing the FNs, we can use the $F_2$, which favors minimizing the recall.

### Learning curves

Different curves can be created by evaluating a classifier over a range of different thresholds[§]. The curves most widely used are:

- Receiving Operating Characteristics (ROC) curve: the $x$-axis represents the $FPR$ while the $y$-axis measures the recall.

[§] This threshold refers to the discrimination threshold of a binary classifier. In the case of fraud detection the output of a probabilistic classifier will indicate the probability of a sample being a fraud. Different predictions can be obtained depending on where we set the probability threshold for considering a fraud.

- Precision-recall (PR) curve: the $x$-axis represents the recall while the $y$-axis measures the precision.

Given the curves of two classifiers if one dominates the other[¶], then this means it is a stronger classifier.

By plotting the curves we can determine the optimal threshold for a given curve. In the PR curve this is not so intuitive, so using the PR-gain curve can help in model selection [15].

The area under these curves is a common assessment to evaluate the quality of a classifier [16, 17, 3]. Using them, we can determine how the classifier performs across all the thresholds, so this metric comprises more information that the ones derived from the confusion matrices, as there are different confusion matrices for different thresholds. The formal definition for the metrics based on the area under these curves are:

- AU-ROC: $\int_0^1 \frac{TP}{TP+FN} \mathrm{d}(\frac{FP}{FP+TN}) = \int (\text{recall} \cdot d(\text{FPR}))$
- AU-PR: $\int_0^1 \frac{TP}{TP+FP} \mathrm{d}(\frac{TP}{TP+FN}) = \int (\text{precision} \cdot d(\text{recall})$

There is a dependency between the ROC space and the PR space. Algorithms that optimize the AU-PR are guaranteed to optimize also the AU-ROC. However, the opposite is not necessarily true [17]. One problem of using the AU-PR is that interpolating this curve is harder [17].

In [18] the authors show that the PR curve is better suited than the ROC curve for imbalanced data. The ROC curve presents in the $x$-axis the False Positive Rate (FPR), which depends on the number of TN. In problems where the TN cases have no importance, such as in fraud detection, the PR curve is more informative because neither of its axis depends on TNs. Information about the FPs in relation with the minority class is lost in the ROC curve. As precision and recall are TN agnostic, the PR curve is a better fit for the fraud detection problem.

### 2.3.4 Feature engineering

The fraud detection problem can be observed from two different perspectives depending on whether we consider the customer as a feature or not:

- **Customer-dependent:** the history of an individual is considered in order to identify frauds. These approaches focus on detecting changes in behavior. However, this is sometimes risky, as a change in the purchase habits is not always an indicator of fraud.

[¶] One learning curve dominates another when all its values are above that second curve.

- **Customer-agnostic:** the customer making the purchase is ignored, so the history associated with the customer is not taken into account to detect the frauds. These techniques only look at the transaction level, so they lack a lot of information that is usually helpful to detect the frauds.

Typically, behavioral models are used as a first line of defense, while transactional ones complement and improve the predictions afterwards. Alternatively, there are mixed approaches where some customer behavior is added to the model in a process called *feature augmentation*.

### 2.3.5 Algorithms

In this section the ML algorithms used in this thesis are explained. These algorithms focus on the binary classification task explained in 2.3.1. Other approaches, like unsupervised methods [19, 20] are also possible, but out of the scope of this thesis.

This thesis faces the additional problem of dealing with large amounts of data. Using distributed implementations of those algorithms allow us to run them in a reasonable amount of time. Some popular ML algorithms, such as Support Vector Machines (SVMs) or Neural Networks (NNs), are not easily distributed. The scalability of the algorithms also has to be taken into account, as some distributed versions of the algorithms scale better than others.

The algorithms that have been studied for the fraud detection problem are weighted logistic regression and Random Forest, which is an ensemble of decision trees.

All of them are distributed in Spark ML and their implementations scale reasonably well when the amount of data increases.

#### Logistic Regression

The fraud probability can be expressed as a function of the covariates $X_1, \cdots, X_n$. We denote this probability as $\pi(x_1, \cdots, x_p)$ for $x_i \in X_i$.

We could consider a linear model for the function $\pi$:

$$\pi(x_1, \cdots, x_n) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

However, the linear equation can yield values in the range $(-\infty, \infty)$, while the probability function $\pi$ can only take values in the range $(0, 1)$. This problem can be solved using a *logit* transformation. The formula of this transformation for a given probability $p$ is specified below.

$$logit(p) = log\left(\frac{p}{1-p}\right) = log(p) - log(1-p)$$

Applying this transformation on the left side of the equation we obtain a logistic regression model [21]:

$$logit\left(\pi(x_1, \cdots, x_n)\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

In the case of class imbalance, observations of the different target classes can be adjusted using a weighted logistic regression to compensate the differences in the sample population [22]. Using inversely proportional values of the class frequencies for those weights can alleviate the imbalance problem.

### Decision trees

The main algorithms for decision trees were described by Quinlan in [23]. The first one was ID3, which only supports categorical values. Then, CART and C4.5 extended ID3 with additional features, but the basic algorithm remained the same. In this section we will explain how ID3 works and comment briefly on the improvements added by the other approaches.

In a decision tree each node corresponds to a feature attribute and each edge is a possible value to that attribute. Each leave of the tree refers to the expected value of the target variable for the records obtained following the path from the root to that leaf (this can be converted into a set of rules).

As in binary classification we only have two categorical values for the target variable, the obtained decision tree is binary. This means that each internal node has two children with the different possible values for the target variable.

Algorithms that generate decision trees are greedy[‖] and perform a recursive binary partitioning of the feature space. These partitions are chosen greedily by selecting the split that maximizes the information gain at each given tree node. The downside of this approach is that the algorithm doesn't look back to improve previous choices.

The information gain measures how well an attribute helps in predicting our target variable and is the difference between the parent node impurity and the weighted sum of its children impurity. If a split $s$ partitions a dataset $D$ of size $n$ into two datasets $D_1$ and $D_2$ of sizes $n_1$ and $n_2$, then the information gain is:

$$IG(D,s) = Impurity(D) - \frac{n_1}{n} Impurity(D_1) - \frac{n_2}{n} Impurity(D_2)$$

[‖] A greedy algorithm follows the heuristic of making a locally optimal choice at each stage. This heuristic is based on the idea that this will lead to an approximation of a globally optimal solution.

Two different impurity measures are used in Spark: Gini impurity and entropy. They are defined at the end of this section when introducing the Random Forest (RF) hyperparameters.

The original ID3 algorithm only supported categorical variables, so in CART bins are used to extend support to continuous variables.

In C4.5 the notion of gain ratios is introduced. Information gain tends to favor attributes with a large number of values, so the gain ratio tries to fix this problem. However, it is also more susceptible to noise.

In the version of Spark used in this thesis the decision trees are built using ID3 with some CART improvements [24].

In this Spark version, the recursive construction of the tree stops when one of the following conditions is satisfied:

1. The maximum depth of the tree is reached.

2. No split candidate lead to an information gain greater than the minimum information gain specified. In our case we have used zero as the minimum, so every split that improves the information gain will be considered.

3. No split candidate produces child nodes with at least a given minimum of instances. In our case we have used 1, so every child node needs to have at least one training instance.

The main advantage of decision trees is that the outcome can be transformed into a set of rules that are easy to interpret. They can also handle categorical and continuous data, while other methods such as logistic regression require the categories to be transformed into dummy variables. Also, feature scaling is not required as in most other methods (such as in logistic regression). They also have good performance with noisy data. As these decision trees are nonparametric models[**], they are good when we don't have prior knowledge about the data, so choosing the right features is not so critical. They can capture non-linearities, so they are popular in fraud detection, where complex non-linear relationships exist.

One of the main problems of decision trees is that they tend to overfit the training data. Non-parametric models have that limitation, but decision trees are very susceptible to it because when the trees are too deep, very specific rules from the training data are learned. Pruning helps simplifying the trees, allowing for better generalization.

In order to reduce the overfitting, ensemble methods are usually used to combine several decision trees. However, the interpretation of the results is harder in this case. Ensembles can also help with the class imbalance problem, as

[**] Nonparametric models use all the data for making the predictions, rather than summarizing the data first with a few parameters.

discussed in Section 3.1. The two main decision trees ensembles are Random Forest and Gradient Boosting Trees.

There are two main voting strategies used when combining the classifiers using ensembles:

- **Hard-ensemble:** it uses binary votes from each classifier. The majority vote wins.
- **Soft-ensemble:** it uses real value votes from each classifier. The different values are averaged.

Hard-ensemble usually performs better. However, in seriously unbalanced datasets it can cause a negative effect [25].

### Random forest

Random forest [26] is an ensemble technique to combine several regression trees which leverages bootstrapping[††] of the training data and random feature selection at each split of the trees generated. The predictions are obtained by aggregating the results of each of the decision trees using a hard-ensemble or a soft-ensemble.

In this thesis RF has been chosen over Gradient Boosting Treess (GBTs) because it scales better and less hyperparameter tuning is required to achieve good results.

As most ML algorithms, RF suffers from problems in highly imbalanced datasets, so poor results in the minority class predictions will be obtained due to the focus of the algorithm in minimizing the overall error rate, which is dominated by predicting correctly the majority instances. There are two main modifications in RF to deal with this problem [27]:

- **Balanced Random Forest:** when taking the bootstrap sample from the training set, a non representative number of observations from the minority class are taken (sometimes even no minority instances are taken). A way to solve this is to apply a stratified bootstrap (e.g. sample with replacement from within each class). This is usually achieved using downsampling of the majority class, but oversampling can also be considered.
- **Weighted Random Forest:** this modification uses cost sensitive learning. A penalty towards classifying accurately the majority class is added, so correct predictions from the minority class have a higher weight. As Weighted Random Forest (WRF) assigns high weights to minority examples, it is more vulnerable to noise, so mislabelled minority cases can have a large negative effect in the results.

[††] Bootstraping refers to those resampling techniques that rely on random sampling with replacement.

Spark RF models have several hyperparameters that need to be set before the training starts. The most important ones are:

- **Number of trees:** number of trees in the ensemble. Increasing the number of trees reduces the predictions variance. However, training time increases linearly with the number of trees.

- **Maximum depth:** maximum depth of each tree in the forest. This allows to capture more complex relationships in the model, but it increases the training time and the risk of overfitting. Training deep trees in RF is acceptable because the overfitting associated with it can be countered when averaging the trees.

- **Subsampling rate:** fraction of the training set used in each tree. Decreasing this value speeds up the training.

- **Feature subset strategy:** number of features to consider as candidates for splitting at each decision node of the tree. This number is specified as a fraction of the total number of features (the values used in this thesis are *onethird*, *log2* or *sqrt*) or using the absolute value of features to consider.

- **Impurity measure:** measure of homogeneity of the labels at a given node. Two different measures are computed using $f_0$ as the frequency of the negative instances in a node and $f_1$ as the frequency of the positive instances in that node.

  - **Gini impurity:** $f_0(1-f_0)+f_1(1-f_1)$
  - **Entropy:** $-f_0 log(f_0) - f_1 log(f_1)$

The decisions taken by RF models, while not intuitive, can be grasped analyzing the feature importance, so they are more comprehensible than other methods, such as NNs.

Single decision trees are easy to interpret and can be plot using binary trees. However, when we create linear combinations of trees, this clarity of interpretation is lost. Therefore, other techniques need to be applied. Obtaining the relative importance of each feature in predicting the outcome is often useful, as only a few of them are usually important [28]. For a single decision tree $T$, we can compute the importance $\mathcal{I}_f$ of a feature $f$ as follows:

$$\mathcal{I}_f(T) = \sum_{t \in T : v(s_t) = f} p(t) \Delta i(s_t)$$

where $t$ is a node with split $s_t$, $v(s_t)$ is the variable used in that split, $p(t)$ is the proportion of observations reaching the node $t$ and $\Delta i(s_t)$ is the increment of the impurity measure in the split of node $t$.

We can see in the previous formula that the feature importance can be extracted by aggregating the importance of each variable in the splits of the tree. Using the split criterion improvement at each node we can get the relative importance of a variable in a given tree.

If we normalize the importance of the variables of each tree and average the values obtained for each variable, then we get the estimate of the importance for each feature. This can help us identifying which features are being the most determinant in the model to take the decisions [29].

If we have $M$ trees we can compute the importance of feature $f$ with the following formula:

$$\mathcal{I}_f = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_f(T_m)$$

## 2.4 Distributed systems

Traditionally, the solution for processing bigger amounts of data was to scale up, which means upgrading to a better machine. However, this is very expensive and has an exponential cost. Instead, new approaches focus on scaling out, which increases the computing power by adding more machines to the grid. This means that high computer performance can be achieved just by using commodity machines. These computers are usually prone to errors, so implementing fault tolerance is required. Data needs to be replicated across the cluster, so in case of a failure the information is not lost.

Thousands of machines can be grouped in computing clusters, which are prepared to deal with Big Data. These distributed systems are usually very complex, as they require overhead regarding replication, fault tolerance and intra-cluster communication among the different machines. However, high level APIs for data processing and ML (such as H2O [30], Flink [31] and Spark [2]) have been created during the last years, making easier to work with Big Data. One of the biggest enablers for this fast adoption of Big Data technologies has been Hadoop [32].

First, in Section 2.4.1 we explain the properties of Big Data and explain why local computations pose an issue when dealing with it. Then, Section 2.4.2 focus on Apache Hadoop, the open-source software framework for distributed computing used in this work.

### 2.4.1 Big data

Traditionally, Big Data has been described by the 4 Vs, which capture most of its challenges:

- **Volume:** the big amount of data generated requires specific technologies and techniques.
- **Velocity:** the high pace of data generation makes necessary to do streaming analytics on the fly.
- **Variety:** the high variety of data types, including unstructured data, usually doesn't fit in traditional databases.
- **Veracity:** the quality of the data is often compromised due to the high speed and big amount of data being gathered. Controlling the trustworthiness levels of the data is very important to get good analytical insights to enable business decisions.

In this thesis our data sources have big volume, moderate velocity and a lot of veracity issues. Variety is not a big problem, as most of the data is structured. However, some useful information is stored as text input from customers, so it needs to be preprocessed accordingly to extract insights from it.

### 2.4.2 Apache Hadoop

Apache Hadoop is part of the Apache Project Foundation[‡‡], an American non-profit corporation, formed by a decentralized open-source community of developers.

Hadoop is a framework for distributed storage and processing that runs on computer clusters of commodity machines. It assumes that machines are likely to fail, so it implements automatic failure recovery.

At first, the aim of Hadoop was to run MapReduce [33] jobs. However, Hadoop 2.0 introduced YARN [32], which enabled other applications to run on top of it.

Below, the Hadoop Distributed File System (HDFS) architecture is described. Then, the most common Hadoop distributions are mentioned. Afterwards, the main components of the Hadoop ecosystem used in this work are briefly explained. Finally, we explain more in detail Apache Spark, the framework used for data processing and modeling in this thesis.

[‡‡] https://www.apache.org/.

### HDFS

The core of Hadoop lies on HDFS, its storage part, which was inspired by the Google File System (GFS) [34].

Regarding HDFS architecture we can distinguish three main types of nodes:

- **NameNode:** central machine of the Hadoop cluster, which keeps the tree of all files in the system and their location across the cluster.
- **Secondary NameNode:** machine in charge of checkpointing the NameNode which can be used for failure recovery.
- **DataNodes:** machines that store the HDFS data.

### Distributions

The release by Google of the MapReduce paper [33] led Apache to implement their own open software version. HDFS and MapReduce rapidly became the distributed file system standard in the industry. Following the simplicity of MapReduce new projects started on top of Hadoop and HDFS. That's why Hadoop 2.0 was released, which included YARN, a scheduler to support all different kind of jobs running on Hadoop. Right now, Hadoop comprises a vast ecosystem of open source projects by Apache that cover a wide variety of areas.

Setting up Hadoop can be cumbersome, so that's why different distributions have been created to ease up that process. Three main products are competing in this category offering an easy entry point for companies into the Big Data world.

- **Hortonworks:** open source platform based on Apache Hadoop which has created the most recent Hadoop innovations, including YARN.
- **Cloudera:** it combines a hybrid approach between proprietary and open software. They have the largest number of clients. Their core distribution is based on the open source Apache Hadoop, although they also count with proprietary solutions.
- **MapR:** they replaced HDFS with their own proprietary file system solution.

The distribution used at Qliro was Hortonworks, which has the downside of suffering from frequent bugs. Some issues can slow down a project, requiring you to wait for next releases that fix them. The advantage is that being open source, everyone can contribute and suggest further improvements.

### Hadoop ecosystem

A lot of technologies run on top of Hadoop, so its ecosystem is quite big. Below we mention the Hadoop components used in this thesis:

- **Apache HDFS:** The distributed filesystem used in Hadoop, which is based on GFS [34].
- **Apache Spark:** distributed programming framework used in this thesis. At the end of this section it is described in detail.
- **Apache Sqoop:** data ingestion tool used to transfer data between traditional databases and HDFS. It has been used to transfer Qliro's data sources from the production SQL databases into the Hadoop cluster.
- **Apache Hive:** simplifies making SQL-like queries on top of HDFS. The data used for this project has been stored in Hive tables, so they can be easily queried using Hive Hybrid Procedural SQL On Hadoop (HPL/SQL). Hive uses the Optimized Row Columnar (ORC) file format, a mix between columnar and row based storage systems, which is very efficient when querying data from HDFS.
- **Apache Ambari:** intuitive and easy to use web User Interface (UI) that provides seamless management of most applications that run in the Hadoop cluster. It also simplifies the cluster configuration and offers access to logs without having to SSH into the machines. The UIs of YARN and Spark are also accessible from Ambari, as well as several metrics that help monitoring the cluster's health.
- **Apache Zeppelin:** notebook-style interpreter for interactive analytics that supports Spark. It eases data exploration and visualization tasks. All HDFS files can be directly accessed from it.
- **Apache Oozie:** scheduling tool that concatenates jobs to be run periodically on top of Hadoop. The Oozie workflows are defined and coordinated using XML configuration files.
- **Apache NiFi:** real-time event processing platform to orchestrate data flows. Most of the data used in this project was updated from production systems using NiFi dataflows.

### Apache Spark

Traditionally, distributed computing was complex, as it had to handle the communication and coordination across the clusters' machines. MapReduce offered a high performance API that was very easy to use, enabling programmers to create efficient distributed jobs abstracting them from the distribution technical details. However, only two main operations (*map* and *reduce*) were supported, so jobs requiring iterative computations or complex logic couldn't be easily built on top of MapReduce. Spark was created as a framework for writing distributed programs that run faster than MapReduce by leveraging in-memory processing. It offered a a higher level API than MapReduce, with a wider range of operations supported, such as *filter* or *groupBy*.

The basic logic structure in Spark is the Resilient Distributed Dataset (RDD), which is immutable and can be persisted in-memory for data reuse [2]. The immutability of the RDDs allows Spark to express jobs as Directed Acyclical Graphs (DAGs) of operations performed on RDDs. This lineage graph can be stored and is used in error recovery to track down the parts of the job that need to be recomputed.

On top of RDDs, DataFrames (DFs) have been created. This abstraction supports SQL-like queries that eases the data scientists' work [35]. In Spark 2.0 Datasets where introduced, a new abstraction that adds type checks to DFs, making the code more comprehensible and maintainable.

Spark provides a programming API that can be used in Scala, Java, Python and R. Lately, it has been extended with more functionalities to support streaming (Spark Streaming [36]), graph processing (GraphX [37]) and distributed machine learning algorithms (Spark MLlib [38]).

Spark ML is the new version of the Spark machine learning API that leverages DFs. It is currently being ported from the previous project, Spark MLlib, which focused on RDD operations. However, not everything has been yet migrated.

Spark uses a master-worker architecture. The driver is the master that coordinates the worker nodes. These workers can run multiple executors. The level of parallelization of the Spark job is given by the number of executors used.

Each executor can run multiple tasks and data can be cached in the executor's memory to be reused in later stages of the execution DAG. As Spark makes the computations in-memory the amount of memory needed for each executor depends on the job and has to be specified to Spark when submitting the Spark job. Jobs that require more memory per executor than the amount assigned will fail.

## 2.5 Fraud detection

Traditional online payment solutions focus on credit card fraud detection. However, payments at Qliro are performed using the Swedish personal number. Hence, fraudsters supplant other people identity using their personal numbers, so credit cards are not involved at all. This type of fraud is known as online ID fraud.

Across the Nordic countries, the increase in popularity of e-commerce is attracting the attention of the criminals, who try to impersonate other customers to make fraudulent purchases. This type of fraud was more than tripled during the past 10 years, from 45 MSEK in 2006 to 142.4 MSEK in 2016 [39].

First, the different problems specific to fraud detection are stated in Section 2.5.1. Then, Section 2.5.2 describes the main components of an FDS. Finally, Section 2.5.3 focus on performance metrics for fraud detection.

### 2.5.1 Fraud detection problems

Fraud detection is a binary classification problem with two main particularities:

- **Cost structure of the problem:** the cost of a fraud is difficult to define and should be aligned with the business logic. Non-trivial matters such as reputation costs for the company or frauds creating cascading effect have to be considered.

- **Time to detection:** frauds should be blocked as fast as possible to prevent more future frauds.

- **Error in class labels:** fraudulent fraud claims and misclassified frauds by the investigators add noise to our data labels.

- **Class imbalance:** the number of frauds is much lower than the number of genuine transactions. This affects the ML algorithms, so different approaches need to be contemplated. Different solutions for this unbalanced classification tasks are analyzed in detail in Section 3.1.

  The imbalance rate in fraud detection varies across datasets. For example in the experiments reported by Dal Pozzolo in [3] only 0.15% of the transactions were fraudulent. However, this percentage is lower in our use case, incrementing the imbalance problem.

- **Non stationarity distribution:** the imbalance ratio and the characteristics of the frauds change over time, so a model that performs well for an interval of time can soon become obsolete and start yielding inaccurate predictions.

- **Alert-feedback interaction:** the frauds labeled by the investigators are received more frequently than the fraud claims. This means that the model can't be updated until the fraud claims are received, which is usually days after the frauds happen. Another problem is that the feedbacks contain bias towards the FDS in place.

### 2.5.2 FDS components

A FDS includes all the processes and infrastructure required for accomplishing fraud detection.

A typical FDS is composed of the following elements:

1. **Terminal:** transactions being blocked at the first level of the application. Checking if the account balance is sufficient would be part of this step.

2. **Transaction blocking rules:** rules designed by experienced investigators to block transaction requests before their authorization. These rules are running in real time and they are very tested. For example, one rule could be to block a transaction if the limit amount of the purchase is over a certain threshold, previously agreed upon with the merchants.

3. **Scoring rules:** rule-based classifiers (usually manually engineered) that assign a score in near real time. Different methods can be used. At Qliro, *user profiling*[§§] is performed at this stage to help in posterior manual checks made by the investigators. The data used for this step is usually aggregated per customer. These rules are easy to deploy, but hard to maintain due to the non-stationarity nature of the data.

4. **Data driven model:** layer that relies on a ML predictive model that captures complex relationships present in the data. The main problem of these models is their interpretation. Most ML models are a black-box, so rules easy to understand by investigators can't be extracted. However, they can handle a much bigger amount of features compared with previous stages because no manual labor is involved.

