Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

POLITECNICO
MILANO 1863

Amaretto: an active learning framework for money laundering detection

Relatore: **Prof. Stefano Zanero**
Correlatore: **Dott. Michele Carminati**
Ing. Luca Primerano

Tesi di Laurea Magistrale di:
Danilo Labanca, Matr. 878457

Anno Accademico 2018-2019

II

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AML Anti-Money Laundering

ML Machine Learning

DL Deep Learning

TPR *True Positive Rate*

FPR *False Positive Rate*

TP *True Positive*

FP *False Positive*

TN *True Negative*

FN *False Negative*

MCC *Matthews Correlation Coefficient*

ACC *Accuracy*

PREC *Precision*

IF *Isolation Forest*

RF *Random Forest*

AE *Auto Encoder*

XGB *eXtreme Gradient Bosting*

CB *CatBoost*

SVM *Support Vector Machine*

CDF Cumulative Distribution Function

PDF Probability Distribution Function

MSE Mean Squared Error

AUROC Area Under Curve - Receiving Operator Characteristic

Abstract

Monitoring financial transactions is a critical anti-money laundering (AML) obligation for all financial institutions. Traditional anti-money laundering approaches are based on heuristics and static rules to highlight unusual behaviours. Such approaches generate a high number of false positives, false negatives, and require a substantial effort to manually review all alerts generated by these systems. Moreover, transactional data is large in volume and highly imbalanced, often having less than 1% of anomalous occurrences. In recent years, several advanced statistical models, as well as machine learning-based systems, have been successfully used to complement traditional rule-based systems. Unfortunately, these solutions also have disadvantages: even if unsupervised models don't require human intervention, they lead to low performance resulting in a high number of false positives; while supervised models require a large amount of labelled data to perform adequately and achieve high detection rate.

In this work we present Amaretto, an active learning framework for money laundering detection that combines supervised and unsupervised learning techniques, taking advantage of their strengths, to improve AML transaction monitoring processes by targeting a subset of transactions for investigation and making more efficient use of the feedback provided by the analyst. We experimentally evaluated Amaretto, on a synthetic dataset simulating a real-world scenario. We show that our approach outperforms state-of-the-art solutions by improving both the detection rate and the precision by 25% and achieving an overall detection rate of 0.6 and an area under the ROC curve (AUROC) of 0.94, with a limited set of labels, showing an improvement in performance after each successive iterations.

Sommario

Il riciclaggio di denaro comprende qualsiasi processo attraverso il quale le entrate di attività illecite come il traffico di droga, il traffico illegale di armi, l'evasione fiscale o altre attività criminali sono introdotte nel sistema finanziario attraverso molteplici operazioni che nascondono le loro origini illecite. Al giorno d'oggi, il riciclaggio di denaro interessa tutte le economie mondiali ed è responsabile della generazione di profitti stimati tra 1.6-2.85$ trilioni all'anno, equivalenti al 2,1% e al 4% del Prodotto Mondiale Lordo [1]. Per molti anni, i governi e le forze dell'ordine hanno affrontato l'evoluzione delle attività criminali diventate sempre più sofisticate, introducendo regolamenti per ridurre al minimo l'impatto del riciclaggio di denaro e rimuoverne le cause. Ad esempio, la Financial Conduct Authority (FCA) suggerisce nelle sue linee guida di monitorare ripetutamente le attività dei clienti [2].

Il monitoraggio delle transazioni in materia di antiriciclaggio consiste in una serie di attività svolte da analisti e sistemi automatizzati per controllare le transazioni dei clienti al fine di rilevare comportamenti sospetti che possono essere collegati a schemi di riciclaggio. Le transazioni finanziarie potrebbero includere trasferimenti bancari, pagamenti con carta di credito o transazioni in banche di investimento, come operazioni su azioni. Il primo passo per implementare adeguate procedure di antiriciclaggio consiste nell'implementare sistemi configurati con una serie di regole predefinite che supportano i processi di monitoraggio delle transazioni. Questi sistemi generano avvisi se tali regole vengono attivate. I vantaggi degli approcci basati su euristiche sono principalmente legati alla facilità di interpretazione dell'output del sistema e alla capacità degli esperti in materia di utilizzare facilmente tali informazioni. Purtroppo, le tecniche di riciclaggio di denaro e il crimine finanziario sono in continua evoluzione e quindi le regole devono essere aggiornate per garantire che siano idonee a cogliere gli scenari in evoluzione e questo rappresenta la più grande vulnerabilità di questi sistemi. Inoltre, le regole possono riguardare solo comportamenti anomali noti e non possono rileveare comportamenti insoliti sconosciuti, con conseguenti falsi negativi. Il fatto che le regole debbano essere configurate utilizzando predeterminate soglie statiche comporta un elevato numero di falsi positivi e, successivamente, un