5. **Investigators:** fraud experts that check the alerts reported by the scoring rules and the data driven model. Due to time and cost limitations, only a small amount of possible frauds can be checked in detail. Manual research and phone calls are usually involved in this process and experienced investigators usually follow their intuition.

[§§] Supervised profiling consists in grouping observations into profiles, usually by using a different set of rules for each profile.

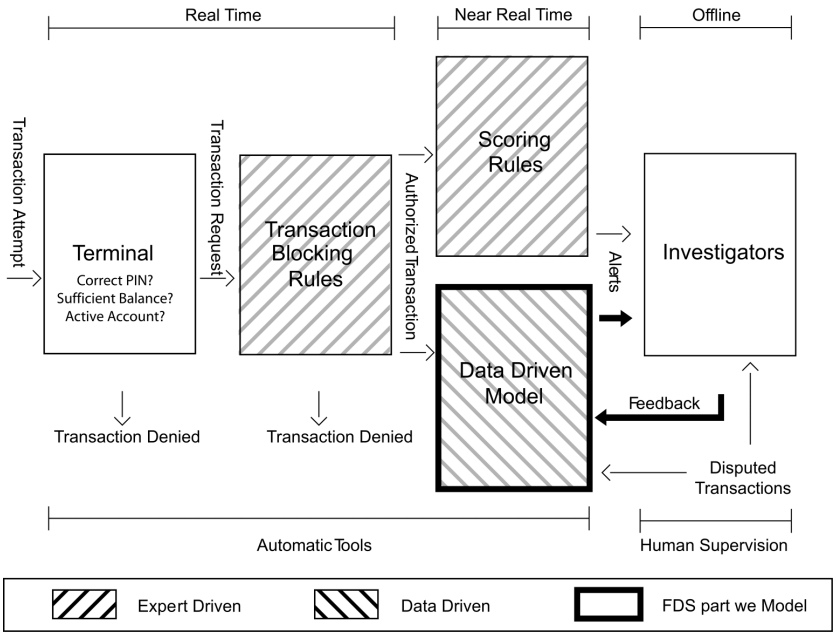



Figure 2.4: Fraud detection system pipeline (figure taken from [3]).

In Figure 2.4 we can see how the different stages explained before are connected.

Expert driven systems are updated manually, while data driven systems are updated automatically. Typically, large train samples are needed to train accurate models, so depending on the volume of the transactions the frequency of the models retraining can vary.

### 2.5.3 Performance metrics

As described in Section 2.3 choosing the performance measure to use is not trivial.

#### Catch rate

Investigators usually use the catch rate to measure their performance. This catch rate is the recall regarding the transaction costs:

$$\text{Catch rate} = \frac{C_{TP}}{C_{FN} + C_{TP}}$$

$C_i$ is the sum of the costs of the different transactions that belong to the element $i$ of the confusion matrix. This cost is the purchase amount and it varies across transactions.

However, this catch rate doesn't have into account the cost of the investigation derived from the FP cases in our ML model. Using a cost-based confusion matrix can solve that problem.

### Cost-based confusion matrix

When using monetary metrics a cost matrix is created, where each element of the confusion matrix has a cost associated [40, 41]. However, some authors consider that is better to use benefit instead of costs to avoid using different baselines [42].

A possible solution to avoid employing different cost baselines for the losses is to use the normalized cost of savings [43].

Another problem to create cost metrics is related to whether the cost should be fixed or transaction dependent. If it is fixed, then small and large frauds are given the same importance, as fraudsters usually test frauds with small amounts. Using a variable cost, however, quantifies the real loss of the company, so it is more aligned with the business goals. The cost of a missed fraud is usually the cost of the transaction, while the cost of false alerts is the cost of the investigation process. This cost is usually low compared with the frauds, but in case of a large number of fraud alerts being generated this amount can be significantly high. False alerts also increase the risk of investigators making mistakes, which can lead in further loses due to blocking valid transactions.

Usually, after the first fraud attempt has been successful fraudsters try to spend as much money as possible. Hence, it might make sense to include in the cost of a transaction the amount of money available in the account to make extra purchases, as this captures the amount of money potentially susceptible of being fraudulently used.

### Ranking based metrics

Some authors state that the fraud detection task is not well defined in terms of classification [44], and that ranking based metrics are better fit for this problem. The goal in this case is not to predict accurately each class anymore, but to return an accurate ranking for the frauds. A measure to use in this context is the Average Precision (AP), which can be computed from the alert precision($P_k$) and the alert recall($R_k$) [3]:

$$P_k = \frac{TP_k}{k} \quad R_k = \frac{TP_k}{N^+}$$

$$AP = \sum_{k=1}^{N} P_k(R_k - R_{k-1}))$$

$TP_k$ is the number of true positives in the first $k$ ranked transactions, $N^+$ is the total number of frauds in the dataset and $N$ is the total number of observations in the dataset.

AP approximates the AU-PR and gives us an overall measure, but the alert precision (also called precision at $k$) can focus on only a subset of the first $k$ transactions and has emerged as a standard accuracy measure for FDS [45].

# 3

# State of the art

*"If you only read the books that everyone else is reading,*
*you can only think what everyone else is thinking."*
– Haruki Murakami, *Norwegian Wood*

This chapter reviews the methods available in the literature to solve the main challenges in fraud detection presented in Section 2.5.

First, in Section 3.1 the different techniques available to tackle the imbalance problem are presented. Then, in Section 3.2 some approaches for handling non-stationary data are discussed. Section 3.3 analyzes both previous problems combined. Finally, Section 3.4 discusses the fraud detection problem from the perspective of alert-feedback interaction systems.

## 3.1 Class imbalance

Unbalanced datasets make harder to train an accurate ML model because algorithms may have difficulties learning concepts related with the minority class [46]. Hence, the natural distribution of our data is not necessary the best one to use for training. However, as stated in [47], the distribution of the classes in the training set differs from one dataset to another, and in the case of substantial class imbalance, a $50:50$ distribution is not the best one to minimize the error rate. Changing the natural distribution in favor of the underrepresented class helps countering the imbalance problem [48].

There are two main approaches to solve the class imbalance problem: data level methods explained in Section 3.1.1 and algorithm level methods analyzed in Section 3.1.2.

The methods presented here try to solve the unbalanced problem between classes. However, class imbalance could also exist within a class when there are clusters of underrepresented rare cases, which is related to the *small disjuncts* problem explained in [49]. This article claims that the small disjuncts are the real problem hindering good model performance, and propose a technique to tackle both problems together. Experiments in real-world datasets with this approach show an increase in the model performance. However, other works argue that the main problem of class imbalance has to do with overlapping classes and noisy class boundaries [50].

### 3.1.1 Data level methods

Data level methods focus on preprocessing the data before the training to correct the imbalance, so the ML algorithms can increase their performance.

These methods can be subdivided depending on the type of techniques employed to correct the imbalance: sampling methods change the dataset distribution, cost based methods use class misclassification costs when sampling and distance based methods use distances to remove noisy borderline examples. Hybrid strategies can be also created by combining different techniques [51].

#### 3.1.1.1 Sampling methods

These methods are usually simpler because they don't have into consideration any class information when adding or removing samples. Some of the main approaches in this category are explained below. They can be used in conjunction with each other and the performance of the different combinations varies depending on the specific characteristics of the dataset.

##### Undersampling

This sampling method consists in downsizing the majority class by removing random observations [52]. It makes the assumption that the majority class contains redundant information, so removing some cases at random doesn't affect the performance significantly. However, as the instances are taken at random, the risk of removing important information is still present.

Unsupervised ML algorithms can be used to select the instances to remove, so samples from all significant clusters are kept [53].

Undersampling is very popular due to its simplicity and because it speeds up the training step.

#### Oversampling

This sampling technique focuses on reducing the imbalance ratio by increasing the number of cases from the minority class. However, the risk of overfitting increases because our model is biased towards the minority instances [52].

The downsides are the absence of new information in the oversampled minority instances and the increase of the training time.

#### SMOTE

This technique is a variant of oversampling where instead of replicating the minority cases, some synthetic samples are created. These new samples form clusters around each minority observation [4].

The main drawback of this approach is that the new synthetic cases are created without having into account neighboring majority instances, so the region of overlapping examples between classes is increased. Some variants of Synthetic Minority Over-sampling Technique (SMOTE) alleviate this problem [54, 55, 56, 57].

Another problem is that the $K$-nearest Neighbors (KNN) algorithm [58], which is used for creating the synthetic samples, requires tuning to determine the optimal value of $k$.

Categorical features also pose an issue for the KNN algorithm, which only works with continuous features. A possible solution is to use SMOTE-NC [4], which supports a mix between categorical and continuous variables, or SMOTE-N [4], when only categorical features are present. Some studies argue that SMOTE is less effective than random undersampling [59].

In Figure 3.1 this technique is presented along with oversampling and undersampling.

### 3.1.1.2 Cost based methods

These techniques sample the data having the costs of misclassification into account. If $C_{ij}$ is the cost of the class $i$ being predicted as $j$, we can sample the data proportionally regarding the misclassification ratios, so the FP and FN costs will be considered ($C_{1,0} = C_{FN}$ and $C_{0,1} = C_{FP}$) [42].

In the case of fraud detection the cost of false positives is usually lower than the cost of false negatives ($C_{FP} < C_{FN}$). Observations from the minority class can be oversampled using $\frac{C_{FN}}{C_{FP}}$ as a ratio. If we want to perform undersampling of the majority class, $\frac{C_{FP}}{C_{FN}}$ can be used instead. However, [42] warns about the risk of using economically incoherent costs. It also states that changing the balance of the dataset has little effect in Bayesian or decision tree classifiers, so keeping

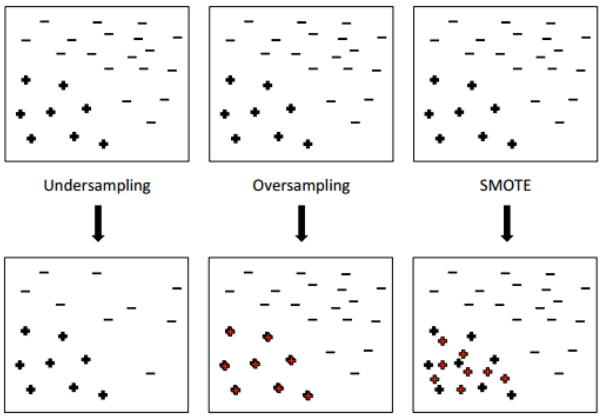



Figure 3.1: Sampling methods for unbalanced classification (reprinted from [3])

the original training set distribution and making decisions based on the classifiers' probability estimates would be preferred.

#### 3.1.1.3 Distance based methods

These methods use distances to remove borderline examples between the two classes, reducing the noise of the class boundary and improving the classification performance.

If categorical features are present in our dataset, the definition of the distance between vectors needs to be adapted [4].

In [60], Tomen links are used to undersample the majority class, removing the majority cases close to the minority observations. This is usually preferred to random undersampling, specially when dealing with noisy datasets [61].

Condensed Nearest Neighbors (CNN) [62] subsamples the majority class by removing the observations that are correctly classified by KNN with $k = 1$. However, as this keeps the samples close to the decision border, it is very sensitive to noise. Different variants of the CNN algorithm have been created by using a different values of $k$ when applying KNN [63, 64].

One Sided Selection (OOS) [61] combines Tomen links to reduce the noise from the class boundary with CNN to remove the cases far from the decision border.

### 3.1.2 Algorithm level methods

These techniques focus on modifying existing algorithms to deal with the class imbalance problem without changing the class distribution of the training set.

Weiss argues that these algorithmic approaches are preferred to data level methods, as they avoid biasing the model towards one class [65]. When applying sampling techniques this bias is desired, but in other scenarios (e.g. when adding more minority examples using active learning) that is not the case. Instead of altering the class distribution, Weiss advocates for removing that bias modifying the decision threshold of the algorithms.

Two main approaches can be distinguished: imbalanced learning, which tries to improve the accuracy of the minority class, and cost-sensitive learning, which focuses on minimizing the classification costs.

### Imbalance learning

Skewed class distributions have a negative effect on traditional ML algorithms [66]. Some classification algorithms affected by this problem are decision trees [23], SVMs [67] and NNs [68].

Ensemble learning can alleviate the imbalance problem. Ensembles combine different classifiers to make predictions. Below some of the main types of ensembling techniques are briefly explained:

- **Bagging:** combination of weak classifiers to obtain a strong one. Each model is trained with an independent bootstrap sample of the training set. This way, each model is fed with a different dataset. The different models' predictions are aggregated by voting [69], so the class with the most votes is selected.

  This technique makes the assumption that the learners are not stable, which means that small changes in the training set lead to significantly different classifiers.

  A powerful ensemble algorithm based on bagging is RF (explained in detail in Section 2.3), where several decisions trees are trained using different bootstrap samples from the original training set [27].

- **Boosting:** instead of bootstrap sampling, boosting assigns a weight to each data instance. The higher the weight, the bigger influence that it has in the model. A final model is constructed by aggregating the classifiers using their votes as a function of their accuracy [70]. This method doesn't require weak classifiers like in bagging, but implicit instability in the models is also needed. AdaBoost was the first practical boosting algorithm and still remains being one of the most widely used [71]. In [72], experiments reported that boosting has better performance than bagging. However, this technique created severe degradation in some datasets.

XGBoost [73] is a modification of GBT widely used in ML challenges to achieve state-of-the-art results.

EasyEnsemble [74] solves the class imbalance problem by training several models with different independently sampled datasets. Those classifiers are combined using AdaBoost [75].

Boosting can also be used in combination with oversampling, such as in SMOTEBoost [76] and JOUS-Boost [77].

- **Stacking:** this technique consists in combining the predictions of several classifiers to construct a new dataset. A combiner model (usually a logistic regression) is trained with that data to get the final predictions [78]. The main problem with this approach is that the new dataset also suffers from class imbalance. In [79] the authors propose a fraud detection method that uses stacking in combination with bagging, which alleviates the unbalanced problems of the stacked output.

- **Cascading:** Viola introduced this approach in [80] for the face detection task. The idea is to chain classifiers in order to accelerate the detection process. Each classifier should reject as many negative candidates as possible [81]. A necessary requirement for those classifiers is to have a very low amount of FN cases. However, in the case of complex problems where boundaries are not well defined, such as fraud detection, this requirement is not satisfied.

  BalanceCascade [75] cascades several classifiers removing correctly classified instances from the majority class, which are considered to be redundant.

#### Cost-sensitive learning

These approaches introduce misclassification costs during the learning phase.

Metacost [82] is a general framework for transforming any classifier into a cost-sensitive one.

One of the main problems with this approach is that costs are hard to estimate in many applications [83].

## 3.2 Non-stationarity

In machine learning it is usually assumed that the training and test sets follow the same distribution. However, in non-stationary problems there is a mismatch of the training and test distributions.

Two main causes can be distinguished: Sample Selection Bias (SSB), explained in Section 3.2.1, and time evolving data distributions, which are discussed in Section 3.2.2.

### 3.2.1 Sample Selection Bias

SSB occurs when the selection process of the training samples is biased. This means that the selected data is not representative of the whole population.

Three different kinds of SSB can be differentiated depending on the type of bias.

- **Class prior bias:** the bias is conditional to the class we try to predict [84]. For example, if a payment company doesn't store rejected frauds, the dataset suffers from this type of bias.
- **Feature bias:** the bias is independent from the target variable [85]. An example of this would be a company only storing information of customers from a certain age range, so not all customers' ages are represented in the training set.
- **Complete bias:** class prior bias and feature bias coexist. This is the most general case and can be solved with the method proposed by Heckman, which awarded him the Nobel prize in economics in 2000 [86].

SSB can be introduced on purpose using sampling techniques to solve the class imbalance problem, as discussed in Section 3.1. Here, the test set maintains its original distribution, while the training set distribution is altered to achieve better predictive results.

### 3.2.2 Time-evolving data distributions

In this case, the mismatch in the distributions is caused by a difference in the data generation process. For example, in fraud detection, the behavior of the fraudsters change over time, so the distribution of the training data will be outdated as time passes. This problem is also known as Concept Drift (CD) [87] or *Dataset Shift* [88].

In some cases, the CD only changes the distributions within the class, leaving the class boundary unchanged (this problem is known as *covariate shift*). However, other times this class boundary is modified, so previously well-calibrated classifiers become obsolete. Distinguishing which one affects our data is problematic because in real-world datasets we usually lack information about the data generation process.

Keeping only up-to-date samples by removing obsolete data can solve the CD problem [89], but getting rid of information that could improve the predictive power is also problematic (this is known as the *stability-plasticity* dilemma [90]).

Two approaches to solve the CD problem can be identified [91]:

- **Active approaches:** the data distribution changes abruptly, so the model only needs to be updated when a change in the distribution is detected.

  The imbalance class problem increases the difficulty of detecting when the distribution changes, as it is required to wait until enough samples of the minority class are gathered.

- **Passive approaches:** the data distribution changes slowly over time.

  A possible solution is to train classifiers over a time window [92]. However, it is important to set a window size that defines the forgetting rate, which should match the rate of change in the distribution [93].

  Other methods try to avoid forgetting previous knowledge, relying on ensembles. All the classifiers are combined using a weighted majority voting, so the latest models are enriched with previous knowledge [94]. These weights should also match the rate of change of the distribution. Other approaches replace obsolete models in order to improve the adaptation to the CD [95].

## 3.3 Non-stationarity with class imbalance

Studies that focus on unbalanced non-stationary distributions are scarce, as it is a recently new domain [3]. Classifiers require large samples to train accurate models, so waiting until we have enough data restricts the model retraining time intervals. Unless the volume of minority class examples is very high, this retraining is performed batch wise.

Sampling methods can be applied to reduce the class imbalance and ensembles can help reducing the CD. Propagating minority cases when training models in different time intervals can help reducing the dependency from resampling methods, alleviating the associated bias. Some works combine undersampling with minority class propagation [96], while others also propagate previously misclassified majority observations [97].

Dal Pozzolo proposed in [3] the Propagate and Forget approach. Minority transactions are propagated to reduce the imbalance ratio and a small number of majority transactions is also kept. A model is created applying sampling techniques to deal with the imbalance. Then, this new model is combined with the latest old ones in an ensemble.

When new batches are received with high frequency, propagating the instances might be costly, so using Hellinger Distance Decision Trees (HDDTs) [98] instead of the traditional C4.5 decision trees [99] can boost the performance and the computation time of the predictions [3].

## 3.4 Alert-Feedback interaction systems

Labeling is problematic in most fraud detection cases. As mentioned before in Section 2.5, the labels contain certain degree of uncertainty.

In a typical FDS two different types of frauds can be found:

- **Customer claimed frauds (delayed samples):** reported by the customers being impersonated by fraudsters.
- **Reported frauds by the investigators (feedbacks):** fraudulent cases found by the investigators on a daily basis.

Learning from feedbacks and delayed samples is different. While feedbacks provide recent up to date information, some delayed samples could be outdated once received. This means that a model using feedbacks can be retrained more often than one using only delayed samples.

The feedbacks depend on the suspicious frauds sent to the investigators for review by the FDS, so they are not independently drawn from the transactions' distribution. This means that a model trained with them will only learn from the subset retrieved by the the classifier or rules used in the FDS. This can lead to bad performance when applied to all the observations.

These two subsets are likely to have different distributions, so [3] proposes creating different models to detect the alerts and the feedbacks and ensemble them. The main advantage of this approach is that the feedbacks' model can be retrained with a higher frequency, so the final model adapts better to the CD.

# 4

# Experimental setup

> *"The language in which we express our ideas has a strong influence on our thought processes."*
>
> – Donald Ervin Knuth, *Literate Programming*

In this chapter the set up used for building the fraud detection models and performing the experiments is described. First, Section 4.1 specifies the technical characteristics of our Hadoop cluster. Then, Section 4.2 refers to the software used and the development decisions taken in our project.

## 4.1 Hardware

The work was realized using part of Qliro's Hadoop cluster, consisting on one NameNode, one Secondary NameNode and four DataNodes. Those machines had the following properties:

- **OS:** centos7 (x86-64*)
- **CPU:** 20 physical cores and 40 logical cores.
- **RAM:** 128 GB.
- **Disc:** 7.5 TB in the DataNodes and 2.5 TB in the NameNodes.

* 64-bit version of the x86 instruction set. Also known as x64, x86_64, AMD64 or Intel 64.

## 4.2 Software

In this thesis we have used an Apache Hadoop distribution based on YARN by using Hortonworks Data Platform (HDP), which provides access to all the services in the Hadoop ecosystem. The different versions of the Hadoop components used are listed in Appendix A. More information about these components and their function can be found in Section 2.4.2.

At the beginning of the project all the code was developed using Zeppelin, which is very useful for data understanding. However, once the project started to increase in complexity in the data preparation and modeling step, it was necessary to keep the code organized by using an IDE, which had proper syntax highlighting and auto-completion features. The project was set up using Git, a Version Control System (VCS) used for tracking changes in files of the project and for creating branches that made it easier to separate production and development code.

In this thesis, Spark jobs were written using Scala, as this is Spark's native language, and provides type safety and great performance. As this language is built on the functional paradigm, programs usually require less code and are more readable. Python was the second option considered as it is a great language with a lot of very useful built-in libraries, but as everything in the Hadoop world is JVM based, it made more sense to use a language built on top of it. Also, as it supports dependency management based on Maven repositories (using *sbt*), the code is easier to maintain.

The OS used in our local machines was Windows 10, so MobaXTerm was used to simulate a Linux terminal. This simplified the process of making SSH connections with the machines in our cluster, something needed to submit Spark, Sqoop or Oozie jobs.

# 5

# Methods

> *"Taking a new step, uttering a new word, is what people fear most."*
>
> – Fyodor Dostoyevsky, *Crime and Punishment*

This chapter presents the research design decisions taken to create the automatic data driven component of Qliro's FDS, performing experiments to determine the best techniques to solve the fraud detection problems presented in Section 2.5.

It is structured using the CRISP-DM methodology explained in 2.2. Each section refers to one of its stages. Although they appear in order, there were several iterations and feedback loops between them. Performing a data science project without these feedbacks is extremely hard because the stages are interconnected, so the results of some experiments can be used to refine previous stages.

## 5.1 Business understanding

As mentioned in Section 1.1, Qliro had in place a FDS that was lacking a data driven component. This made this system hard to maintain.

The main goal of this project is to detect online purchases with high probability of being fraudulent and send them to the investigators, so they can review them. However, the number of cases sent for review can't be very high due to the limited amount of time and resources available.

The most important business metric to track the performance of the FDS is the catch rate, defined in Section 2.5. The precision of the predictions was also considered because too many false alerts increases the investigation time and cost.

However, it didn't exist a clear way to unify these two measures from the business point of view.

To understand the business requirements of the project the following tasks were performed:

- **Meeting with Qliro's head of fraud:** he explained in detail how Qliro's FDS worked and the main challenges the fraud team was facing.
- **Peer review of the daily work of a fraud investigator:** sitting next to an investigator and observing the steps required to detect fraudsters was extremely valuable to get domain knowledge insights.
- **Literature study:** research regarding FDSs to put all the fraud detection activities carried on at Qliro in a theoretical context.
- **Open communication channel with investigators:** this way a lot of doubts were solved without spending time doing research to find the answers.
- **Presentation of findings to investigators:** we presented some possible business objectives to the investigators. They pointed out some business concepts we had misunderstood, so they were corrected before the development started (e.g. at the beginning we thought we wanted to prevent frauds in credit card payments, while in reality we were dealing with online ID fraud).

Using the insights extracted from all those tasks we defined our business objective: to detect frauds being ignored by Qliro's FDS, which were being either manually tracked by investigators or missed. Hence, catching frauds currently captured by the FDS wouldn't add business value.

Optimizing our model to only solve those cases would create a bias towards their current system, so we didn't take that approach. A better alternative was to remove the irrelevant cases from the predicted frauds retrieved by our model before handing them to the fraud team. This would require that our model detected a significant amount of new cases. This approach could enable removing complex rules that are almost completely covered by our model, reducing the complexity of the rule-based part of the FDS.

As we had Big Data requirements, the project was performed in the Business Intelligence (BI) team, which had the Hadoop infrastructure needed to develop the fraud classifier.

With all the previous considerations in mind, we agreed to deliver a *reasonable* amount of new potential frauds every day. The amount to deliver wasn't clear in business terms. Hence, we decided to send them with a confidence probability, so the fraud team could filter them using a threshold to match the precision and recall

values they wanted to obtain. This was a good solution for the fraud team because the daily amount of time available for reviewing varied, so different thresholds could be used depending on the specific circumstances.

## 5.2 Data understanding

The data understanding phase starts with the data collection. Afterwards, tasks to obtain the first insights about the data and identify data quality issues are performed.

Having a clear understanding about the business problem to solve, we still needed to locate the data required to achieve our business objectives. The different data understanding activities are summarized in Figure 5.1. In this diagram we observe that first we collected the data needed from Qliro's SQL server. Then, that data was explored and quality issues were detected.

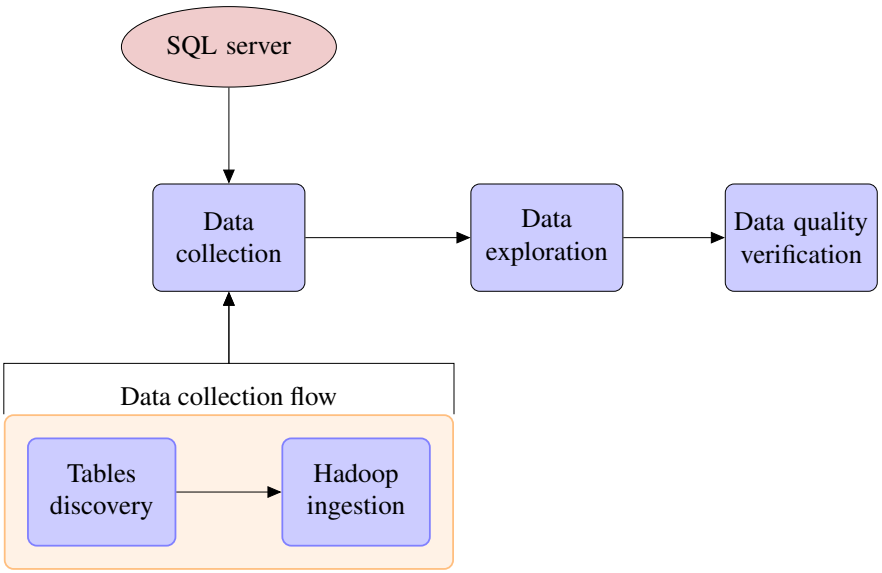



Figure 5.1: Data understanding flow.

### 5.2.1 Data collection

These tasks comprise the discovery of the tables needed and the ingestion of that data from SQL into our Hadoop cluster.

#### Tables discovery

Several SQL databases, each with a large number of tables, were present at the company, so the first step was to identify which ones could be used to detect the frauds.

The fraud team pointed out several potentially useful tables. However, some contained information that was not historized*, so they couldn't be used for ML purposes.

One of the most important tables used contained hundreds of columns with information about all the purchase orders. Information about the decisions taken by the fraud team was also present in other tables, so we could compare our model with their past predictions.

#### Hadoop ingestion

Before analyzing the data it was necessary to import it into our Hadoop cluster.

The data from each SQL table was moved into two Hive tables: one in raw text format† and other in ORC format optimized for querying in Hive.

Ingesting the data into those Hive tables had two steps:

1. Export the data using Sqoop from the SQL tables into Hive text external tables (Snappy compression was used to reduce the size of the files).

2. Insert the data from those external tables into the Hive ORC tables using HPL/SQL statements.

### 5.2.2 Data exploration

Once the data was in Hadoop, it was analyzed using Spark in Zeppelin and ad-hoc Hive queries.

It is worth noting that some of the data exploration tasks were performed after the data selection step explained in Section 5.3. The data preparation and its understanding are highly tied, so having a feedback loop between them is key to increase the predictive power of the selected features.

The data exploration tasks realized were: data description, tables exploration, analysis of time distributions, correlation analysis and Principal Component Analysis (PCA).

* If the value of a feature for a customer changes over time and those changes are not stored, then we don't have that data historized. † Raw text if the default format in Sqoop. It is also possible to import data directly into ORC format using HCatalog, but it is more complicated and it doesn't support schema changes.

### Data description

Descriptive summary statistics (such as mean, max value, min value or standard deviation) were obtained for the different continuous features.

For categorical features, the distinct number of possible values was computed. Some features had too many possible values, so they needed either to be discarded or preprocessed. Looking at the possible values of each category helped understanding its meaning (their names were usually not very informative) and detecting quality issues. Also, some categories appeared as integers, which needed to be decoded to be understood.

### Tables exploration

Relationships between tables were explored. Some of them contained old customer purchases that didn't exist in other tables that had been created more recently. Documentation was scarce, so a lot of time was spent exploring the semantics of the different tables.

### Analysis of time distributions

Time was an important variable in the reservations, so the distribution of frauds and non frauds was studied across the years. The time distribution of some features was also analyzed.

### Correlation analysis

Some columns were correlated, while others didn't have any statistical significance.

The correlation matrix was used to detect highly correlated continuous features.

Using the chi-square test* columns that didn't have statistical relevance in predicting the target variable were removed.

### PCA

PCA† was used to reduce the dimensionality of the feature vectors, so they could be plotted in two dimensions. This helped visualizing the boundary between frauds and non-frauds.

* The chi-square test is a statistical test of independence to determine if two variables are correlated. It can be used to select the most relevant features if we apply the test between the target variable and the different features. † PCA is a statistical procedure that converts a vector with possibly correlated variables into another vector with less variables that are not correlated, called principal components [100].

A two dimensional plot using the first two components obtained in the PCA analysis is presented in Section 6.1. Other components might be more informative, but as non supervised approaches are out of the scope of this thesis, a more rigorous analysis was not performed.

### 5.2.3 Data quality verification

These tasks were related with solving the data quality issues detected in the data exploration. The problems encountered were: null values, invalid categories, consistency issues, confusing semantics and duplicated information.

#### Null values

The main quality issue was the big amount of nulls because a lot of features were only filled when some conditions were met. This posed a problem, as it added a lot of noise. We replaced the null values with zero in continuous features, and with a new category for null values in the case of categorical features. Some columns that contained a lot of nulls could be merged with others that had a similar meaning to reduce them.

#### Invalid categories

Some categories were full of wrong values, but this didn't happen very often. When most of the values were noisy that category was removed. In other features, most of the values were the same, so they were also discarded.

#### Consistency issues

A consistency check with the original SQL sources to detect if the data was ingested correctly was performed. The total amount of rows and the content of the columns was analyzed. Thanks to this check, we detected data being inserted incorrectly into the ORC tables due to schema discrepancies. This check allowed us to correct this issue early in the project, saving a lot of time.

#### Confusing semantics

The meaning of some variables was not intuitive and understanding all the features was complicated because we had hundreds of them. Some feature definitions changed over time, so values from the past sometimes had a different meaning, adding noise. One solution was not to consider those features. However, in some cases they were very important, so using them was preferred. Working only with a

subset containing the latest data helped solving this problem because the changes in the meaning of the features were reduced.

#### Duplicated information

A feature could be present in different columns with similar values because they were obtained from different sources.

If both contained useful information merging them was the solution. This merging could be done either by averaging the values if they were continuous or using more complex techniques if they were categorical.

If one column contained good quality data, we could remove the others, specially if they were noisy, as merging them could decrease the overall quality of that feature.

## 5.3 Data preparation

The data preparation stage transforms the data available into datasets that can be used to train and evaluate ML models. A good data understanding helps selecting the most adequate features, engineering new ones, and cleaning the data. The data preparation is crucial because the performance of the ML model depends on the data used for its training.

Two main subtasks were performed in this stage: the data preprocessing, consisting on cleaning and selecting the relevant features, and the data sampling, where techniques to tackle the class imbalance problem were applied.

The data preparation and modeling are interconnected, so first we are going to explain how the flow of both of them look like. Then, we will describe the different parts of the data preparation step.

In Figure 5.2 we can see a diagram with the flow of the data preparation and the data modeling. First, data is preprocessed and the features to consider in the model are selected. Then, the training and test datasets are created using a time interval $T$ from our data.

The generated training data is fed into a ML pipeline that outputs a pipeline model*. This model makes predictions based on what it learned from the training data. Then, test predictions are obtained feeding the test set to that model. Different metrics are computed from these predictions to quantify how well our model classifies the fraudulent transactions. All these tasks are part of the data modeling step and are explained in detail in Section 5.4.

* The Pipeline API facilitates the creation of ML workflows concatenating different preprocessing and modeling tasks, so they can be cross validated together. A pipeline model can both transform the raw data into a suitable format and run the predictions.

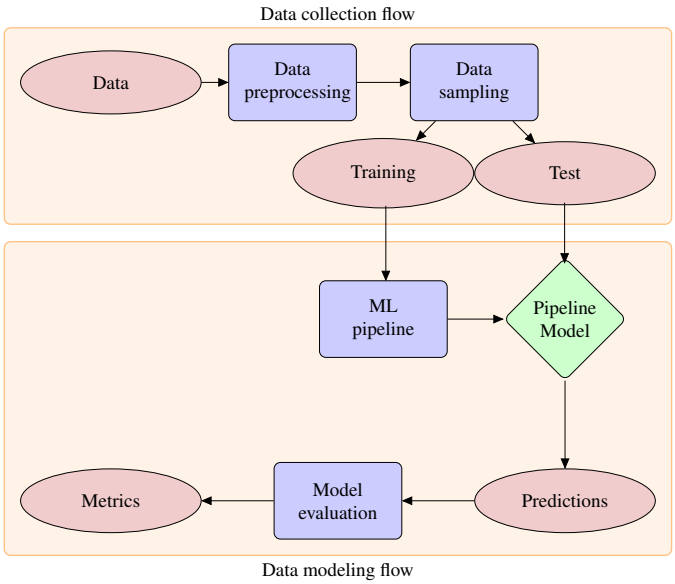



Figure 5.2: Data preparation and modeling flow.

The feedback obtained from the test metrics can be used to improve the ML pipeline. In this thesis a large number of models have been created and evaluated, incrementally refining them and obtaining more accurate ones.

### 5.3.1 Data preprocessing

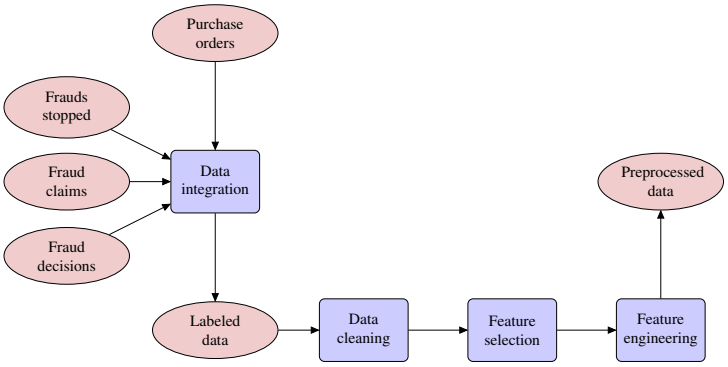



Figure 5.3: Data preprocessing flow.

After ingesting the data into our Hadoop cluster, we needed to integrate the different sources, clean the data, select the relevant variables, and engineer new

features. Figure 5.3 summarizes this process.

Once we have created our features and the fraudulent transactions are properly tagged, the preprocessed data is fed into the dataset creation step, where the training set and test set are created.

### Data integration

The different useful tables detected needed to be joined together. The data with information about purchase orders was *left joined*[*] with the tables that had information about the frauds and the decisions taken by the fraud team. Three main tables were used to label the frauds:

- **Fraud claims:** claims made by the fraud victims. Some of these claims could be fraudulent themselves, but this is rare and very hard to solve, so it has not been dealt with. The claims are received days after the fraud attempts, and they can be also present in the frauds stopped or fraud decision tables if the fraud team caught them.

- **Frauds stopped:** frauds not caught by the scoring rules from Qliro's FDS, but which were stopped by the investigators who detected them via manual searches. Most of these cases are fraudulent, so they were labeled as frauds, although the risk of them being mislabeled is slightly higher than in the previous case.

- **Fraud decisions:** contains the decisions taken by the investigators on the cases detected by the scoring rules of the current FDS. This table contains a lot of cases labeled as suspicious which are not fraud. The investigators tagged the cases with different codes depending on their confidence. The cases with the highest risk code have been used, as their noise is similar to the frauds stopped. This table combined with the stopped frauds can be used to compare our model performance with the fraud team predictions.

### Data cleaning

All the data quality issues were tackled here. Features were converted into continuous or categorical. The null values were replaced and the invalid categories were removed. The correlation matrix and the chi-square test were also used to detect unnecessary features.

[*] The left join is an operation that matches records from two tables, keeping all the results from the left one and filling the values with nulls if there is no match.

#### Feature selection

Once the frauds were labeled and the data was cleaned, we selected the features with high predictive power. To determine the best ones, models with different subsets of features were created and the feature importance of the obtained models was analyzed (this feature importance is obtained in the data modeling explained in Section 5.4.2).

#### Feature engineering

New features were created to improve the model performance using Spark User Defined Functions (UDFs)*.

For example, a mail provider feature was created, so an UDF needed to be created to extract the provider from the mail address.

Columns with the same meaning were combined. For instance, two string columns with the same semantics coming from different sources could be compared to detect if they were the same or not. This comparison was performed using the Levenshtein distance† of the strings considered, normalized by their average length. Two words were considered the same if that value was below a certain empirical threshold. This technique was necessary because typos were frequent. If two sources had different values, this could be an indicator of fraud.

Features aggregated by customer were also engineered. However, the customer ID was removed to avoid an over-optimistic testing. In real-world cases once an ID is flagged as fraud it is black listed, so we don't encounter the same fraudulent ID twice. The approach taken treats each purchase order independently by removing its customer ID, so information about its ID is only considered in the augmented features. This mixed approach is mentioned in Section 2.3.4.

### 5.3.2 Data sampling

In this step we created the training and the test set, used for training our model and evaluating its predictions.

The validation set was created from the training set when doing CV, as explained in Section 2.3.2. All the different data sampling techniques to deal with the imbalance are also applied in this step.

Stratified sampling have been implemented, so we can control the ratio of frauds in the training and test set. If random sampling is performed instead, most of the instances selected would be non frauds due to the class imbalance.

The stratified sampling process is summarized in Figure 5.4.

* UDFs are column-based functions that can be defined to extend the Spark SQL's data selection language. † The Levenshtein distance measures the minimum number of single-character edits between two words.

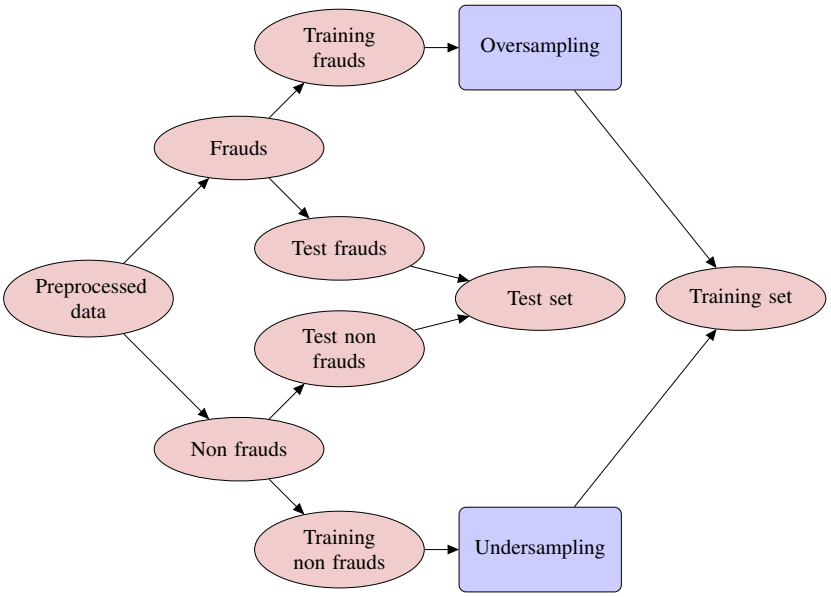



Figure 5.4: Data sampling flow (stratified sampling).

The data is first divided into frauds and non frauds. These subsamples are split into 70% training and 30% test. The frauds and non-frauds from the test subsamples are joined, as we want to preserve the original ratio for evaluating the performance of our algorithm. However, for creating the training set, the frauds are oversampled and the non frauds are undersampled. When creating the samples, we specify the oversampling ratio and the imbalance percentage we want to obtain. The level of undersampling applied in the training non-frauds is obtained from those two parameters.

Two different oversampling techniques have been applied:

- **Duplicating the frauds:** each fraud is replicated as many times as specified by he oversampling ratio.
- **SMOTE-NC**: modification of SMOTE to support categorical features. This algorithm has been implemented in Spark with some minor modifications to handle integer features. For example, if the oversampling rate is 80, the original frauds are kept and SMOTE-NC adds synthetic samples from them until the size is 80 times the original sample.

Experiments to select the most appropriate value for the oversampling rate and the unbalance rate have been carried on. Models obtained from those experiments are compared in Section 6.3. Using no oversampling was also compared against both random oversampling and SMOTE-NC. Cases without undersampling are

not considered in our experiments because our imbalance ratio is so big that the resulting models would predict everything as non-fraud.

The size of these datasets constructed was moderately big. The exact number can't be disclosed due to privacy concerns, but it was on the order of a few dozens gigabytes. Spark can handle bigger volumes of data, so if the size of the datasets increased, more resources could be assigned to the Spark jobs.

## 5.4 Modeling

In the data modeling, the training data is transformed into a feature vector supported by Spark*. A ML model is then trained and the transformations are bundled together with the classifier in a ML pipeline model, which creates the feature vectors and makes the predictions. Performance metrics and curves are computed from them.

In Figure 5.2 we can see how the data modeling relates with the data preparation. The ML pipeline and the model evaluation are explained in detail below.

### 5.4.1 ML pipeline

This process is summarized in Figure 5.5 and it comprises all the tasks that create the pipeline model used to predict the fraudulent purchase orders.

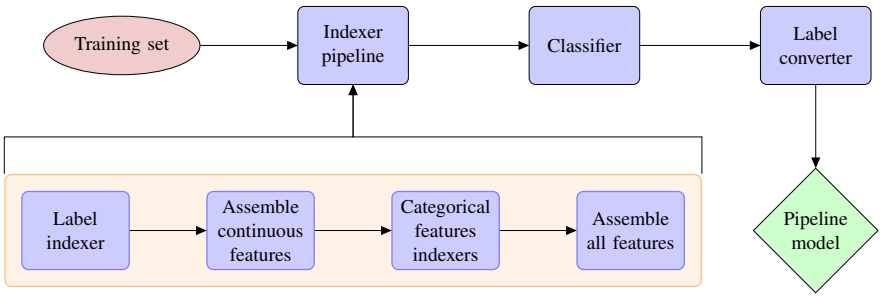



Figure 5.5: ML pipeline.

This pipeline has the following components:

- **Indexer pipeline:** first the labels are indexed, so they are doubles instead of strings. Then, the continuous features are assembled together into a vector. Afterwards the categorical features are also indexed and represented

* Spark vectors have integer-typed indices and double-typed values.

as doubles. Finally, the categorical and continuous features are assembled together into Spark feature vectors.

- **Classifier:** it receives as input the feature vectors and trains a classification model. In our case we have trained different RF models varying the hyperparameters. However, other models could also be trained reusing this same pipeline. For example, a weighted logistic regression was also trained to compare it with RF. In this case it was required to apply *min-max scaling*[*] to the continuous features and *one hot encoding*[†] to the categorical ones.

- **Label converter:** the labels are converted from doubles back to the original string labels.

### 5.4.2 Model evaluation

The predictions obtained from our training and test set can be used to get model performance metrics. All the metrics and curves explained in Section 2.3.3 are computed.

Metrics regarding the decisions taken by Qliro are also calculated, so our results can be compared with the former FDS.

This process is summarized in Figure 5.6.

Most of the experiments were performed at this stage, as they involved comparing the AU-PR obtained using different algorithms and techniques. In other cases the $F_2$ measure was also used, as sometimes the AU-PR didn't capture correctly the best models.

Different curves, such as the PR curve, and the recall, precision or $F_2$ by threshold curves have been also analyzed in some cases. Confusion matrices are also helpful for visualizing the model predictions.

Using AU-PR is usually preferred when no specific goal of precision and recall is set. If the fraud team has requirements to get a specific number of frauds per day, then using a ranking metric, such as the alert precision explained in Section 2.5.3, is preferred. Setting this threshold is not intuitive and in our thesis we have used the $F_2$ curve by threshold to select the most suitable one. This metric heuristically captures our preference for optimizing recall over precision.

The first models created had very bad predictive performance, as the data sampling techniques explained in the previous section hadn't been yet applied.

[*] Min-max scaling consists in rescaling the features into the range $[0,1]$. This is recommended in logistic regression to avoid some feature weights being updated faster than others. [†] One hot encoding transforms a categorical feature with $k$ categories into $k$ continuous binary features. Some ML algorithms like logistic regression can't handle categories, so this transformation is one possible solution.

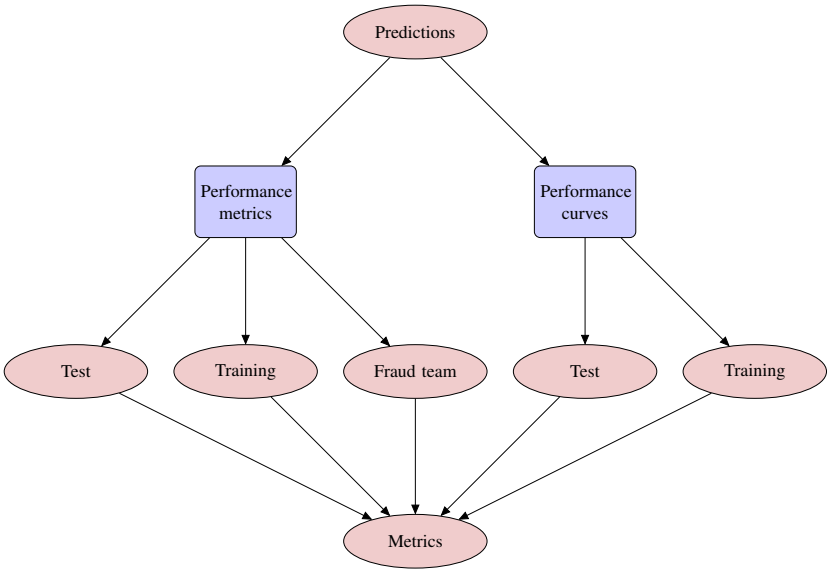



Figure 5.6: Model evaluation.