aumento del volume delle indagini manuali. Tecniche di machine learning (ML) e deep learning (DL) migliorano le tradizionali tecniche di antiriciclaggio superando alcune delle insidie nei sistemi basati su regole. I modelli di ML non si limitano ad un insieme di regole predefinite, ma possono estratte e analizzare schemi dai dati e valutare correlazioni insolite che potrebbero essere sconosciute agli esperti in materia. Sfortuntamente, queste soluzioni presentano, a loro volta, degli svantaggi. tecniche supervisionate richiedono un'elevata quantità di dati etichettati per poter funzionare correttamente e ottenere un alto tasso di rilevamento.

Al fine di raccogliere un insieme di dati etichettati nel modo più rapido ed efficiente possibile, i sistemi moderni possono sfruttare tecniche di active learning (AL). L'active learning è una tecnica aggiuntiva che utilizza modelli di machine learning per selezionare le transazioni da investigare che hanno la più alta probabilità di migliorare la qualità dell'output del sistema di machine learning.

In questo documento, presentiamo Amaretto, un sistema di active learning che combina modelli non supervisionati e supervisionati organizzati in un framework che combina il prezioso feedback di un analista. Il modello non supervisionato consente al sistema di rilevare anomalie sconosciute e nuovi schemi mai visti prima, mentre il modello supervisionato può utilizzare dati precedentemente etichettati da analist per migliorare in modo rapido il tasso di rilevamento. Il sistema è in grado di preelaborare i dati transazionali grezzi, convertendoli in vettori aggregati. I modelli non supervisionati e supervisionati prendono questi vettori come input e calcolano un punteggio di anomalia per ciascun vettore. In seguito, viene applicata la strategia di selezione per individuare i campioni che verranno esaminati dall'analista. Quindi, il feedback raccolto dall'analista viene utilizzato come dati di addestramento per il modello supervisionato che calcolerà il punteggio finale per i vettori di aggregazione.

Infine, presentiamo la valutazione sperimentale condotta su un set di dati sintetico fornito dal partner con cui abbiamo collaborato, che è stato generato da un insieme di dati che ingloba i tipici schemi di riciclaggio di denaro. In primo luogo, confrontiamo le tecniche non supervisionate comunemente utilizzate nelle attività di rilevamento delle anomalie e nelle soluzioni più all'avanguardia, e dimostriamo che la Isolation Forest è l'algoritmo migliore. Quindi, abbiamo condotto una valutazione simile tra le tecniche supervisionate e concludiamo che Random Forest supera le altre tecniche per questo set di dati. Successivamente, dimostriamo che i contributi forniti da un modello non supervisionato completano la capacità dei modelli supervisionati, essendo in grado di rilevare nuovi tipi di anomalie. Alla fine, conduciamo esperimenti con l'obiettivo di confermare la solidità del design di Amaretto in uno scenario reale, identificando le migliori strategie di selezione da quelle proposte e dimostrando infine che Amaretto supera le soluzioni all'avanguardia migliorando sia il tasso di rileva-

mento che la precisione del 25%, raggiungendo un tasso di rilevamento complessivo di 0,6 e un'area sotto la curva ROC (AUROC) di 0,94.

I contributi di questo lavoro sono i seguenti:

- comparazione di algoritmi supevisionati e non supervisionati applicati all'attività di rilevamento del riciclaggio di denaro;
- prove sperimentali che dimostrano l'importanza di un modello non supervisionato in combinazione con un modello supervisionato per ottenere migliori prestazioni e ridurre i costi (es. indagine manuale);
- sviluppo di un framework di AL con una nuova strategia di selezione per rilevare potenziali schemi di riciclaggio di denaro.