However, refining the data sampling and the feature selection allowed us to iteratively build better models.

The datasets used in the different experiments sometimes vary, but not within each experiment. Information about the training sets, such as its time interval or other descriptive statistics can't be presented due to data privacy concerns.

The different experiments performed are explained as follows. First, a motivation and description of the experiment is given. Then, the expected outcome and its assessment are detailed.

### Feature importance

- **Motivation:** the feature importance, explained in 2.3.3, is computed after training a model to understand the relevance of those features. It is useful to asses whether or not new engineered and selected features improve our model.

  The features that are less important can be removed in future modeling iterations. Analyzing the feature importance across the categorical and continuous variables allow us to know the importance of categories in our model. One of the reasons to use RF is that it supports categorical features, so they don't need to be one hot encoded, losing part of their predictive power. Hence, the importance of categories would reaffirm our choice of using RF. The importance of the new features engineered by us is also

evaluated.

- **Outcome:** bar plot with the feature importance of the 50 most important features. Categorical and continuous features are represented with different bar colors. The new features engineered are also specified. The result is presented in Section 6.2.
- **Interpretation:** all the feature importances sum up one. Usually some of the features contain most of the information. We have considered that features with less than 0.1% importance were irrelevant and iteratively removed when building new models. This way other features could be introduced in next models to test if they improved their predictive power.

**Comparison between training and test errors**

- **Motivation:** we want to analyze how well our model is learning from the training set and detect possible overfitting or underfitting.
- **Outcome:** the PR curve is used to asses the performance of the model in the training and test set. The training error is compared against the test set in cases with and without oversampling. Also, curves representing the recall by threshold are used to visualize the improvement of oversampling.
- **Interpretation:** to understand the PR curve we present the baseline for a random classifier. This experiment pretends to asses the effect of oversampling in the learning of our model.

**Comparison between weighted logistic regression and RF**

- **Motivation:** we want to compare the performance of a weighted logistic regression with a RF model. Due to the presence of categorical features and the big imbalance ratio our assumption is that RF outperforms weighted logistic regression.
- **Outcome:** the AU-PR of the models is compared using a bar plot. Also their confusion matrices are presented, so the results are easily interpreted. The confusion matrices have been normalized to mask any sensitive information. Instead of ratios we have considered that we have 10000 observations with 8 frauds.
- **Interpretation:** the best model is the one with a highest AU-PR. The confusion matrix can help us understand how the predictions would look like. However, these confusion matrices are for a specific probability threshold of the model, which is not necessarily the best one.

### Stratified CV

- **Motivation:** we want to perform stratified cross validation to maximize the AU-PR in order to tune the RF hyperparameters. Due to the big amount of different combinations, four smaller cross validation grids were performed to reduce the computational time. This way we could detect the best combination of RF hyperparameters for our training set. Due to the big amount of combinations we have ran the experiments using only two folds to speed up the training.

- **Outcome:** from each combination of parameters of the cross validation we obtain the average AU-PR for the different validation sets. The average values obtained are higher than the ones obtained in the test set because the training set was sampled using stratified sampling, so the data was already oversampled and undersampled. In the stratified cross validation we ensured that the frauds were only oversampled in the training set and that the ratio of frauds and non-frauds was maintained. However, as the imbalance ratio of the data used to construct validation test was smaller than in the original dataset, the results of AU-PR were better.

  The stratified cross validation wasn't present in Spark, so we had to implement it. It is part of the open-source code released in the project spark-imbalance presented in Appendix B.

  The best models obtained in each cross validation grid are compared regarding the AU-PR and their confusion matrices masked and normalized are also presented.

  The PR curves of these four best models are also presented, together with the recall, precision and $F_2$ by threshold. This curves are plotted to understand the differences between the models.

- **Interpretation:** the best combination of parameters is the one that yields the highest average AU-PR in the stratified cross validation.

### Ensembles

- **Motivation:** we want to test whether an ensemble of several RF models with different non frauds in the training set can improve the predictions. Each random forest model is oversampled and undersampled, but the non-frauds undersampled differ between the different models. The models' predictions are aggregated using bagging with two different voting strategies: hard-ensemble and soft-ensemble. Both strategies are compared. We have also varied the number of models combined in the ensemble.

- **Outcome:** the results are presented in a table with the precision, recall and $F_2$ values for each ensemble. The $F_2$ measure is then compared in a bar plot.

- **Interpretation:** the best model is the one that maximizes the $F_2$ measure. We didn't use here the AU-PR because in the case of hard ensembling we don't obtain probability thresholds.

#### Concept Drift

- **Motivation:** the CD changes are not abrupt as fraud behavior changes constantly, so passive approaches to solve it need to be explored. We want to detect if using the latest data helps in boosting the model performance. Fraud propagation could also help to reduce the undersampling, so it is also considered in the experiments.

- **Outcome:** bar plots regarding the AU-PR and the $F_2$ are presented in Section 6.8.

- **Interpretation:** usually the best model maximizes both, the $F_2$ and the AU-PR. However, in this experiment that is not true, so the reasons for that are analyzed. Here, the best model is the one maximizing area under the $F_2$ by threshold. This makes us think that the AU-PR is problematic in some cases because it gives the same importance to precision and recall.

#### Alert-feedback interaction

- **Motivation:** As fraud alerts and feedbacks from the investigators are likely to conform two different populations, we have divided the data into two different datasets: alerts from the fraud claims and feedbacks from the investigators.

  At Qliro, both fraud types are overlapping, as some found by the fraud team can be reported by the customers some time afterwards. The time frequency of these samples differs. The feedback from the investigators is received real-time, while the claims come with some delay. In both cases the non-frauds considered are the same because if not the dataset would be too balanced and it wouldn't perform properly when making the real predictions. Also, it would be dependent from the scoring rules.

  We want to test whether separating the fraud detection problem into two fraud detection problems helps in the model performance.

- **Outcome:** bar plot with the comparison of the AU-PR for models trained for predicting only alerts or feedbacks. They were compared against a model trained with both.
- **Interpretation:** the best model is the one that maximizes the AU-PR.

**Performance metric**

- **Motivation:** choosing the right metric to evaluate our models is hard, specially because each observation has a cost associated. In this thesis we have only considered the AU-PR and $F_2$ metrics to evaluate our models. In this experiment we explore an alternative cost-based metric.
- **Outcome:** an alternative metric having the costs into account.
- **Interpretation:** the limitations of the proposed metric are commented. A more in-depth analysis comparing cost-based metrics against traditional classification metrics is proposed as future work in Section 7.2.

## 5.5 Evaluation

Once we have achieved good model performance, it is still needed to review if we have achieved the business objectives (defined in Section 5.1) before deploying the model.

The catch rate, an important business metric haven't been presented here due to privacy issues, but we can measure the business suitability of our model using other metrics.

Two main experiments have been carried out: a comparison with the previous FDS and an analysis of new frauds detected that were being missed before.

**Comparison against former FDS**

- **Motivation:** the model performance has been compared against Qliro's scoring rules, without considering the investigators' posterior reviewing. It has also been compared against the investigators decisions to determine if the model could replace the fraud team.
- **Outcome:** bar plot measuring the $F_2$ measure.
- **Interpretation:** the model with a higher $F_2$ measure performs better. However, this metric is not used in the business, so a cost-based analysis is also needed when determining if a model is better in business terms than the current system.

**Business evaluation regarding our business goal**

- **Motivation:** we wanted to detect if frauds ignored by the previous FDS were caught by our model. Some of them could have been missed by the fraud team, so these ones would be the most valuable ones.

- **Outcome:** bar plot with the precision, recall and $F_2$ for different subsets of frauds with different business value from the business perspective.

- **Interpretation:** if the frauds missed by the previous FDS and the new ones are insignificant, then our model has failed to provide business value.

## 5.6 Deployment

Once, the model was validated regarding our business objectives, the deployment of it into the company's flow was required.

The purchase orders that come into the company need to be classified by our model and the frauds predicted need to be sent to the fraud team.

Qliro has a moderate volume of frauds, so a very frequent model update is not required. However, the non-stationarity has a big impact in the data distribution and the imbalance rate changes substantially across time, due to the fast growth rate of the company.

In Figure 5.7 the deployment process is summarized.

A Spark job was set to run every hour using Oozie because there was no business requirement for real time predictions. The purchase order data was obtained from production every 5 minutes using NiFi and inserted in a Hive table partitioned by hour and time.

Every hour the purchase orders from the previous hour are collected and inputted to the Spark ML pipeline model that computes the predictions. These predictions are exported using Sqoop into a SQL table previously created in the fraud team database, so they can access our model's predictions.

Oozie was used to schedule the job every hour, using as inputs the date and time of the previous hour. In case of failure the workflow would send us a mail with its ID, so we could check the reason looking at the logs.

The time spent making the predictions varies depending on the data volume in that hour. It usually takes between 1 and 5 minutes and most of this time is the overhead of initializing the Spark job.

Retraining the pipeline model is important because each day it passes its performance decreases due to CD. Another Oozie job was designed to retrain the model weekly.

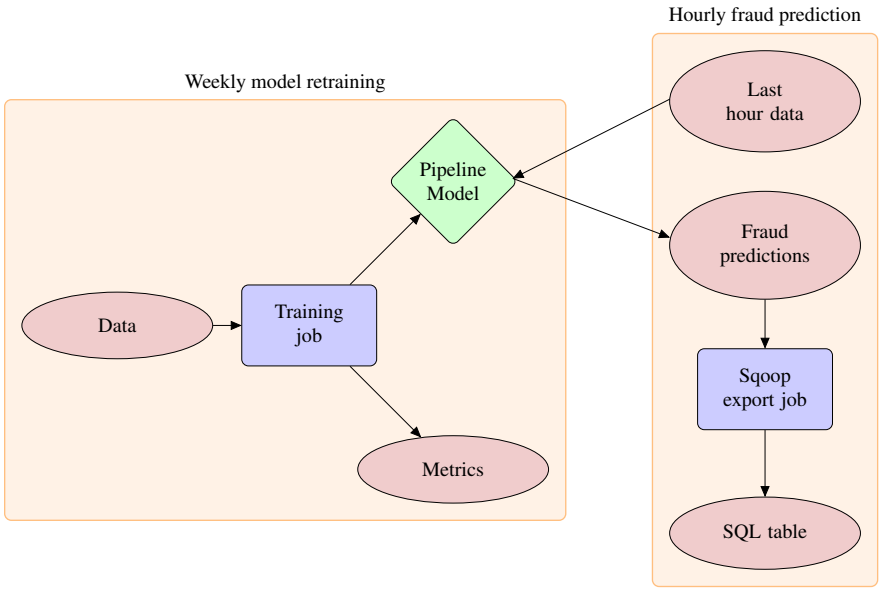



Figure 5.7: Deployment pipeline.

This retraining is almost the same as the one shown before in Figure 5.2. All those steps have been bundled together in a Spark job that outputs a pipeline model trained with the latest data and the metrics associated with this new model.

In this retraining the oversampling is fixed, but as the data size might vary, the imbalance rate changes according to the size of the training set. The unbalanced rate is automatically computed to use the biggest possible unbalanced rate, so we don't lose non-fraud information.

These metrics can be used to quantify how good the model is and also to provide the fraud team with information regarding the prediction and recall levels they can obtain with the different thresholds.

Experiments to assess the scalability of our solution were performed to see how the training time improved when increasing the memory and the parallelization.

Several models were trained varying the number of executors to compare their training time. The elapsed training time regarding the amount of memory assigned to the executors and the driver was also analyzed. The results for these experiments are presented in Section 6.12.

# 6

# Results

*"The story is not in the words; it's in the struggle."*

– Paul Auster, *The New York Trilogy*

In this chapter we present the outcome of the different experiments performed, which were justified in Chapter 5.

## 6.1 PCA

In Figure 6.1 some frauds and non frauds randomly sampled from the total population are plotted according with the values obtained after performing PCA. Prior to performing PCA the data was min-max scaled, so all values were comprised between zero and one. Some outliers that were distant from the rest of the population have been removed from the plot to visualize better the boundary between the frauds and non-frauds.

We observe that this boundary is not well defined. A big number of the fraudulent cases are arranged in a vertical line in the rightmost part of the plot. Those cases appear to be easier to detect, although there are also non frauds also along that line.

The rest of the frauds are scattered in the same area as the non frauds, so the class boundaries are clear in these two components. This tells us that our problem is hard, so getting a very good predictive model with this data seems unlikely.

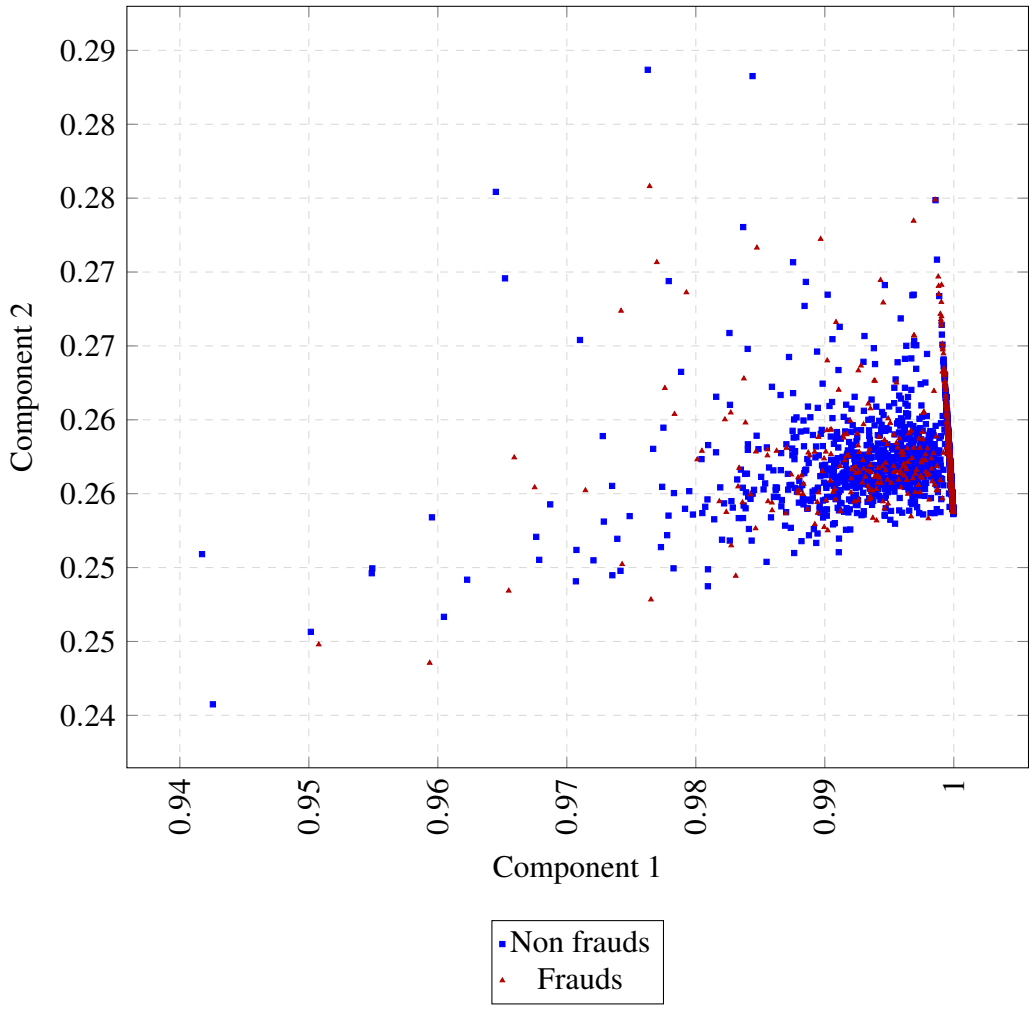



Figure 6.1: PCA for a subset of the data.

## 6.2 Feature importance

The feature importance presented here is computed from a RF model which uses the hyperparameters in Table 6.1.

Table 6.1: Hyperparameters of model used in the feature importance experiment.

| Hyperparameter | Value |
|---|---|
| Max depth | 28 |
| Number of trees | 200 |
| Impurity measure | entropy |
| Feature subset strategy | onethird |
| Subsampling rate | 0.3 |
| Oversampling rate | 80 |
| Unbalanced rate | 2 |

In Figure 6.2 we present a bar plot with the feature importance values for the 50 most important features of that model, as explained in 5.4.2.

The most relevant features are variables that hold aggregated customer information for their past behavior. Features related with the purchase amounts also have a high importance.

Below we summarize the main conclusions from analyzing the feature importance:

- Our model has 53 continuous features and 29 categorical ones.
- Out of the 82 features of this model, 65 have an importance bigger than 0.1%.
- The four most important features are continuous and hold 44.05% of the feature importance.
- There are 19 categorical features present in Figure 6.2 holding 22.58% of the feature importance.
- The number of features engineered by us was 11, being feature 5 the most relevant one and feature 39 the least relevant. They are all present in Figure 6.2 and can be identified by the letter E preceding their ID. They sum up 18.52% of the model importance.

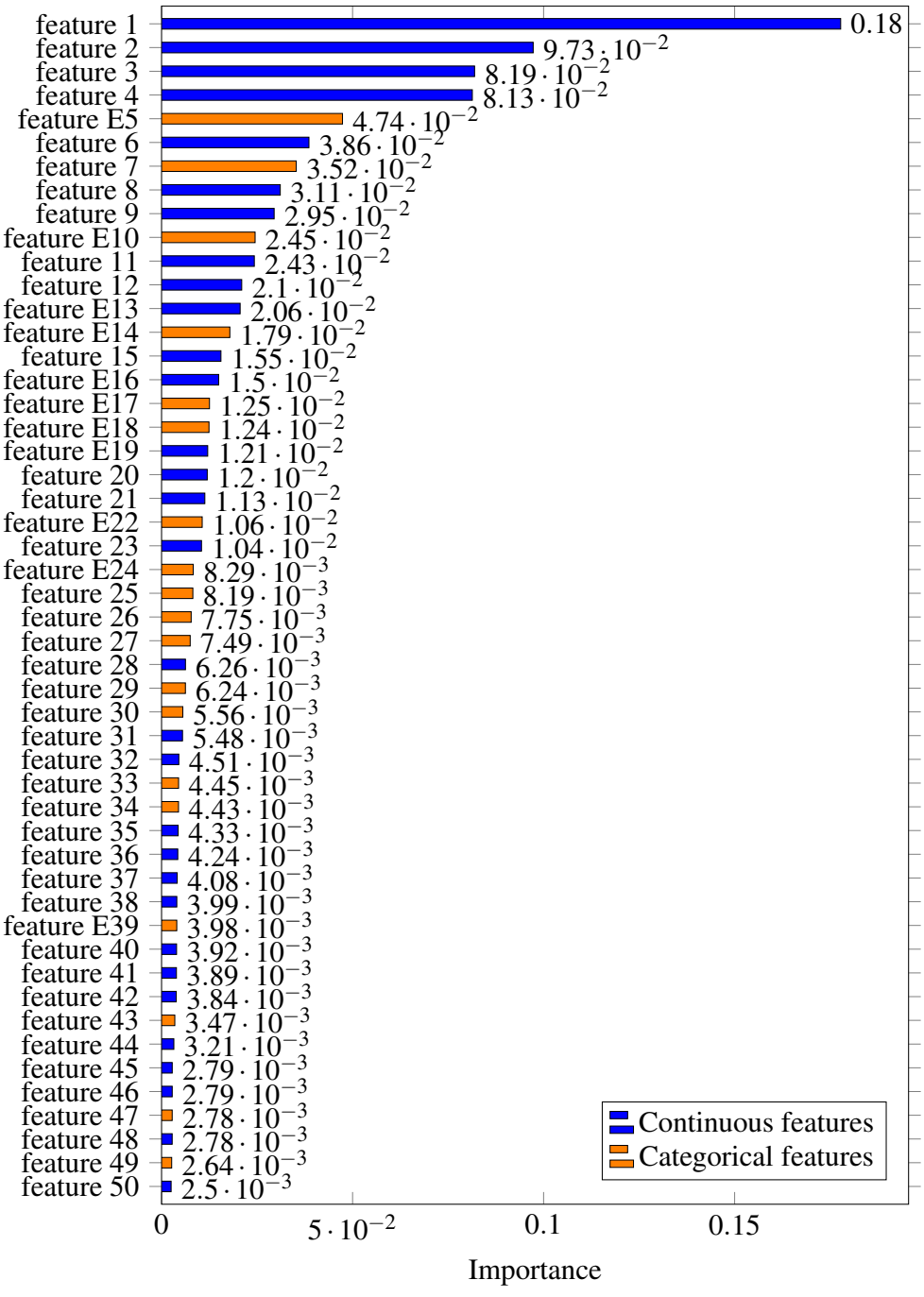



Figure 6.2: Feature importance for the 50 most relevant features.

We can conclude that the features engineered by us were very important, so they improved the model performance significantly. The categorical features were also significantly relevant, so choosing an algorithm that doesn't support categories is likely to perform worse.

## 6.3 Data sampling

Here we present the experiments related with the data sampling, analyzing the different oversampling techniques and ratios. The unbalanced rate effect in the performance of the models is also studied.

The hyperparameters used for the RFs used in these experiments are specified in 6.2. The oversampling rate and the unbalanced rate have varied in the different experiments.

Table 6.2: Hyperparameters of models used in the sampling experiments.

| Hyperparameter | Value |
|---|---|
| Max depth | 24 |
| Number of trees | 100 |
| Impurity measure | entropy |
| Feature subset strategy | onethird |
| Subsampling rate | 0.3 |

We can observe in Figure 6.3 that oversampling increases significantly the AU-PR. This is due to a big increase in the precision at the cost of having less recall. When oversampling the minority class 200 times, the increase of the precision is not worth it anymore in terms of the AU-PR.

We can also observe that using an unbalanced training set yields better results. This helps reducing the FPs, as more frauds are used to train the model. However, this precision increase is made at the cost of a decrease in the recall.