Chapter 1

Introduction

Money laundering encompasses any process by which the income of unlawful activities such as drugs trafficking, illegal arms trafficking, tax evasion or other criminal activities is introduced into the financial system through multiple operations that conceal their illicit origins. Nowadays, money laundering affects all worldwide economies and is responsible for generating profits estimated between $1.6-2.85 trillion per year, equivalent to 2.1% and 4% of the Gross World Product [1]. For many years, governments and law enforcement agencies dealt with the evolution of criminal activities becoming more and more sophisticated, by introducing regulations to minimise the impact of money laundering and to remove its root causes. For example, the Financial Conduct Authority (FCA) suggests in its guidelines to monitor customers' activities repeatedly [2].

Transaction monitoring in anti-money laundering consists of a set of activities carried out by analysts and automated systems to scrutinise customers' transactions to detect suspicious behaviours that may be linked to laundering patterns. Financial transactions could include bank transfers, credit card payments or investment banking transactions such as equity trades. The first step to implementing adequate anti-money laundering (AML) procedures consists of implementing expert systems configured with a set of pre-defined rules that support transaction monitoring processes. These systems generate alerts if such rules are triggered. The benefits of heuristic-based approaches are mainly related to the ease of interpretation of the output of the system and the ability for subject matter experts to easily use that information. The disadvantage of expert systems is that money laundering techniques and financial crime are always evolving and so the rules need to be updated to ensure they are fit to capture the evolving scenarios. Moreover, rules can only cover known anomalous behaviours and they can not detect unknown unusual behaviours resulting in false

negatives. The fact that rules have to be configured using specific static thresholds results in a high number of false positives and subsequently, an increase in the volume of manual investigations. Machine learning and deep learning enhance traditional AML techniques by overcoming some of the pitfalls in rule-based systems. Machine learning models are not limited to a set of pre-defined rules, instead, they can extract and analyse patterns and insights from data and assess unusual correlations that may be unknown to subject matter experts.

In order to collect a valuable set of labelled data as quickly and as efficiently as possible, modern systems can leverage active learning. Active learning is an additional technique that uses machine learning models to select transactions for the investigation that have the highest probability of improving the quality of the output of the supervised machine learning system.

In this paper, we present AMARETTO, an active learning system that combines unsupervised and supervised models organised in an "analyst-in-the-loop" framework. The unsupervised model allows the system to detect unknown anomalies and new patterns that have not been seen before, while the supervised model can use labels previously classified by subject matter expert to improve the detection rate. The system can preprocess the raw transactional data, converting it to aggregated vectors. The unsupervised and supervised models take the vectors as input and compute an anomaly score for each vector. Then, the selection strategy is applied to retrieve the samples that will be reviewed by the analyst. In the end, the feedback collected from the analyst is used as training data for the supervised model that will compute the final score for the aggregation vectors.

Finally, we present the experimental evaluation conducted on a synthetic dataset provided by the industry partners we collaborated with, that was generated from a real-world dataset that resembles typical genuine and potential money laundering patterns. Firstly, we compare unsupervised techniques, commonly used in anomaly detection tasks and in state of the art solutions, and we demonstrate that the Isolation Forest is the best algorithm. Then, we conducted a similar assessment amongst supervised techniques and we conclude that Random Forest outperforms the other techniques for this dataset. Subsequently, we prove the contributions made by an unsupervised model complement the ability of supervised models by being able to detect new types of anomalies. In the end, we conduct experiments with the aim of confirming the robustness of our design in a real-world scenario, identifying the best selection strategies from the ones proposed and finally showing that AMARETTO outperforms the state of the art solutions by improving both the detection rate and the precision by 25% and achieving an overall detection rate of 0.6 and an area under the curve (AUROC) of 0.94.

The contributions of this work are the following:

- comparison of supervised and unsupervised algorithms applied to the task of detecting money laundering focused on highlighting the most effective techniques;
- experimental evidence that demonstrates the importance of an unsupervised model in conjunction with a supervised model to achieve better performance and to reduce cost (i.e., manual investigation);
- development of an active learning framework with a novel selection strategy to detect potential money-laundering patterns.

The structure of this thesis is as follows:

– Chapter 2 presents a description of the anomaly detection task and the money laundering problem; a review of works regarding anti-money laundering and fraud detection in the academic world; an overview of the threat models and the challenges that the problem involves;

– Chapter 3 presents the description of the dataset we used for this work, showing its main characteristics;

– Chapter 4 presents a high-level description of Amaretto;

– Chapter 5 describes the details of every module in the Amaretto system;

– Chapter 6 presents all the experiments that validate the end to end flow that Amaretto implements;

– Chapter 7 we present the limitations of our approach, together with interesting paths for future works;

– Chapter 8 we exhibit our conclusions and a resume of the impact of our work.

Chapter 2

Motivation

In this chapter, we present the motivation lying behind our research work. After providing the necessary background concepts, we present the topic of money laundering and how anomaly detection in that domain is carried out. We conduct then a review of the literature on the subject. Finally, we summarize the goals of our work and the challenges to be faced in achieving such objectives.

2.1 Background

2.1.1 Anomaly Detection

Anomaly detection, also called outlier detection, is the branch of data mining that addresses the problem of discovering instances, called outliers, that do not conform with the expected behaviour of the dataset they belong to. In general, the presence of an outlier in a dataset can be regarded as an interesting occurrence, potentially indicating something unusual, even of critical importance. The application domains of anomaly detection are endless, as stated in [3], and in each domain, the presence of an outlier means something different. For example, in the context of network traffic, an anomaly may indicate the presence of an attacker trying to steal sensitive information [4], while in a Magnetic Resonance Imaging result it could hint the presence of a tumour [5]. When applied to the financial domain, anomaly detection is referred to as fraud detection, and it finds many different applications: frauds are sought for example in online banking [6] and investment funds transactions [7] [8], insurance contracts [9], financial statements [10] and several other contexts. We decided to focus on money laundering in capital markets.

2.1.2 Money Laundering

Money laundering can be considered as one of the most challenging tasks regarding anomaly detection and fraud detection because, due to its multiple typologies, it is a complex problem to formalize. First of all, there isn't a common regulation between states and this makes it difficult to discern which transactions should be considered fraudulent and which are not. Furthermore, some money laundering processes involve a high number of transactions between banks and different countries, camouflaging fraudulent transactions between licit ones and therefore impossible to detect even with automatic systems. Money laundering is a fraudulent activity that can be sum up with the following sentence: *"money laundering is the process of obscuring the source, the ownership or use of funds, usually cash, that are profits of illicit activities"* [11]. In other words, it is how money originated from criminal activities such as drugs smuggling, people trafficking, cybercrime and fraud, in general, is made accessible and usable. Given the illegal nature of the transactions associated with money laundering, it is extremely hard to give accurate estimates of the magnitude of the phenomenon, and very few attempts have been made in this sense: the flow of money it generates has been estimated as $1.6-2.85 trillion per year, responsible for 2.1-4% of GDP [1]. With such a huge impact on the global economy, public and private regulators worldwide propose guidelines and enforce policies and laws concerning anti-money laundering activities, to be complied with financial institutions. An example is given by the UK's Financial Conduct Authority (FCA), which periodically issues such guidelines and regulations to financial firms [2].

The typical structure of one "instance" of money laundering activity, as defined by the intergovernmental organization Financial Action Task Force on Money Laundering (FATF) [12], is composed of three phases:

- **Stage 1: criminal proceeds are transferred to, or collected by, PMLs**: in the first stage, funds are transferred, physically or electronically, to professional money launderers (PMLs) or to entities operating on their behalf. The precise manner of introduction of the funds into the ML scheme varies depending on the types of predicate offence(s) and the form in which criminal proceeds were generated (e.g., cash, bank funds, virtual currency, etc.):

  - *cash*: when illicit proceeds are introduced as currency, they are usually passed over to a cash collector. This collector may ultimately deposit the cash into bank accounts. The collector introduces the cash into the financial system through cash-intensive businesses, MVTS providers or casinos, or physically transports the cash to another region or country.

  - *bank accounts*: some types of criminal activity generate illicit proceeds held in bank accounts, such as fraud, embezzlement and tax crimes. Unlike drug, proceeds of these crimes rarely start out as cash but may end up as cash after laundering. Clients usually establish legal entities under whose names bank accounts may be opened for the purposes of laundering funds. These accounts are used to transfer money to the first layer of companies that are controlled by the PMLs.
  - *virtual currency*: criminals who obtain proceeds in a form of virtual currency (e.g., owners of online illicit stores, including Dark Web marketplaces) must have e-wallets or an address on a distributed ledger platform, which can be accessed by the PMLs.