In Figure 6.4 we see that SMOTE-NC performs significantly worse than simply replicating the minority instances (the random oversampling model is the same as the unbalanced model used in Figure 6.3). One reason for this decrease is that the amount of categories in our dataset is very high, so the modification applied to SMOTE to support categories can create observations that contain noise in the continuous variables due to a lot of differences in the categorical values. Also, the categories in the synthetic samples might not capture relevant information, as they don't consider possible relationships among categories. This

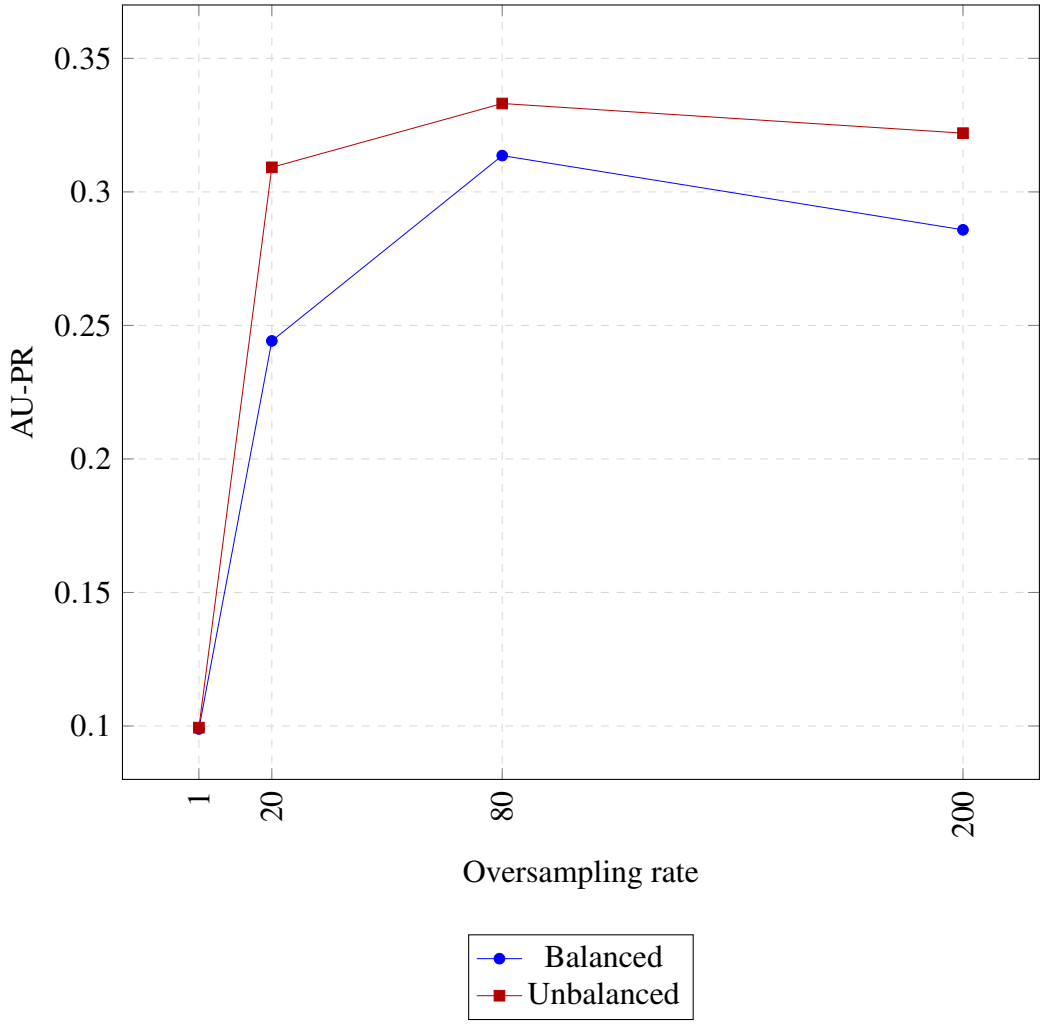



Figure 6.3: Comparison of models regarding the AU-PR varying the oversampling rate. The unbalanced rate is fixed: 1 in the balanced model and 2 in the unbalanced one.

can lead to losing information and the borderline between the frauds and non-frauds gets noisier.

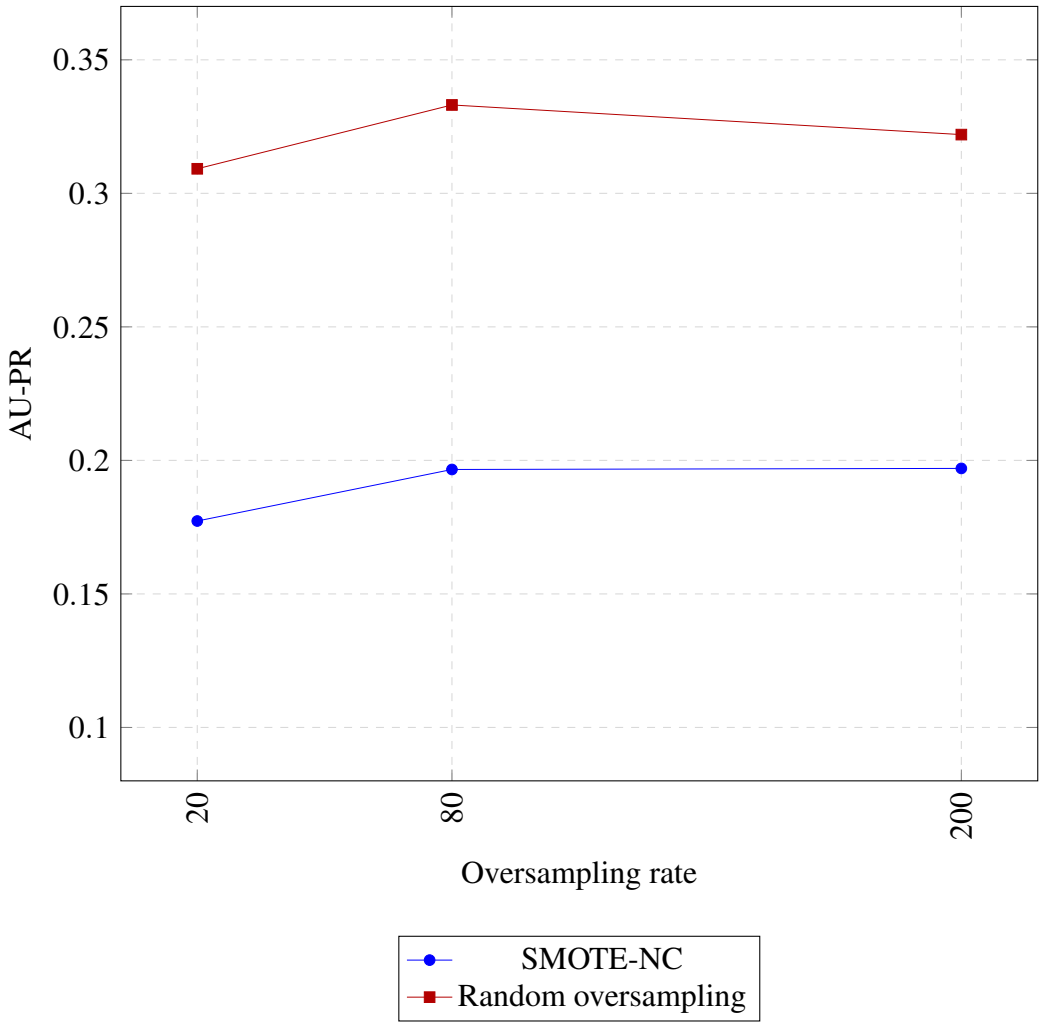



Figure 6.4: Comparison of models regarding the AU-PR varying the oversampling rate using different oversampling techniques. The unbalanced rate used for both models is 2.

In the Figure 6.5 we notice that the unbalanced rate is a good way to increase the AU-PR. In this figure no oversampling is used. If we compare it with Figure 6.6 we can observe that the performance is much better using oversampling.

When increasing the unbalanced rate, the recall decreases dramatically if we don't use oversampling. If our main goal is to have a very good precision at the cost of a very low recall, then not using oversampling could give us a precision

higher 0.5, which can't be obtained using oversampling in this case. However, in our problem we prioritize the recall over the precision, so the oversampling is preferred.

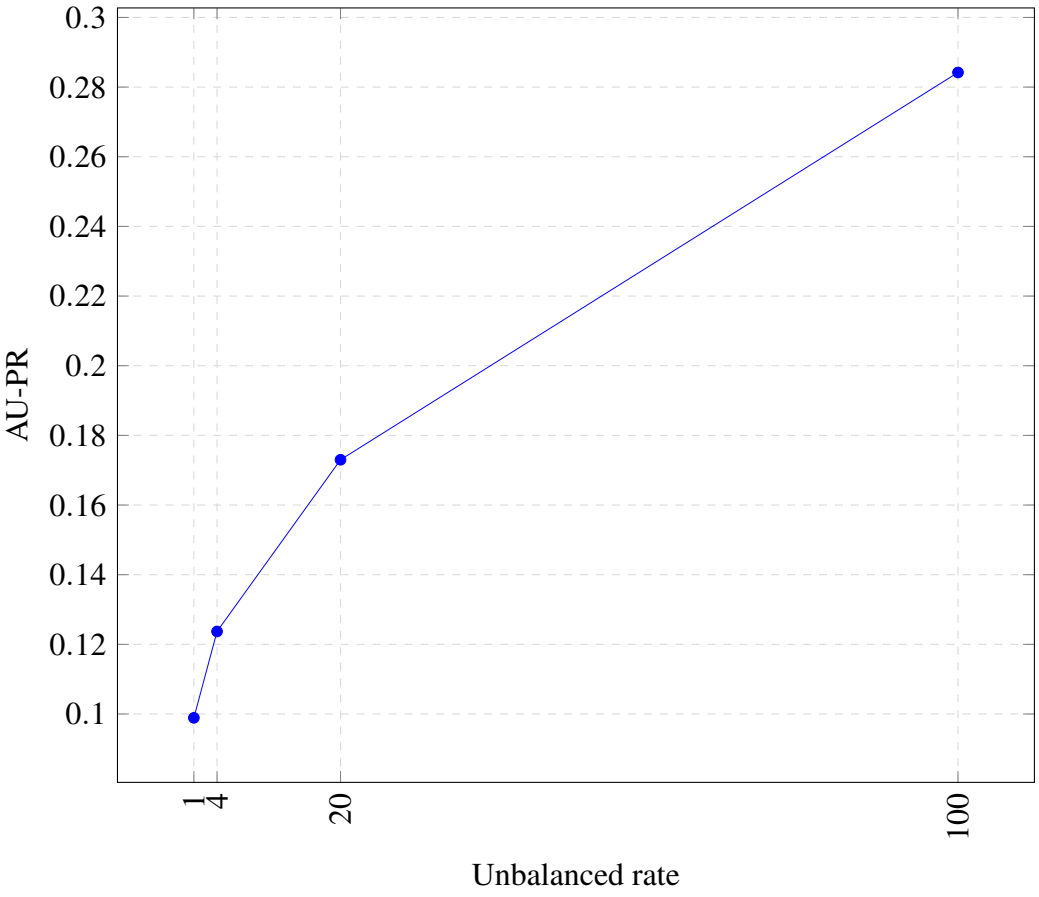



Figure 6.5: Comparison of models regarding the AU-PR varying the unbalanced rate without oversampling.

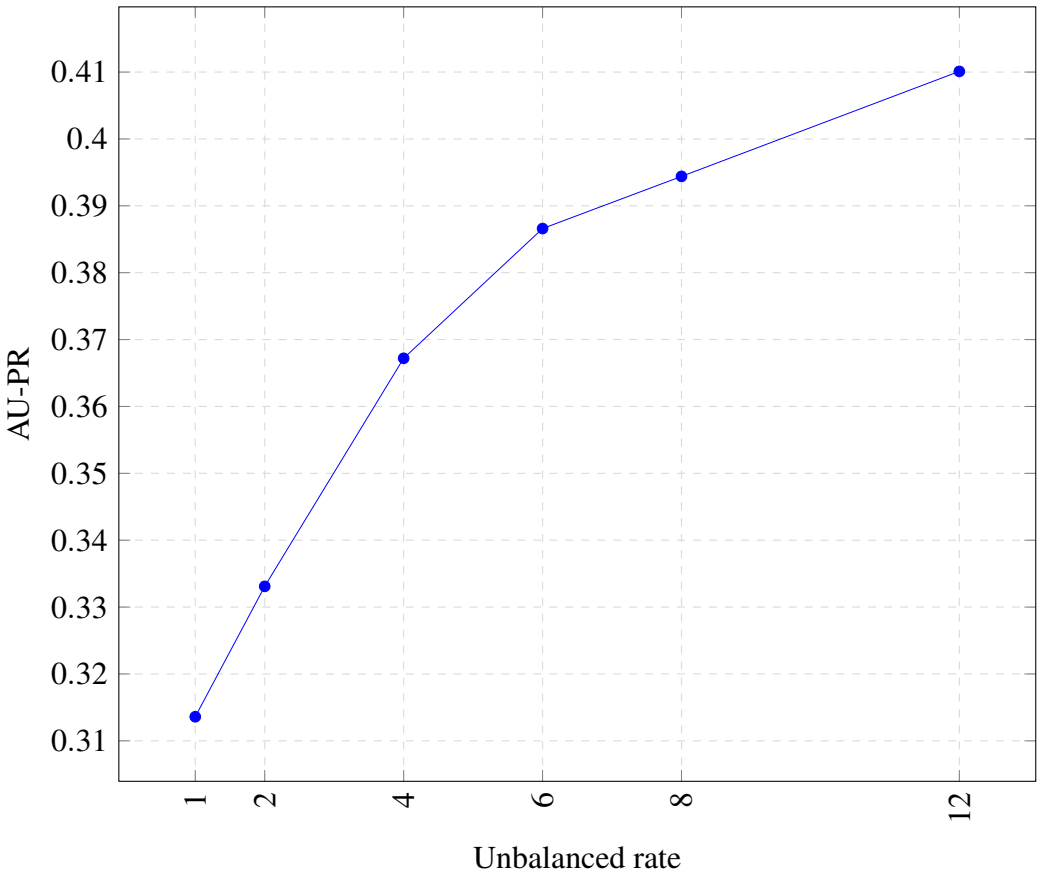


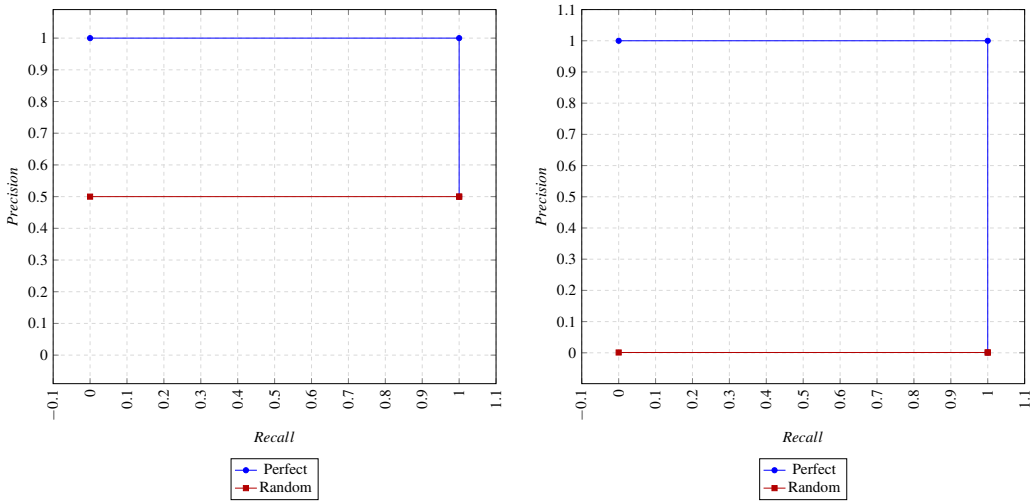

Figure 6.6: Comparison of model regarding the AU-PR varying the unbalanced rate, using an oversampling rate of 80.

## 6.4 Comparison between training and test errors

In Figure 6.7 we present the baselines for the AU-PR in the case of a balanced datasets and in our unbalanced specific example.

We observe that due to the high class imbalance, the AU-PR is a horizontal line with a precision equal to our imbalance ratio, which in this case is very close to zero. In the balanced case this value is 0.5, so the random model performs much better. Our aim is to obtain a curve that is as close to the perfect model as possible. If our model presents a very high precision with zero recall, this means that it has a good early retrieval, but if it is very low it might mean that we are not learning enough from our training data.

The precision, recall and $F_2$ associated to the random model have values very close to zero because it doesn't predict anything. Conversely, the perfect model has a value of 1 in all those metrics because all the frauds are captured with no FPs.




(a) PR curve baselines in a balanced dataset.

(b) PR curve baselines for our unbalanced dataset.

Figure 6.7: PR curve baselines.

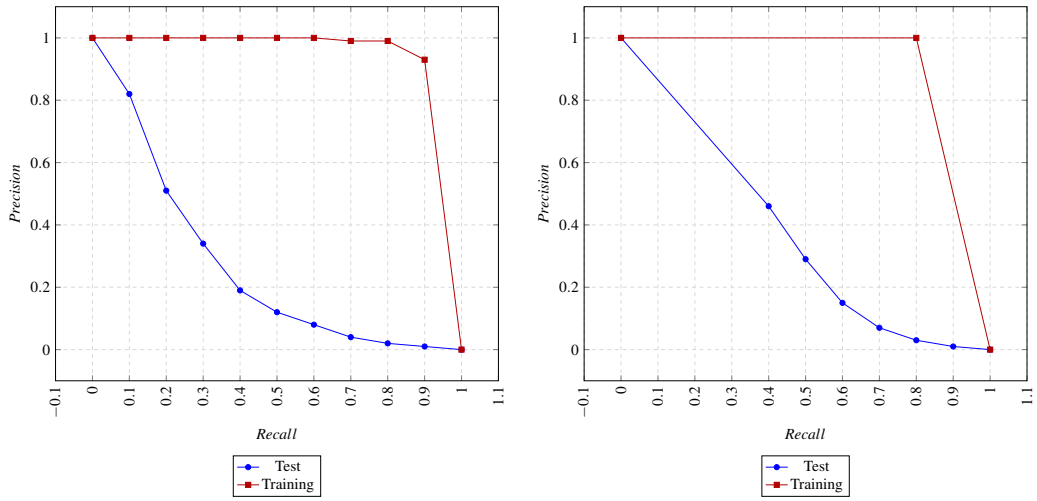
The hyperparameters used for the two models compared in the experiments are presented in Table 6.3. These models were selected from the ones with the best performance in Section 6.3.

In Figure 6.8 we observe that the difference of area between the training and test set is very similar in both cases. The test set predictions using oversampling present better generalization.

Table 6.3: Hyperparameters of models used in the comparison between training and test errors.

| Hyperparameter | No oversampling | Oversampling |
|---|---|---|
| Max depth | 24 | 24 |
| Number of trees | 100 | 100 |
| Impurity measure | entropy | entropy |
| Feature subset strategy | onethird | onethird |
| Subsampling rate | 0.3 | 0.3 |
| Oversampling rate | 0 | 80 |
| Unbalanced rate | 100 | 12 |

We can see that in both cases there is overfitting in the learning phase. However, the PR curve is misleading in this case because in Figure 6.8a the predictions have a very high precision for all the thresholds due to the undersampling, so we can't appreciate what the model is learning from the frauds from that curve. To observe that, we can use the recall by threshold curve, which will be a better indicator of how well our model captures the fraud distinctive features.




(a) PR curves with undersampling.

(b) PR curves with oversampling.

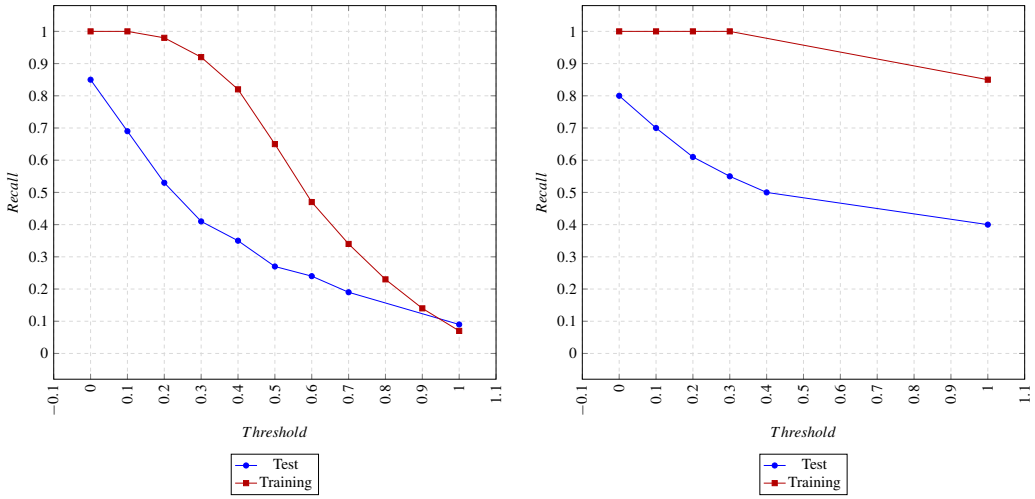
Figure 6.8: Training and test PR curves comparison using undersampling and oversampling.

In Figure 6.9 the recall by threshold shows that the recall of the model decreases very fast in the case of undersampling. However, when oversampling we are overfitting, but this helps boosting the performance of the test predictions.

We can observe that the distance between both curves is higher in the oversampling case, so we are overfitting more. However, this overfitting is preferred to the underfitting observed in the undersampled model.

The test curve in Figure 6.8b is above the test curve in Figure 6.8a except for very small threshold values, which are not very useful because of the low precision associated with them.

The problem with undersampling without oversampling is that either we keep most non-frauds, creating a lot of FPs, or we remove most non-frauds at the cost of learning less from the frauds and increasing the FNs. Replicating the frauds allows us to reduce the undersampling needed, reducing that precision loss.




(a) Recall by threshold curve with undersampling (b) Recall by threshold curve with oversampling

Figure 6.9: Recall by threshold curves comparison between undersampling and oversampling.

## 6.5 Comparison between weighted logistic regression and RF

In this section we have compared a RF with with two different weighted logistic regression models.

The following models have been considered:

- $\boldsymbol{LR_1}$: Logistic regression using oversampling 80 with no undersampling (no unbalanced rate was specified, so the imbalance ratio was big even after oversampling).

- $\boldsymbol{LR_2}$: Logistic regression using oversampling rate 80 and unbalanced rate 2.

- ***RF***: Random forest with hyperparameters specified in Table 6.4.

Table 6.4: Hyperparameters of RF model used for comparison with weighted logistic regression models.

| **Hyperparameter** | **Value** |
|---|---|
| Max depth | 15 |
| Number of trees | 100 |
| Impurity measure | gini |
| Feature subset strategy | log2 |
| Subsampling rate | 0.4 |
| Oversampling rate | 80 |
| Unbalanced rate | 2 |

We observe in Figure 6.10 the AU-PR for the models with threshold 0.5 (this might not be the best threshold, but it serves for the comparison purposes). The RF clearly outperforms the other models.

To get a better idea about the predictions made by our models, their masked confusion matrices are presented in Figure 6.11.

We can see that both the precision and the recall are much higher in the case of RF.

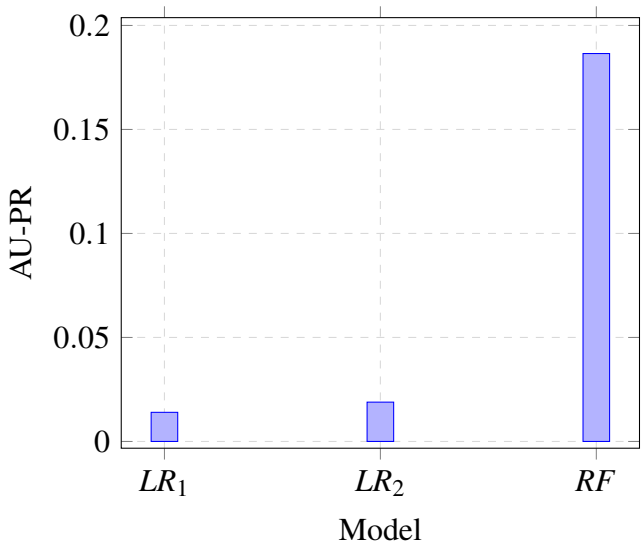



Figure 6.10: Comparison between weighted logistic regression and random forest.

| | **Predicted non fraud** | **Predicted fraud** |
|---|---|---|
| **Non frauds** | 7299 | 2693 |
| **Frauds** | 1 | 7 |

(a) Confusion matrix for $LR_1$

| | **Predicted non fraud** | **Predicted fraud** |
|---|---|---|
| **Non frauds** | 9668 | 324 |
| **Frauds** | 4 | 4 |

(b) Confusion matrix for $LR_2$

| | **Predicted non fraud** | **Predicted fraud** |
|---|---|---|
| **Non frauds** | 9826 | 166 |
| **Frauds** | 2 | 6 |

(c) Confusion matrix for $RF$

Figure 6.11: Confusion matrices comparison between $LR_1$, $LR_2$ and $RF$ using 0.5 as a probability threshold.