- **Stage 2: layering stage executed by individuals and/or networks**: in the layering stage, the majority of PMLs use account settlement mechanisms to make it more difficult to trace the funds. A combination of different ML techniques may be used as part of one scheme. The layering stage is managed by individuals responsible for the co-ordination of financial transactions.

  - *cash*: ML mechanisms for the layering of illicit proceeds earned in cash commonly include: TBML and fictitious trade, account settlements and underground banking.
  - *bank accounts*: funds that were transferred to bank accounts managed by PMLs are, in most cases, moved through complex layering schemes or proxy structures. Proxy structures consist of a complex chain of shell company accounts, established both domestically and abroad. The funds from different clients are mixed within the same accounts, which makes the tracing of funds coming from a particular client more difficult.
  - *virtual currency*: criminals engaged in cybercrime or computer-based fraud, as well as in the sale of illicit goods via online stores, often use the services of money mule networks (see Section IV). The illicit proceeds earned from these crimes are often held in the form of virtual currency and are stored in e-wallets or virtual currency wallets that go through a complex chain of transfers.

- **Stage 3: laundered funds are handed back over to clients for investment or asset acquisition**: in the last stage, funds are transferred to accounts controlled by the clients of the PML, their close associates or third parties acting on their behalf or on behalf of affiliated legal entities. The PML may invest

the illicit proceeds on behalf of these clients in real estate, luxury goods, and businesses abroad (or, in some cases, in countries where the funds originated from). The funds can also be spent on goods deliveries to a country where the funds originated or to a third country.

Given the variety of domain in which money laundering can be pursed, we decided to focus on money laundering in capital markets. So considering an outlier [13] defined as *"an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data"*, we recognize an anomalous transaction as a fraudulent one. The challenge in applying fraud detection to anti-money laundering is that – whilst fraud mostly occur at transactional level – money laundering often involves multiple transactions across multiple accounts, hence multiple correlations have to be taken into account.

2.2 State of the Art

This section focuses on reviewing existing academic papers and research. Anomaly detection is a widely researched topic, covering areas such as intrusion detection systems or biomedical systems. In particular, we focus on money laundering detection but we also analyse the banking and credit card fraud detection context which share many common aspects.

A general overview of anomaly detection techniques has been presented in [3], which thoroughly explores the subject, offering a detailed description and categorisation of several of the algorithms with their respective applications. Moreover, in literature, there are a good number of paper reviews related to the fraud detection sector. Amongst these, we find [14], [15], and [16], in which the authors review a wide set of papers focused on supervised techniques applied to cases of fraud detection (credit card fraud, insurance fraud and money laundering) and anomaly detection in general.

Unsupervised learning is mainly used to detect unusual correlations and is applied where it is expensive to obtain labels (i.e., it requires reviewing multiple data points). The main principle adopted in the use cases related to money laundering detection is to quantify how a transaction (or group of transactions) deviates from the norm. In [17], the authors have directed their work along this way, proposing a formulation for outliers, according to the distance of a point from its neighbours. In [18], the authors proved that a Replicator Neural Network can detect anomalies in very diverse datasets and in some cases it overcame issues commonly affecting Neural Networks such as training with a small dataset. Another work concerning fraud detection is BankSealer [6], where the authors work in a semi-supervised setting, extracting a

local, global, and temporal profile for each user to capture their behaviours. Then they use various techniques in combination, such as HBOS [19] and (DBSCAN) [20], to rank new transactions. Concerning money laundering, the authors of [7] and [8] propose in the use case of detecting anomalies in investment funds, an approach based on clustering profiles into categories and feeding a Backpropagation Neural Network with the transformed data to output an anomaly score for each transaction. The above approach is extremely specific to that particular problem and dataset: the entire learning process is carried out on just two high-level features derived from the raw data, which appears to offer a limited perspective on the complexity of users' behaviours. In [21], the authors address the problem of money laundering in Brazilian exports using a type of Replicator Neural Network called Auto Encoder, which is considered a state of the art machine learning algorithm for anomaly detection. The disadvantage of unsupervised models is that in practice an analyst will still have to verify whether all the predictions were correct, and an unsupervised model will not be able to fully leverage the output of the reviews as part of subsequent runs. Also, unsupervised techniques tend to generate a large number of false positives due to unusual data correlations that are perfectly acceptable [22]. This is an issue for institutions, as false positives being reviewed translate into the direct cost for the organisations, which lead to potential money laundering cases not being reviewed promptly or being missed.

Supervised techniques are reviewed by [14], [15], and [16]; the most commonly used are Linear Regression, Bayesian Networks, Decision Trees, Neural Networks. Additionally, Random Forest and Support Vector Machines have started to gain more attention. In [23], the authors demonstrate that in a real credit card fraud scenario a Random Forest outperforms Support Vector Machines and Linear Regression across all metrics used for the comparison. Specifically for money laundering, one of the first studies applied to automated systems uses a rule-based method (Decision Trees) which is employed in the FAIS system [24]. This system allows the analyst to follow evidence sequentially left by linked transactions. An important component is a score relative to the anomaly degree for a transaction. Simple Bayesian networks are used to update and combine evidence that a transaction or activity is illicit. Supervised learning can utilise manually reviewed transactions (i.e., labelled data) and generally outperforms unsupervised learning in anomaly detection and classification tasks, as long as enough labelled data is available [25]. However, a large amount of labelled data is required to ensure adequate performance is achieved; additionally, they are not as effective at detecting new anomalous patterns (resulting in false negatives) when compared to unsupervised learning. This is where active learning plays an important role in bridging unsupervised and supervised anomaly detection.

Amaretto implements an active learning system combining both supervised and unsupervised learning, leveraging their strengths and mitigating their weaknesses. Active learning is a process whereby a model queries a subject matter expert for the label of a transaction or group of transactions (suspicious or genuine). Within active learning a dedicated model is used to select which transactions the subject matter expert should investigate to minimise manual data reviews and at the same time ensure the output of the overall anomaly detection system is improved. This approach has already been successfully deployed in anomaly detection as described in [26]. In their paper, the authors propose an ensemble of unsupervised methods including a Density-based model, a Matrix Decomposition-based model and a Replicator Neural Network. Combining the anomaly scores computed by the three models, their system ranks the instances based on the most anomalous and consequently present them to the subject matter expert for review; subsequently the feedback collected is used to train a Random Forest. Further to this research, in [27] the authors pointed out the importance of selecting data from a different type of anomalies to enhance active learning frameworks (i.e., selecting different classes of anomalies).

Amaretto explicitly focuses on reducing the cost for a bank, represented by the daily budget allocated on transaction monitoring and by the cost of not detecting illicit activities, and on optimizing the selection strategy to spot new anomalous patterns and to improve the detection rate of the system.

Another approach commonly applied in fraud and money laundering detection is the analysis of graphs. [28] is a detailed and in-depth review of research regarding graph-based anomaly detection methods in fraud detection, intrusion detection, telecommunication networks and opinion networks. In [29], the authors also provide a detailed analysis of graphs and their application to anomaly detection. Finally, the authors of [30] provide an example of usage for a graph-based model applied to anomaly detection using the KDDCup99 dataset for evaluation. As graph analytics was out of scope for Amaretto, further investigation was not performed.

Our work has also taken advantage of the progress and discoveries made within our research group, regarding the use of an ensemble of unsupervised models for anti-money laundering tasks [31], the use of supervised models [32], [33] and the implementation of active learning systems [34] in banking fraud detection.

2.3 Goals and Challenges

2.3.1 Money Laundering Detection: Goals

The review of literature on the topic of money laundering detection brought to light the lack of a system that can decisively tackle the problem of money laundering. Therefore, we define the following goals for our research work:

- develop an anti-money laundering detection system, that can model user's behaviours over a temporal frame and so not considering only the characteristics of individual transactions;
- prove that the system is reliable in a real-world scenario, detecting specific money laundering related patterns, recognized by the Financial Action Task Force on Money Laundering (FATF) [35], an inter-governmental body that promotes effective implementation of legal, regulatory and operational measures for combating money laundering, and demonstrating that the computational time is suitable for individual profiles' modelling;
- compare the system performances with other state-of-the-art anomaly detection solutions and verify that our system outperforms the existing solutions.