## 6.6 Stratified Cross-validation

Different cross validation parameter grids will be run to select the best parameters for the model previously created. The metric used for choosing the best results in the validation sets is the AU-PR.

The following parameter grids have being used:

- $\boldsymbol{CV_1}$: grid varying the feature subset strategy, the maximum depth of the trees and the subsampling rate.
- $\boldsymbol{CV_2}$: grid varying the feature subset strategy and the impurity measure.
- $\boldsymbol{CV_3}$: grid varying the impurity measure, the subsampling rate and the maximum depth of the trees.
- $\boldsymbol{CV_4}$: grid varying the number of trees and their maximum depth.

In Table 6.5 the different fixed parameters for each grid are listed.

Table 6.5: Hyperparameters of models used in the cross validation experiments.

| **Hyperparameter** | $\boldsymbol{CV_1}$ | $\boldsymbol{CV_2}$ | $\boldsymbol{CV_3}$ | $\boldsymbol{CV_4}$ |
|---|---|---|---|---|
| Max depth | - | 16 | - | - |
| Number of trees | 100 | 100 | 100 | - |
| Impurity measure | gini | - | entropy | entropy |
| Feature subset strategy | - | - | onethird | onethird |
| Subsampling rate | - | 0.4 | - | 0.3 |
| Oversampling rate | 80 | 80 | 80 | 80 |
| Unbalanced rate | 2 | 2 | 2 | 2 |

We can see in Table 6.6 that using *log2* as feature subset strategy yields slightly better results for all the parameters in the grid. For both feature subset strategies, an increase in the depth of the trees is desirable. However, the subsampling rate behaves differently depending on the depth of the trees. When the trees are of depth 8 having a higher subsampling is desirable, but as the number of trees increases keeping more data out of the tree with a smaller subsampling rate helps to achieve better results. The increase in accuracy in this case is not very significant, but it could be for deeper trees. This result is coherent with the behavior of the trees, as when we increase the depth it is necessary to keep more data out of each tree to increase their generalization.

Table 6.6: Cross validation results for $CV_1$ regarding AU-PR.

| Feature subset strategy | Max depth | Subsampling rate | Average metric |
|---|---|---|---|
| *sqrt* | 8 | 0.4 | 0.4065 |
| *sqrt* | 8 | 0.6 | 0.4104 |
| *sqrt* | 14 | 0.4 | 0.6429 |
| *sqrt* | 14 | 0.6 | 0.6401 |
| *log2* | 8 | 0.4 | 0.4142 |
| *log2* | 8 | 0.6 | 0.4145 |
| ***log2*** | **14** | **0.4** | **0.6437** |
| *log2* | 14 | 0.6 | 0.6414 |

We observe in Table 6.7 that the best feature subset strategy depends on the selected impurity measure. For the *gini* the best one is *log2*, while for the *entropy* the one that achieves the best results is *onethird*. The number of features is 82 (the same as in Section 6.2), so *log2* represents 6 features, *sqrt* 9 features and *onethird* 27 features. The *entropy* seems to outperform the *gini*, but this might vary depending on other hyperparameters.

Table 6.7: Cross validation results for $CV_2$ regarding AU-PR.

| Feature subset strategy | Impurity measure | Average metric |
|---|---|---|
| *4* | *gini* | 0.6547 |
| *4* | *entropy* | 0.6582 |
| *log2* | *gini* | 0.6825 |
| *log2* | *entropy* | 0.6921 |
| *sqrt* | *gini* | 0.6769 |
| *sqrt* | *entropy* | 0.7035 |
| *onethird* | *gini* | 0.6468 |
| ***onethird*** | ***entropy*** | **0.7063** |
| *40* | *gini* | 0.6266 |
| *40* | *entropy* | 0.7000 |

In Table 6.8 we have confirmed that the best impurity measure is the *entropy*. Creating deeper trees seems to increase the results, although this is caused by an

increase in the precision and a reduced recall. Small changes in the subsampling rate seem to have little effect in the performance, but 0.3 is the best one in most of the cases.

Table 6.8: Cross validation results for $CV_3$ regarding AU-PR.

| Impurity measure | Subsampling rate | Max depth | Average metric |
|---|---|---|---|
| *gini* | 0.3 | 16 | 0.611 775 |
| *gini* | 0.3 | 20 | 0.649 171 |
| *gini* | 0.3 | 24 | 0.664 738 |
| *gini* | 0.3 | 28 | 0.677 359 |
| *gini* | 0.4 | 16 | 0.613 574 |
| *gini* | 0.4 | 20 | 0.648 387 |
| *gini* | 0.4 | 24 | 0.662 262 |
| *gini* | 0.4 | 28 | 0.674 608 |
| *entropy* | 0.3 | 16 | 0.667 289 |
| *entropy* | 0.3 | 20 | 0.698 023 |
| *entropy* | 0.3 | 24 | 0.714 221 |
| ***entropy*** | **0.3** | **28** | **0.720390** |
| *entropy* | 0.4 | 16 | 0.666 415 |
| *entropy* | 0.4 | 20 | 0.695 244 |
| *entropy* | 0.4 | 24 | 0.711 937 |
| *entropy* | 0.4 | 28 | 0.720 114 |

In Table 6.9 we appreciate that adding more trees has little improvement in our model and it increases significantly the training time. Increasing the depth of trees seems to be far more efficient to increase the performance of our models.

Adding more trees doesn't improve the performance of the models, which in some cases decreases. As this happens for all the depths it doesn't seem caused by the random nature of the bootstrap. A possible answer could be that the condition of highly independence of trees required in the RF algorithm is not satisfied due to the combination of a big minority class oversampling and large number of trees.

The AU-PR for the best model obtained in each of the cross validation grids are compared in Figure 6.12. The best one is $CV_3$ which is also the deepest one.

In Figure 6.13 we compare different masked confusion matrices for the threshold 0.5. We observe that it is not clear for this threshold whether $CV_3$ is

Table 6.9: Cross validation results for $CV_4$ regarding AU-PR.

| Max depth | Number of trees | Average metric |
|---|---|---|
| 16 | 100 | 0.683 254 97 |
| 16 | 150 | 0.681 571 50 |
| 16 | 200 | 0.683 074 05 |
| 20 | 100 | 0.710 574 26 |
| 20 | 150 | 0.709 818 64 |
| 20 | 200 | 0.710 134 14 |
| 24 | 100 | 0.722 620 45 |
| 24 | 150 | 0.721 586 44 |
| **24** | **200** | **0.72272906** |

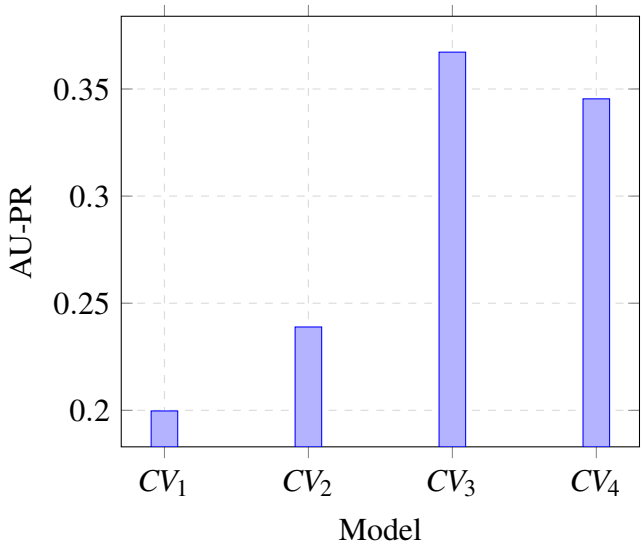



Figure 6.12: Comparison regarding the AU-PR between best model for each cross validation experiment.

superior to $CV_4$. From this matrices it seems that $CV_4$ is preferred because more frauds are predicted at the cost of very few extra FPs.

| | **Predicted non fraud** | **Predicted fraud** |
|---|---|---|
| **Non frauds** | 9788 | 202 |
| **Frauds** | 3 | 7 |

(a) Confusion matrix for $CV_1$

| | **Predicted non fraud** | **Predicted fraud** |
|---|---|---|
| **Non frauds** | 9891 | 99 |
| **Frauds** | 3 | 7 |

(b) Confusion matrix for $CV_2$

| | **Predicted non fraud** | **Predicted fraud** |
|---|---|---|
| **Non frauds** | 9969 | 21 |
| **Frauds** | 5 | 5 |

(c) Confusion matrix for $CV_3$

| | **Predicted non fraud** | **Predicted fraud** |
|---|---|---|
| **Non frauds** | 9960 | 29 |
| **Frauds** | 4 | 6 |

(d) Confusion matrix for $CV_4$

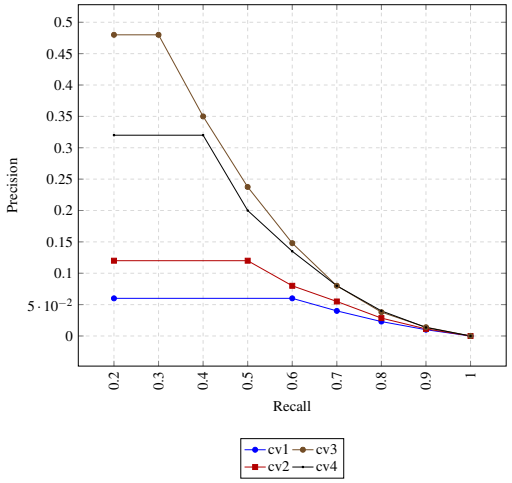
Figure 6.13: Comparison of confusion matrices for the best models in the cross validation experiments using a probability threshold of 0.5.

Let's look at the PR curve and $F_2$ measure by threshold for these models to understand better their behavior.

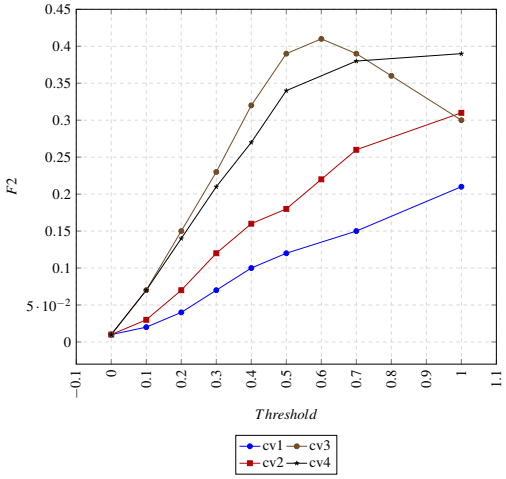
In Figure 6.14a all the curves can achieve a recall of 1 if everything is considered as fraud (threshold 0). However, this has no use. Each curve only has points in the parts where the recall level specified can be achieved and then for lower levels of recall they follow an horizontal line because no further improvement is possible.

The PR curve of $CV_3$ completely dominates the others which indicates that this is indeed the best model. Even in terms of the $F_2$ measure $CV_3$ is better, althought the best possible threshold is 0.6 instead of 0.5. This means that the increase of precision achieved is worthed it according to this metric, even at the cost of slightly reducing the recall.

In Figure 6.15 we observe that better models in terms of AU-PR have lower recall levels at the cost of higher precision, so they are able to be more precise selecting high threshold values. This gives the investigators more flexibility to select the number of suspicious cases they want to investigate. In Figure 6.15a we see that the models have a limit in the precision they can obtain, which is much higher in the case of $CV_3$. In Figure 6.15b we also see that there is a lower bound in the recall that can be achieved by the models, which corresponds to the predictions with probability threshold 1.

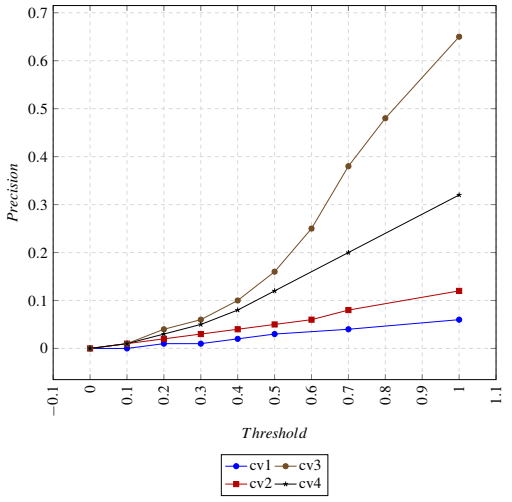



(a) AU-PR measure by threshold for the best cross validation models.

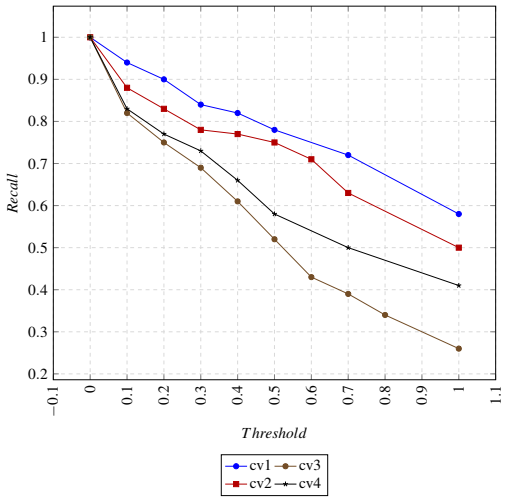



(b) $F_2$ measure by threshold for the best cross validation models.

Figure 6.14: AU-PR and $F_2$ by threshold comparison for the best models in the cross validation experiments.




(a) Precision by threshold for the best cross validation models.




(b) Recall by threshold for the best cross validation models.

Figure 6.15: Precision and recall by threshold comparison for the best cross validation models.

## 6.7 Ensembles

In this experiment we have used the hyperparameters in Table 6.10 to create different models to ensemble using two different voting strategies: hard-ensembling and soft-ensembling.

The ensemble models are noted as $E_k$, where $k$ is the number of models used in the ensemble. To differentiate from hard and soft ensemble we have added the subscript $s$ for soft-ensembling and $h$ for hard-ensembling.

Table 6.10: Hyperparameters of RF models used for creating the ensemble, concept drift and alert-feedback interaction experiments.

| **Hyperparameter** | **Value** |
|---|---|
| Max depth | 24 |
| Number of trees | 100 |
| Impurity measure | entropy |
| Feature subset strategy | onethird |
| Subsampling rate | 0.3 |
| Oversampling rate | 80 |
| Unbalanced rate | 2 |

In this experiments all the models have used the threshold 0.5 and have been compared using the $F_2$ for that threshold. We couldn't use the AU-PR to consider all possible thresholds because the outcome of the hard-ensembling doesn't have a probability associated.

In Table 6.11 we observe that the recall doesn't improve when performing ensembling, while the precision shows a slight increase. The best ensemble regarding the $F_2$ is $E_{h20}$, which uses hard-ensembling with 20 models. However, $E_{h10}$ is preferred, as the $F_2$ measure is very similar and it requires half of the models, reducing the computational cost of the ensemble. Nevertheless, this ensemble was not used in the model deployed either because the performance gain wasn't enough to justify the big increase in the training time, along with its higher footprint.

In Figure 6.16 the $F_2$ values are compared. We see that in most of the cases hard-ensembling outperforms soft-ensembling. This might be due to having very non-stable models due to the noise present in our dataset. In those scenarios hard-ensembling is usually recommended, while soft ensembling usually works better for good calibrated models, as it uses the probabilities to combine the models.

Table 6.11: Ensemble comparison for different datasets which differ in the non frauds observations undersampled.

| | ***Precision*** | ***Recall*** | ***F2*** |
|---|---|---|---|
| No ensemble | 0.1533 | 0.5665 | 0.3681 |
| $E_{h5}$ | 0.1602 | 0.5595 | 0.3733 |
| $E_{s5}$ | 0.1600 | 0.5653 | 0.3752 |
| $E_{h10}$ | 0.1675 | 0.5561 | 0.3799 |
| $E_{s10}$ | 0.1619 | 0.5642 | 0.3769 |
| $E_{h15}$ | 0.1648 | 0.5618 | 0.3792 |
| $E_{s15}$ | 0.1647 | 0.5630 | 0.3795 |
| $E_{h20}$ | 0.1668 | 0.5584 | 0.3800 |
| $E_{s20}$ | 0.1635 | 0.5618 | 0.3777 |
| $E_{h35}$ | 0.1637 | 0.5607 | 0.3776 |
| $E_{s35}$ | 0.1633 | 0.5584 | 0.3763 |

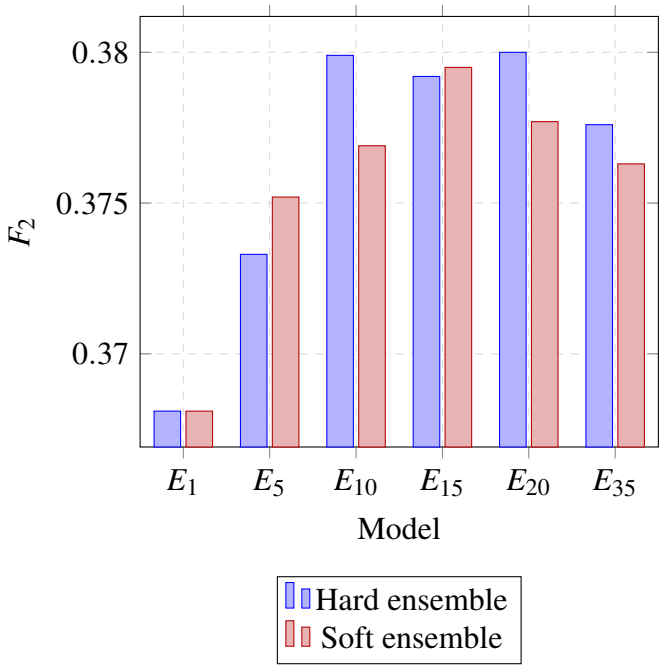



Figure 6.16: Comparison between ensemble models.

## 6.8 Concept Drift

In this experiment we have created models trained with datasets that simulated the CD problem using the hyperparameters listed in Table 6.10.

The datasets used to simulate the CD are:

- **Test set:** subset of the data from a time interval $T$.
- **Training set:** different training sets using consecutive and non-overlapping intervals prior to $T$ and with the same length as $T$. We refer to them as $T_k$, where $k$ is the number of intervals of time used to construct the dataset. A training set $T'$ was also created, which covers the same time interval as $T$, but in the previous year. Finally, frauds propagation was also applied in those datasets, and it is noted with the subscript $p$: $T_{kp}$ or $T'_p$. The frauds propagation performed adds to the training set all the frauds from the previous year that are in that same time interval*.

  For example, if the interval $T$ chosen is November and December from the current year, then $T_2$ would include the interval of time comprehended between July and October of that same year. In that example, $T'$ would have November and December data from the previous year. If we applied fraud propagation to that example we would have that $T_{2p}$ would also have the frauds between July and October from the previous year and $T'_p$ would also have the frauds from November and December from two years back.

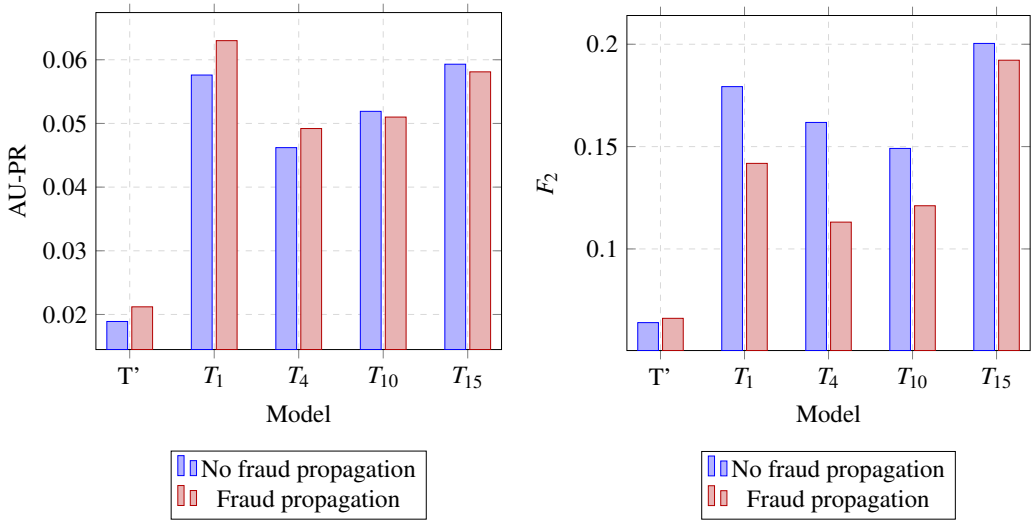
In Figure 6.17 the different models are compared regarding the AU-PR and the $F_2$. We see discrepancies in the best models. According to the AU-PR the best model is $T_{1p}$, but the highest $F_2$ measure is obtained by $T_{15}$. Frauds propagation worsens the results in terms of $F_2$ except in $T'$, and slightly improves the results in terms of the AU-PR when using a small amount of intervals in the training set.

Both metrics report bad results for $T'$. This means that trends from previous years are not important and that our data suffers a lot from CD.

$T_{15}$ is the only model that captures a complete year and a lot of features are time related, so this explains its good performance. Conversely, $T_1$ has very good results because it contains a lot of information about the latest trends among fraudsters.

We still need to decide which model is better $T_{1p}$ or $T_{15}$. In Figure 6.18 the PR curve and the $F_2$ by threshold are plotted. We clearly see that $T_{15}$ is better in

* If we added all the frauds from the previous year to the training set, the model would learn that everything outside the training time interval chosen is fraud. In the case of a training set that doesn't contain the period in the test set this will mean that most of the test predictions will be fraud. This is because of the time-based features used in our models.




(a) Comparison of CD the models regarding the AU-PR.

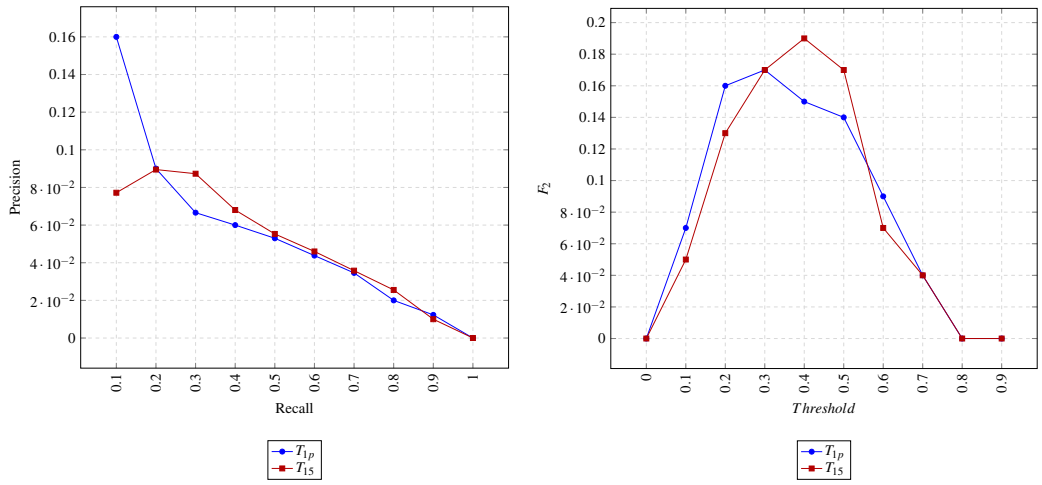
(b) Comparison of CD the models with threshold 0.5 regarding the $F_2$.

Figure 6.17: Comparison between CD models.

terms of $F_2$. However, $T_{15}$ also dominates regarding the PR curve for recall levels higher than 0.3. Hence $T_{15}$ is preferred. The bad performance in the AU-PR was caused by a decrease in precision for small recalls. This is caused by a lot of FPs with high probability thresholds.

Summarizing, in this experiment we have extracted the following conclusions regarding the CD:

- The CD effect is very important in our data. We can observe that the performance results from this experiment are lower than in previous ones because of that. This implies that retraining the model frequently is critical.
- Our models depend on time-based features, so using a dataset with data from the whole year is preferable.
- Propagating frauds adds noise to our data, so the predictions retrieved are worse.
- The latest data contains more information about the frauds than older data, as observed by the good results obtained in $T_1$.




(a) PR curve for the two best concept drift models.

(b) $F_2$ measure by threshold for for the two best concept drift models.

Figure 6.18: PR curve and $F_2$ by threshold comparison for $T_{1p}$ and $T_{15}$.

## 6.9 Alert-feedback interaction

In this section we perform experiments related to the alert-feedback interaction using three different datasets:

- $M$: baseline model trained from both the alerts and the feedbacks.
- $M_a$: alert model trained from the fraud claims.
- $M_f$: feedback model trained from the investigators' feedbacks.

The hyperparameters used in the RF models of this section are listed in Table 6.10.

We can observe in Figure 6.19 that training separate models using the alerts and the feedbacks decreases the performance. The performance in the feedbacks in much lower, which makes sense, as those frauds are more likely to be mislabeled. However, when combining both, the AU-PR increases, meaning that the feedbacks contain useful information.

In our experiment we have used the same non-fraud population for both cases. If we had used in $M_f$ only the non-frauds caught by the rules in their previous system maybe the results could have improved. However, this approach would introduce bias towards the previous system.

Creating other data subgroups using clustering or fraud rules could help us creating several models with higher performance. However, this is out of the scope of this thesis, so it is proposed as future work in Section 7.2.

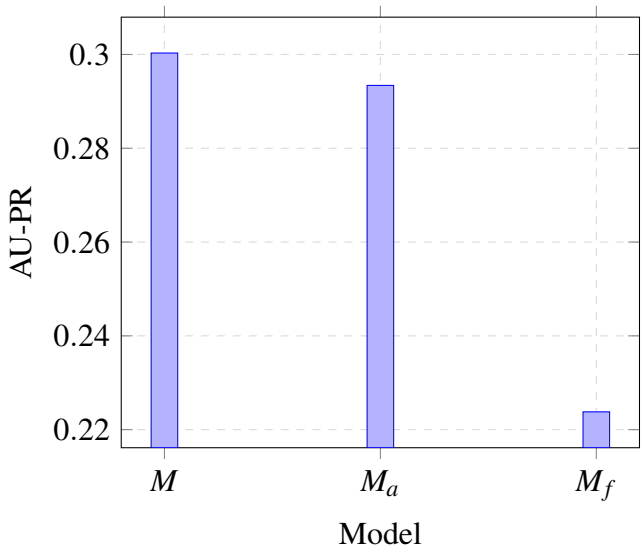



Figure 6.19: Comparison between alerts and feedback models.

## 6.10 Performance metric

In this section we compare the advantages and disadvantages of the different performance metrics used and explore other possible alternatives.

Optimizing the AU-PR is not always desirable due to big decreases in the recall, so using the $F_2$ is sometimes preferred in those cases. However, this metric is not evaluated across all the thresholds. However, neither of them consider costs, something required to be validated by the business.

The AU-PR gives the same importance to precision and recall, so it can be misleading if the precision is very high and the recall very low, as the AU-PR would have a decent result. A very high precision with very low recall is considered much better than other cases with better recall and a decent precision. This is not aligned with our goals. Giving a higher importance of recall over precision solves this problem, so the $F_2$ measure is suitable for those problematic cases.

The precision at $k$ described in Section 2.5.3 is useful when investigators want to obtain a fixed amount of frauds per day, but at Qliro this was not the case. Hence, when there is uncertainty about $k$, that metric is not so valuable because it focuses on optimizing the model only for that value of $k$. Optimizing over all the possible $k$ values gives more flexibility to the fraud team, something needed, as the number of frauds investigated in a real environment usually vary depending on the day.

Hence, the AP is a better way to measure the performance of our fraud detection models. The AP is one way to interpolate the AU-PR, which is the metric used in this work. Our result of AU-PR might differ slightly from the AP because Spark uses another implementation to compute it*.

If we want to capture the losses due to the investigation costs, we can create a metric based on the $F_2$ measure that captures the losses of the FPs and FNs, giving a higher importance to reducing misclassification costs in the FNs. A recall based on costs is the catch rate, so we could try to also create a precision based on costs and use a $F_2$ measure on those metrics. A higher importance to the cost recall would make sense because even if we save the same amount of money, reducing the FNs is always preferable, as non-caught frauds have a chain effect and can cause frauds increasing in the future. The TPs and FPs are reviewed by the investigators, but they sometimes make mistakes and misclassify some cases. This misclassification probability can be computed from historical data and used when computing the costs.

We propose a gain and cost matrix with positive costs in the cases when the

* Spark uses the trapezoidal rule [101] to interpolate the area under the curve from a sequence of points, so depending on the points chosen to construct the curve, which are not necessary the same as $k$, we will obtain slightly different results.

company receives money. If $p$ is the probability of the investigators classifying the instances correctly, $C_i$ is the investigation cost of one case, $t$ is the cost of a transaction, and $T_i$ are the set of transactions' costs associated with the element $i$ of the confusion matrix, we can compute the matrix presented in Figure 6.20.

| | | **Predicted value** | |
|---|---|---|---|
| | | NOT FRAUD | FRAUD |
| **Actual value** | NOT FRAUD | $\sum_{t \in T_{TN}} t$ | $\sum_{t \in T_{FP}} -p \cdot t - C_i$ |
| | FRAUD | $\sum_{t \in T_{FN}} -t$ | $\sum_{t \in T_{TP}} p \cdot t - C_i$ |

Figure 6.20: Matrix with costs and gains associated to the predictions.

The costs specified in this matrix are transaction specific, as each of the purchase orders have a different amount associated.

From this matrix, where each element is denoted as $M_i$ (being $i$ is an element of the confusion matrix), we can aggregate all costs using the $F2$ measure to give more importance to recall:

$$F_2\text{-cost} = \frac{5 \cdot M_{TP}}{5 \cdot M_{TP} + 4 \cdot M_{FN} + M_{FP}}$$

The problem with this formula is that only works for a specific threshold. Besides, it is hard to understand, so it is unlikely to be adopted by the business. The positive and negative values in the matrix makes it difficult to interpret, specially when combining the values using the $F_2$-cost.

Most metrics used in the business have into account the amount of money saved, which is what this matrix is capturing.

Another option could be to use costs to choose the best possible threshold instead of using metrics derived from the confusion matrix. It would be interesting to design a metric that can help us select the best possible threshold for our model having into account the transaction costs, the investigation costs and the probability of mislabeled samples.

Choosing the appropriate metric is complex, as it is highly related with the business logic. Further experiments and comparisons between different cost-based metrics need to be performed in order to have a clearer understanding of which metric to use.

## 6.11 Business evaluation

The hyperparameters used for business evaluation are listed in Table 6.12.

Table 6.12: Hyperparameters of RF model used in business evaluation.

| Hyperparameter | Value |
|---|---|
| Max depth | 26 |
| Number of trees | 100 |
| Impurity measure | entropy |
| Feature subset strategy | onethird |
| Subsampling rate | 0.3 |
| Oversampling rate | 80 |
| Unbalanced rate | 8 |

This evaluation was made before deploying our model at Qliro.

Using the $F_2$ curve by threshold shown in Figure 6.21 we have determined that the best probability threshold to use in our model is 0.3.

The values of the metrics obtained in our experiments are not presented due to privacy concerns. The catch rate was also used to evaluate our models, but those results are not discussed here due to their sensitivity.

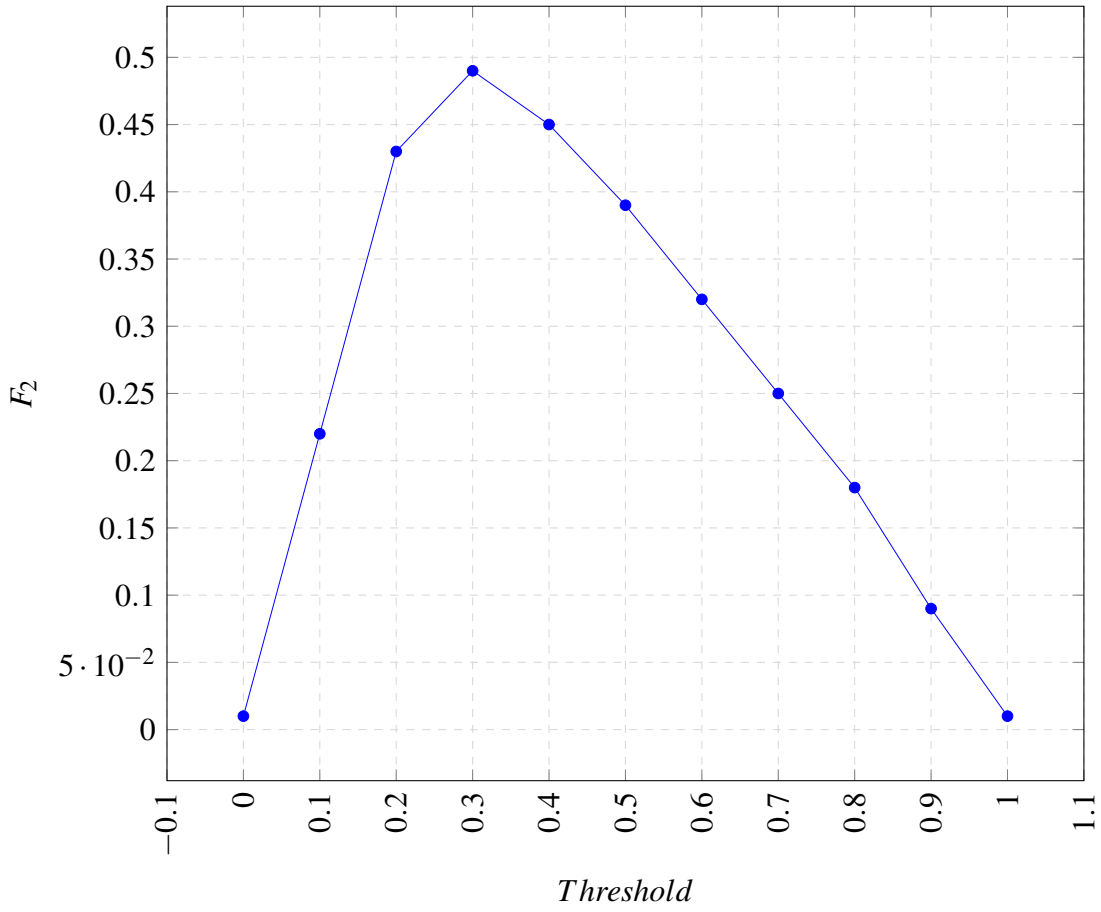
In Figure 6.22 we show a comparison of our model against the scoring rules of the FDS and the investigators' decisions. This comparison is made in terms of the $F_2$ score, although the values are not specified in the $y$ axis due to privacy concerns. Two different probability thresholds have been used to compare their performance: $M_{0.5}$ uses a threshold of 0.5 and $M_{0.3}$ uses 0.3, which is the best one regarding the $F_2$ measure.

We observe that the $F_2$ score of our models is far superior than the scoring rules alone. If we compare the model outcome with the decisions of the investigators, the performance is similar. Our best RF model achieves a slightly better $F_2$ score. However, the test data used is from the same time period as the training set, so CD changes are not contemplated. This means that our model could potentially perform slightly worse than the investigators.

Although the results obtained from our model and the ones from the investigators are similar, we can't use our model alone without the investigators' supervision because the precision of our model is not high enough, so the FPs would end up causing losses to the company.

The main objective of our model is to help catching frauds that were being missed. Some of the predictions of our model are new frauds not detected before

Figure 6.21: Best threshold selection for the deployed model.



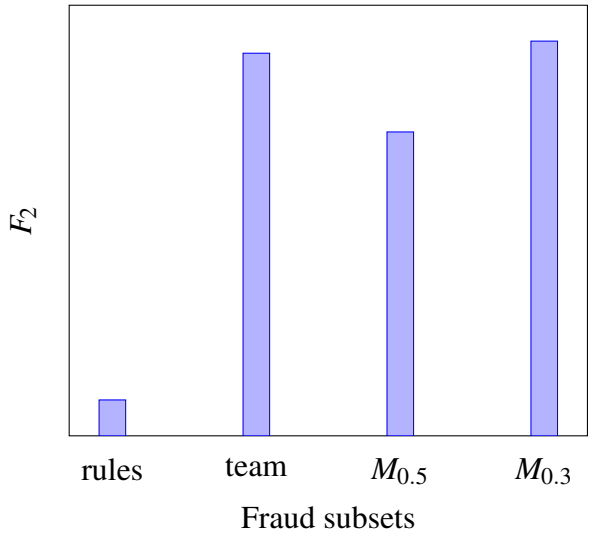



Figure 6.22: Comparison with previous FDS and investigators.

by the investigators. Others were already been caught, but some of them weren't detected by the scoring rules, so investigators had to dedicate extra time to look for them. Providing these cases could save manual work to the investigators which could access them easily just checking the output of our model.

To evaluate the business objectives specified in Section 5.1 we distinguish three different subsets of our model's predictions:

- **Accepted:** possible frauds that our model is sending to the investigators. The ones removed are stopped due to decisions taken by the terminal or the transaction blocking rules, the first two steps in the FDS.
- **Missed:** frauds that our model sends to the investigators and that were missed by the scoring rules. Some of them could be caught thanks to manual work done by the investigators.
- **New:** frauds sent to the investigators missed by the scoring rules and the fraud team. These are the most valuable frauds detected by our model.

In Figure 6.23 we observe the precision, recall and $F_2$ for all the predictions and for the three subsets previously defined. The metrics are all computed in relation with the total values of TPs, FPs and FNs retrieved by our model. The percent values are not presented in the $y$ axis due to privacy issues.

We can see that most of our predictions are accepted, so almost all of our model's predictions are handed out to the fraud team. However, most of them are already detected by the scoring rules, so there is a big decrease in the recall

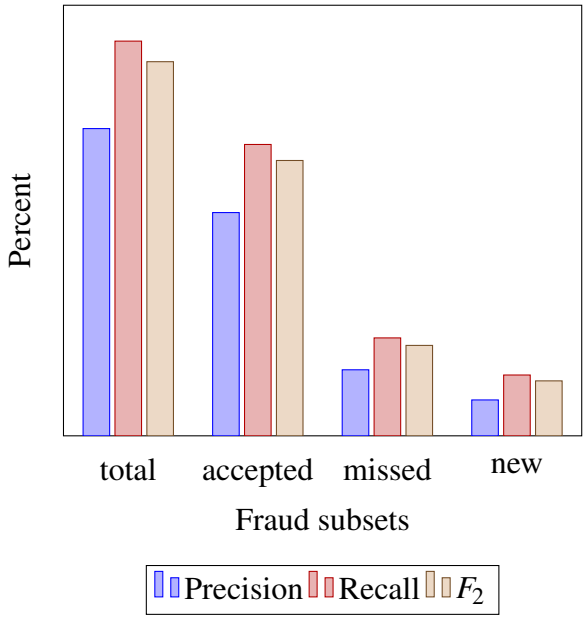



Figure 6.23: Business evaluation for the deployed model with threshold 0.3.

when comparing the accepted frauds and the missed ones. However, this jump is no so big between the missed and the new ones detected. This is possibly due to the difficulty of manually finding the frauds that were not detected by the scoring rules.

A significant amount of new frauds are being retrieved by our model, meeting the business goals previously defined.

## 6.12 Scalability

These experiments analyze our models' training time regarding the number of executors and the memory used.

The hyperparameters of the RF models used in these experiments are shown in Table 6.13.

In Figure 6.24 we observe that the training decreases significantly adding more executors. However, the increase of time performance after 24 executors is not worth it taking into account the amount of extra resources needed.

We can see in Figure 6.25 that the time using 2 Gb for the executors memory and 1 Gb for the driver's memory differs from the result in Figure 6.24. The reason for this is that those were two different experiments and the time is non-deterministic and can vary due to multiple reasons.

The memory used per executor doesn't seem to affect much and the improvements

Table 6.13: Hyperparameters of RF model used in scalability experiments.

| Hyperparameter | Value |
|---|---|
| Max depth | 18 |
| Number of trees | 100 |
| Impurity measure | entropy |
| Feature subset strategy | onethird |
| Subsampling rate | 0.3 |
| Oversampling rate | 80 |
| Unbalanced rate | 2 |

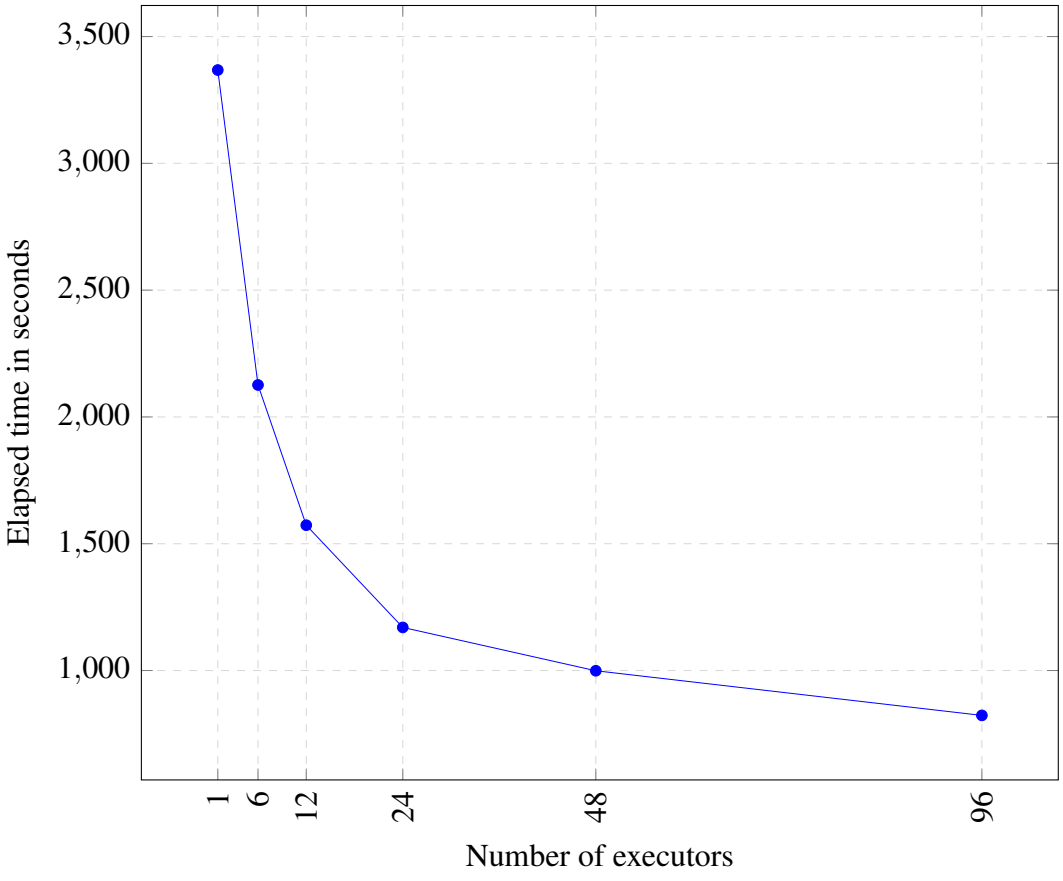



Figure 6.24: Comparison of training elapsed time regarding the number of executors with 1 Gb driver and 2 Gb per executor.

are very small. It is surprising the higher amount of time when the driver has 8 Gb. Having more memory for the driver when it is not needed cause overhead, increasing the training time. However, the reason for 8 Gb being so inefficient could be caused by particular cluster conditions at the time of the experiments.

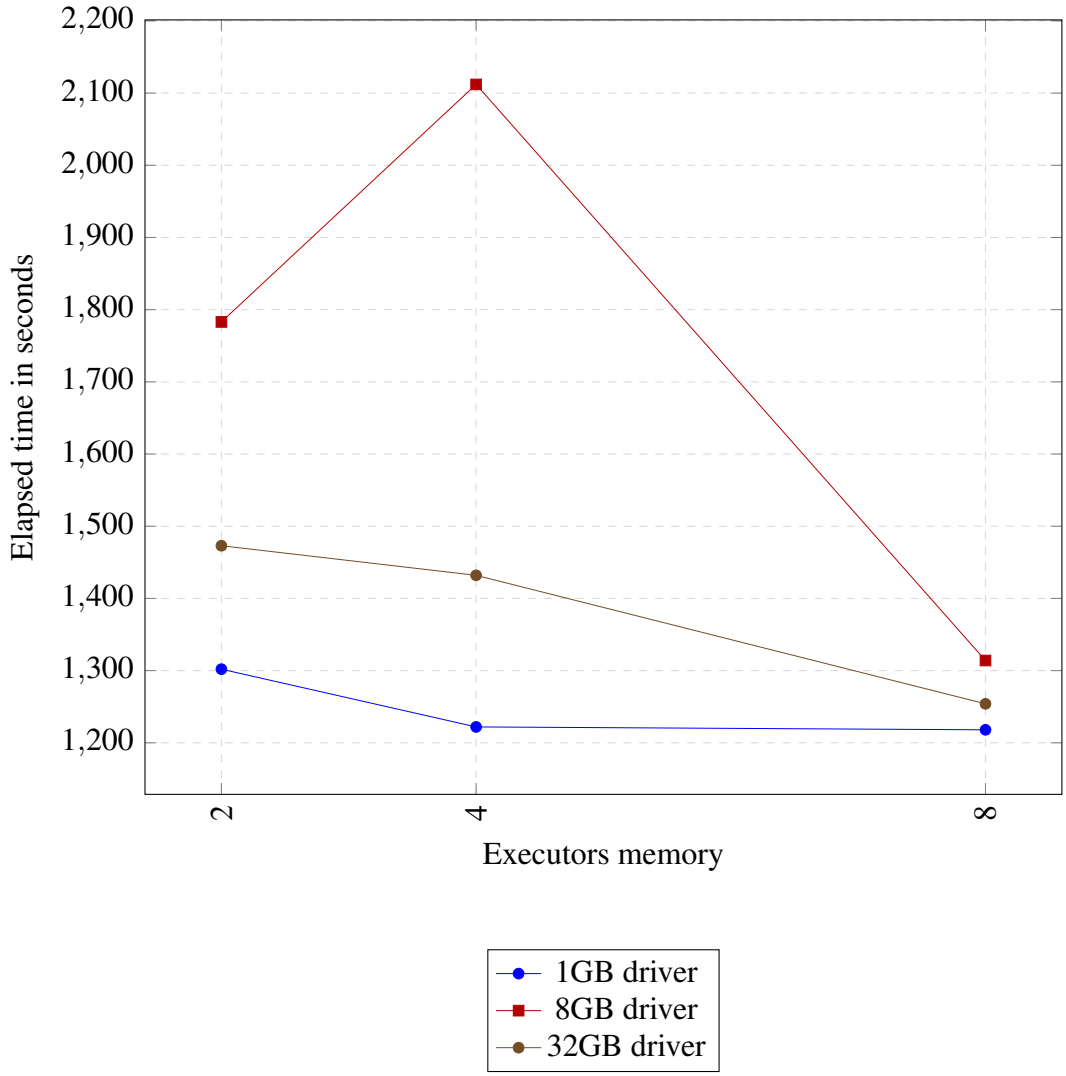



Figure 6.25: Comparison of training elapsed time regarding the executor memory using 12 executors.

We can conclude that our models scale with more data and that the amount of memory assigned doesn't play an important role reducing the training time.

# 7

# Discussion

*"The difficulty lies not so much in developing new ideas as in escaping from old ones. "*
– J. M. Keynes, *The General Theory of Employment and Money*

This chapter summarizes the results exposed in the previous section and discusses their implication in the fraud detection field, suggesting future improvements for some of the open issues. First, the findings are summarized, presenting the learned lessons. Then, future lines of investigation are suggested.

## 7.1 Summary of findings

After all our experiments, we have reaffirmed that fraud detection is a very hard task due to the noisy borderline between frauds and non-frauds. Therefore, improving the ETL and feature engineering performed would create variables with higher predictive power, making that borderline clearer, so frauds could be more easily identified. Particularly, creating variables that aggregate customer history would be very beneficial.

RF models perform well for this kind of problem because they can handle categories and they are not very sensitive to noisy data. Besides, most of the literature work reviewed considered RF the best ML algorithm for fraud detection.

Class imbalance affects enormously our results, so undersampling is key for achieving good results. Oversampling increases the overfitting, but also helps when dealing with very unbalanced datasets, as more non-frauds can be considered without increasing the undersampling ratio. This is very important to increase the precision of our results. The loss of recall derived from the oversampling is bearable because it boosts the precision a lot. However, creating

synthetic samples using SMOTE-NC adds too much noise to our data because having a lot of categories seems to affect negatively to this technique.

When using a high oversampling rate, the models work better with the highest possible balance ratio. However, if we want to control the loss in the recall an upper limit can be found using the the $F_2$ measure.

The other problem that affects us is the non-stationarity, which shouldn't be underestimated. Updating our model as frequently as our data allows us is very important, specially because the behavior of fraudsters changes very fast. Features regarding the time of the year were helpful, but restrict ourselves to use datasets containing a whole year. It would be interesting to remove those features and rerun the experiments to analyze if the models using the most recent data yield better results.

Ensembling different models improved the precision slightly and it could be used to combine models trained in different time periods to capture past data and deal with the CD.

The alert and feedback interaction didn't seem to work in our case. However, applying clustering or using the scoring rules to detect specific populations inside our feedbacks and alerts where hard cases are easily detected is a great field for further research.

Deciding which metric to use have been also problematic, so more work needs to be put into deciding which one is the best metric for fraud detection.

Evaluating the problem in business terms was underestimated, as it took more time than expected. All the comparisons were performed in monetary terms, as the ML metrics were not meaningful to the business stakeholders. This means that a gap between our modeling and the business evaluation exists due to the absence of a cost-based metric. Studying the positive or negative effect on using that metric should be explored.

Using Spark made all the implementation process much harder. Also, Spark ML lacks a lot of features right now, which make it more suitable for easier problems. Another frameworks, such as H2O, which also have distributed ML algorithms could be explored.

The deployment of our models took time, but not as much as expected, specially because we had it in mind during the rest of the development, so some implementation issues had already been dealt with.

In general, the outcome of the project has been a success, as we have detected frauds that are valuable for Qliro, applying our findings for dealing with fraud detection using machine learning.

## 7.2 Future work

In this section we talk about the future improvements that could expand the knowledge and solve some of the technical limitations of this work.

- **Creating a fraud detection collaborative dataset:** using mocked real world data a dataset could be created. This dataset could be used as a standard to compare academic results in the fraud detection task.

  Collaboration across the community is hard. Creating competitions around public datasets, such as the ImageNet competition in the computer vision field [102], challenges the community and boosts up advances in the area.

- **Comparing different distributed frameworks:** this could help overcoming the limitations of the algorithms implemented in Spark ML. Spark is a distributed framework mainly used for ETL, so using other libraries specialized solely on ML might be better. For instance, H2O implements distributed ML algorithms and has more features than Spark ML. Comparing those algorithms and studying their differences could be interesting. It is worth noting that H2O can be easily integrated with Spark using Sparkling Water, which eases the comparison between them as the data preparation can be shared.

- **Exploring alternative fraud detection approaches:** this thesis approaches the fraud detection problem from the perspective of a binary classification problem. However, other approaches can be also used and might be preferred in other use cases. Analyzing them in depth and comparing them could be very interesting. Some of the suggested for further study are:

  - **Small disjuncts problem:** tackling the fraud detection problem from the point of view of small disjuncts, where detecting intra classes in the frauds is the main purpose. This problem is explained in Section 3.1.
  - **Network analysis:** modeling the problem as a graph of the actions of the users in the system to detect previous patterns that can lead to fraud.
  - **Unsupervised analysis:** using unsupervised learning to improve the sampling of the classes, removing the noisy fraud and non-fraud cases.

- **Comparing SMOTE variants:** compare algorithms based on SMOTE that use KNN. It should be interesting to analyze the effect of categorical variables in those approaches. SMOTE is usually used as a baseline

to compare other approaches, but as in our case this technique doesn't improve the results, comparing the SMOTE variants against oversampling and undersampling instead could be interesting.

- **Alleviating the problem of mislabeled purchase orders:** both, the investigators' feedbacks and the alert claims can be incorrect. We have not tackled this problem in this thesis, but it could be solved by creating a FDS specifically designed to detect the mislabeled cases.

- **Researching about fraud detection metrics:** a standard evaluation metric in fraud detection doesn't exist. It would be interesting to perform a formal comparison between cost-based metrics and traditional ones. It would be also helpful to explore if using the PR-gain curve [15] can solve some of the problems of the PR curve.

# 8

# Conclusion

*"The end was contained in the beginning."*
– George Orwell, *1984*

In this thesis we have implemented a real world Fraud Detection System using Spark ML, which periodically sends possible fraud cases to investigators. They can decide the amount of frauds to review by filtering our model predictions using a confidence threshold we provide.

Experiments to deal with class imbalance, concept drift and alert-feedback interaction have been performed, so the final model deployed leverages all our findings to achieve good results in terms of AU-PR, a suitable metric to optimize in fraud detection when the number of frauds to obtain is not well defined.

We found that oversampling and undersampling are key for solving the class imbalance. Additionally, ensembling several models with different undersampling of the majority class improves the precision.

The project has been developed following the CRISP-DM methodology, putting special attention on addressing the most common problems derived from it.

Increasing the cluster resources when training the models speeds up the jobs significantly, so our solution scales with bigger amounts of data. There is a lack of resources in Spark ML to deal with class imbalance, so we have open sourced a repository with helpful code for working with unbalanced data using Spark ML.

The findings from this thesis can be applied by other researchers working to solve real-world problems in fraud detection, but also by companies building FDSs, as our findings can be extrapolated to other similar noisy datasets in the industry.

# Appendix A

# Versions and components table

In Table A.1 the different versions of the components used are specified. These components were managed by the HDP 2.6 distribution. Some other components from this distribution that are not relevant for this thesis are not specified here*.

Table A.1: Versions used

| Name | Version |
|---|---|
| Scala | 2.11.8 |
| Spark | 2.1.0 |
| Zeppelin | 0.7.0 |
| Hive | 1.2.1 |
| Sqoop | 1.4.6 |
| Oozie | 4.2.0 |
| Ambari | 2.5.0 |

* All the components and their versions can be found in https://es.hortonworks.com/products/data-center/hdp/.

# Appendix B

# Project spark-imbalance

Some of the code of this thesis has been made public in a GitHub repository*.

It contains some implementations that aim to ease the process of dealing with class imbalance when using Spark ML.

The main features implemented are:

- Stratified cross validation.
- SMOTE-NC.
- Spark tree visualizer.
- UDFs to ensemble model predictions.

In Listing B.1 we can see an example of how to create a RF model using stratified cross validation with the implementations we provide.

Listing B.1: Stratified cross validation

```
val randomForestCrossValidation =
  new RandomForestCrossValidation(
    labelIndexer,
    indexerPipeline,
    trainingData,
    testData
    )

val performanceGrid = randomForestCrossValidation.combined

val bestModel = randomForestCrossValidation.bestModel
```

* https://github.com/xinofekuator/spark-imbalance.git.

In Figure B.1 we present an example of a decision tree obtained using the Spark tree visualizer. This tree is obtained in JSON format and it is plotted using an online JSON visualizer*.

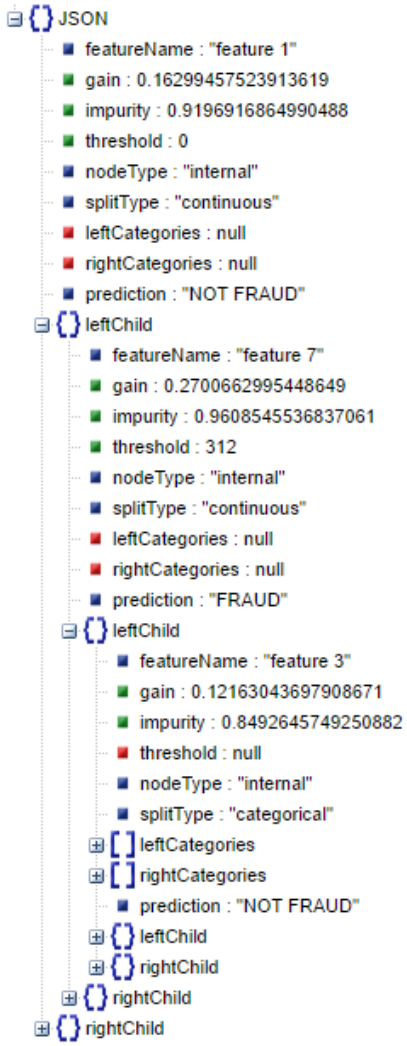



Figure B.1: Example of an output obtained using the Spark tree visualizer.

It could be possible to extend the visualizer to see the tree visually, but due to the big depth of the trees this was left undone because it is not very useful.

* http://jsonviewer.stack.hu/.

# Bibliography


[1] M. Araujo, M. Almeida, J. Ferreira, L. Silva, and P. Bizarro, "Breachradar: Automatic detection of points-of-compromise," in *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 2017, pp. 561–569.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[3] A. Dal Pozzolo and G. Bontempi, "Adaptive machine learning for credit card fraud detection," 2015.

[4] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[5] C. Shearer, "The CRISP-DM model: the new blueprint for data mining," *Journal of data warehousing*, vol. 5, no. 4, pp. 13–22, 2000.

[6] G. Piatetsky, "CRISP-DM, still the top methodology for analytics, data mining, or data science projects," http://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html, accessed: 2017-08-06.

[7] J. Taylor, "Four problems in using CRISP-DM and how to fix them," http://www.kdnuggets.com/2017/01/four-problems-crisp-dm-fix.html, accessed: 2017-08-06.

[8] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, "Machine learning: The high interest credit card of technical debt," in *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.

[9] J. Haffar, "Have you seen ASUM-DM?" https://developer.ibm.com/predictiveanalytics/2015/10/16/have-you-seen-asum-dm/, accessed: 2017-08-06.

[10] A. Shah, "Machine Learning vs Statistics," http://www.kdnuggets.com/2016/11/machine-learning-vs-statistics.html, accessed: 2017-08-06.

[11] Y. Lee, "Support vector machines for classification: a statistical portrait," *Statistical Methods in Molecular Biology*, pp. 347–368, 2010.

[12] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2. Stanford, CA, 1995, pp. 1137–1145.

[13] S. Fortmann-Roe, "Understanding the bias-variance tradeoff," 2012.

[14] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics Springer, Berlin, 2001, vol. 1, ch. 7, pp. 223–228.

[15] P. Flach and M. Kull, "Precision-recall-gain curves: Pr analysis done right," in *Advances in Neural Information Processing Systems*, 2015, pp. 838–846.

[16] N. V. Chawla, D. A. Cieslak, L. O. Hall, and A. Joshi, "Automatically countering imbalance and its empirical relationship to cost," *Data Mining and Knowledge Discovery*, vol. 17, no. 2, pp. 225–252, 2008.

[17] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 233–240.

[18] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PloS one*, vol. 10, no. 3, 2015.

[19] R. J. Bolton, D. J. Hand *et al.*, "Unsupervised profiling methods for fraud detection," *Credit Scoring and Credit Control VII*, pp. 235–255, 2001.

[20] D. K. Tasoulis, N. M. Adams, and D. J. Hand, "Unsupervised clustering in streaming data," in *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*. IEEE, 2006, pp. 638–642.

[21] S. Sperandei, "Understanding logistic regression analysis," *Biochemia medica: Biochemia medica*, vol. 24, no. 1, pp. 12–18, 2014.

[22] G. King and L. Zeng, "Logistic regression in rare events data," *Political analysis*, vol. 9, no. 2, pp. 137–163, 2001.

[23] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[24] "Spark mllib decision trees improvement," https://issues.apache.org/jira/browse/SPARK-8078, accessed: 2017-08-22.

[25] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, 2006.

[26] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data," Department of Statistics, University of Berkeley, Tech. Rep., 2004. [Online]. Available: http://www.stat.berkeley.edu/users/chenchao/666.pdf

[28] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Advances in neural information processing systems*, 2013, pp. 431–439.

[29] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics Springer, Berlin, 2001, vol. 1, ch. 10, pp. 367–369.

[30] "H2O," https://www.h2o.ai/, accessed: 2017-08-07.

[31] "Apache Flink," http://flink.apache.org/, accessed: 2017-08-07.

[32] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[33] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[34] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[35] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1383–1394.

[36] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters." *HotCloud*, vol. 12, pp. 10–10, 2012.

[37] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework." in *OSDI*, vol. 14, 2014, pp. 599–613.

[38] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[39] "Evolution of card fraud in europe," http://www.fico.com/europeanfraud/sweden, accessed: 2017-08-16.

[40] Y. Sahin, S. Bulkan, and E. Duman, "A cost-sensitive decision tree approach for fraud detection," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5916–5923, 2013.

[41] A. C. Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, "Cost sensitive credit card fraud detection using bayes minimum risk," in *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, vol. 1. IEEE, 2013, pp. 333–338.

[42] C. Elkan, "The foundations of cost-sensitive learning," in *International joint conference on artificial intelligence*, vol. 17, no. 1. Lawrence Erlbaum Associates Ltd, 2001, pp. 973–978.

[43] A. C. Bahnsen, D. Aouada, and B. Ottersten, "Example-dependent cost-sensitive decision trees," *Expert Systems with Applications*, vol. 42, no. 19, pp. 6609–6619, 2015.

[44] G. Fan and M. Zhu, "Detection of rare items with target," *Statistics and Its Interface*, vol. 4, no. 1, pp. 11–17, 2011.

[45] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, "Data mining for credit card fraud: A comparative study," *Decision Support Systems*, vol. 50, no. 3, pp. 602–613, 2011.

[46] G. E. Batista, A. C. Carvalho, and M. C. Monard, "Applying one-sided selection to unbalanced datasets," in *MICAI*, vol. 2000. Springer, 2000, pp. 315–325.

[47] G. M. Weiss and F. Provost, "The effect of class distribution on classifier learning: an empirical study," *Rutgers Univ*, 2001.

[48] A. Estabrooks, T. Jo, and N. Japkowicz, "A multiple resampling method for learning from imbalanced data sets," *Computational intelligence*, vol. 20, no. 1, pp. 18–36, 2004.

[49] T. Jo and N. Japkowicz, "Class imbalances versus small disjuncts," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 40–49, 2004.

[50] R. C. Prati, G. Batista, M. C. Monard *et al.*, "Class imbalances versus class overlapping: an analysis of a learning system behavior," in *MICAI*, vol. 4. Springer, 2004, pp. 312–321.

[51] A. Dal Pozzolo, O. Caelen, S. Waterschoot, and G. Bontempi, "Racing for unbalanced methods selection," in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2013, pp. 24–31.

[52] C. Drummond, R. C. Holte *et al.*, "C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," in *Workshop on learning from imbalanced datasets II*, vol. 11. Citeseer Washington DC, 2003.

[53] S.-J. Yen and Y.-S. Lee, "Cluster-based under-sampling approaches for imbalanced data distributions," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5718–5727, 2009.

[54] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," *Advances in intelligent computing*, pp. 878–887, 2005.

[55] S. Hu, Y. Liang, L. Ma, and Y. He, "Msmote: improving classification performance when training data is imbalanced," in *Computer Science and Engineering, 2009. WCSE'09. Second International Workshop on*, vol. 2. IEEE, 2009, pp. 13–17.

[56] L. Ma and S. Fan, "Cure-smote algorithm and hybrid algorithm for feature selection and parameter optimization based on random forests," *BMC bioinformatics*, vol. 18, no. 1, p. 169, 2017.

[57] H. He, Y. Bai, E. A. Garcia, and S. Li, "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*. IEEE, 2008, pp. 1322–1328.

[58] E. Fix and J. L. Hodges Jr, "Discriminatory analysis-nonparametric discrimination: consistency properties," California Univ Berkeley, Tech. Rep., 1951.

[59] R. Blagus and L. Lusa, "Evaluation of smote for high-dimensional class-imbalanced microarray data," in *Machine learning and applications (icmla), 2012 11th international conference on*, vol. 2. IEEE, 2012, pp. 89–94.

[60] I. Tomek, "Two modifications of cnn," *IEEE Trans. Systems, Man and Cybernetics*, vol. 6, pp. 769–772, 1976.

[61] S. Suman, K. Laddhad, and U. Deshmukh, "Methods for handling highly skewed datasets," *Part I-October*, vol. 3, 2005.

[62] P. Hart, "The condensed nearest neighbor rule (corresp.)," *IEEE transactions on information theory*, vol. 14, no. 3, pp. 515–516, 1968.

[63] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, pp. 408–421, 1972.

[64] J. Laurikkala, "Improving identification of difficult small classes by balancing class distribution," *Artificial Intelligence in Medicine*, pp. 63–66, 2001.

[65] G. M. Weiss, "Foundations of imbalanced learning," *Imbalanced Learning: Foundations, Algorithms, and Applications*, p. 73, 2013.

[66] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent data analysis*, vol. 6, no. 5, pp. 429–449, 2002.

[67] P.-H. Chen, C.-J. Lin, and B. Schölkopf, "A tutorial on $\nu$-support vector machines," *Applied Stochastic Models in Business and Industry*, vol. 21, no. 2, pp. 111–136, 2005.

[68] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[69] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[70] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[71] R. E. Schapire, "Explaining adaboost," in *Empirical inference*. Springer, 2013, pp. 37–52.

[72] J. R. Quinlan *et al.*, "Bagging, boosting, and c4. 5," in *AAAI/IAAI, Vol. 1*, 1996, pp. 725–730.

[73] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.

[74] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2009.

[75] X. Zhang and C. Cheng, "Imbalanced data classification algorithm based on boosting and cascade model," in *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2861–2866.

[76] N. Chawla, A. Lazarevic, L. Hall, and K. Bowyer, "Smoteboost: Improving prediction of the minority class in boosting," *Knowledge Discovery in Databases: PKDD 2003*, pp. 107–119, 2003.

[77] D. Mease, A. Wyner, and A. Buja, "Cost-weighted boosting with jittering and over/under-sampling: Jous-boost," *J. Machine Learning Research*, vol. 8, pp. 409–439, 2007.

[78] L. Breiman, "Stacked regressions," *Machine learning*, vol. 24, no. 1, pp. 49–64, 1996.

[79] C. Phua, D. Alahakoon, and V. Lee, "Minority report in fraud detection: classification of skewed data," *Acm sigkdd explorations newsletter*, vol. 6, no. 1, pp. 50–59, 2004.

[80] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–I.

[81] H. Wu, K. Deng, and J. Liang, "Machine learning based automatic detection of pulmonary trunk," in *Proc. of SPIE Vol*, vol. 7963, 2011, pp. 79 630K–1.

[82] 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*. ACM, 1999, pp. 155–164.

[83] M. Maloof, P. Langley, S. Sage, and T. Binford, "Learning to detect rooftops in aerial images," in *Proceedings of the Image Understanding Workshop*, 1997, pp. 835–845.

[84] M. Saerens, P. Latinne, and C. Decaestecker, "Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure," *Neural computation*, vol. 14, no. 1, pp. 21–41, 2002.

[85] M. Sugiyama, M. Krauledat, and K.-R. MÃžller, "Covariate shift adaptation by importance weighted cross validation," *Journal of Machine Learning Research*, vol. 8, no. May, pp. 985–1005, 2007.

[86] W. H. Greene, "Sample selection bias as a specification error: A comment," *Econometrica: Journal of the Econometric Society*, pp. 795–798, 1981.

[87] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

[88] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset shift in machine learning*. The MIT Press, 2009.

[89] C. Alippi, G. Boracchi, and M. Roveri, "Just-in-time classifiers for recurrent concepts," *IEEE transactions on neural networks and learning systems*, vol. 24, no. 4, pp. 620–634, 2013.

[90] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural networks*, vol. 1, no. 1, pp. 17–61, 1988.

[91] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[92] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, no. Dec, pp. 2755–2790, 2007.

[93] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine learning*, vol. 23, no. 1, pp. 69–101, 1996.

[94] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, vol. 31, no. 4, pp. 497–508, 2001.

[95] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 377–382.

[96] J. Gao, W. Fan, J. Han, and P. S. Yu, "A general framework for mining concept-drifting data streams with skewed distributions," in *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 2007, pp. 3–14.

[97] R. N. Lichtenwalter and N. V. Chawla, "Learning to classify data streams with imbalanced class distributions," *New Frontiers in Applied Data Mining. LNCS. Springer, Heidelberg*, 2009.

[98] D. A. Cieslak, T. R. Hoens, N. V. Chawla, and W. P. Kegelmeyer, "Hellinger distance decision trees are robust and skew-insensitive," *Data Mining and Knowledge Discovery*, vol. 24, no. 1, pp. 136–158, 2012.

[99] R. Kohavi and R. Quinlan, "C5. 1.3 decision tree discovery," 1999.

[100] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[101] S.-T. Yeh, "Using trapezoidal rule for the area under a curve calculation," *Proceedings of the 27th Annual SAS® User Group International (SUGI'02)*, 2002.

[102] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

TRITA Trita-ICT-EX-2017:153