2.3.2 Money Laundering Detection: Challenges

The realization of an anti-money laundering system able to fulfil the stated characteristics has to address challenges specific to the financial domain, which are common to all its different subdomains:

- **highly imbalanced datasets**: concerning the enormous amount of legitimate transactions produced daily, the percentage of illicit ones is extremely limited, which leads to datasets with a fraud percentage which typically varies between 0.005% and 1.0%. With such a small number of outliers, anomaly detection systems are likely to obtain terrible performances, i.e., of samples that are classified as outliers when they are not. Incorrect classifications can cause considerable damage as both in terms of workload and in terms of unacknowledged illicit resources that can lead to sanctions by authorities. Therefore, the ultimate challenge for a financial money laundering detection system is to be able to indicate with the highest accuracy the transactions considered anomalous, reducing the possible costs for the banking institution. In case of systems that produce anomaly scores (i.e., sorting the analyzed data points from

the most to the least anomalous), this means to be able to place a consistent part of the illicit transactions in the very top positions of the ranking, to ensure a good detection rate while keeping the amount of work required to the analysts as low as possible;

- **the volume of data**: in an ideal world, with an unlimited amount of resources, an analyst would look at every transaction and then decide if the transaction or group of transactions present a risk of money laundering. Considering the large volume of transactions executed daily in global markets, this approach is not feasible because financial institutions, regulators and enforcers have a limited amount of subject matter experts to deal with such a significant demand. To solve this problem, organisations employ a risk-based approach by adopting automated systems to flag and allocate transactions for review. The objective is to maximise the time spent investigating suspicious activities;

- **lack/scarcity of labels**: a burdensome challenge in dealing with money laundering task is finding a valid and trustworthy dataset of transactions. Unfortunately, banks do not share data or information to not damage their reputations and some of them do not even have a record of laundering reports. This, apart from strongly limiting the chance of using supervised learning approaches, also impacts the possibility of evaluating the effectiveness of the developed solutions, because there is no ground truth against which the output of the anomaly scoring or classification can be compared. Therefore, the solution usually adopted when evaluating financial money laundering detection tools is to either resort to an injection of synthetic frauds in real datasets or to produce entirely synthetic datasets shaped to resemble the characteristics of the real ones. This last solution is often the adopted one due to the restrictive privacy policies that concern data owned by financial institutions, which make it hard to get any real-world dataset to work on when making research in this domain;

- **dynamic frauds patterns**: another major challenge is that even if the financial institution has historical labelled data, fraudsters evolve their techniques over time. Given this dynamic behaviour of fraudulent patterns, any solution developed on old data may become obsolete and its performances decrease over time. Therefore, one of our goals is to provide a system able to create a labelled dataset over time, to provide a ground truth knowledge base that can be leveraged to develop supervised models or further knowledge for the domain experts.

Chapter 3

Dataset Analysis

In the anti-money laundering domain, one of the major limitations is the difficulty to obtain a real dataset from financial institutions, due to privacy concerns; besides, it is a much more complicated challenge to be able to use a dataset labelled by human analysts. However, we have the great opportunity to work with *NapierAI*, who is an anti-money laundering company that work within the capital market and help their clients comply with AML regulations, that provided us with a synthetically forged dataset consistent with an international capital market. In this chapter, we describe and analyze this dataset.

3.1 Dataset Description

A synthetic dataset was generated to simulate transaction profiles consistent with customers of the international capital market. The data was generated by our industry partner using a custom-built data generator, that combined more than 10,000 parameters. The dataset consists of 29,704,090 transactions executed by 400 end-customers, covering 60 days divided in 12 weeks (a week is composed of 5 days, Saturdays and Sundays are not included because during the weekend markets are closed) that are buying or selling specific securities in a specific market. Key fields contained in the data include the amount of the transaction, the product class (e.g., Equity, Fixed Income, etc.), product type (e.g., cash equity, future equity, bond, etc.), time field, currency, market. Within the data, it is not possible to identify any specific statistical distributions in any key field. Table 3.1 shows the summary statistics for the synthetic data.

To replicate real-world scenarios, we set the number of anomalies to less than 1% of the data. As part of this dataset, 5 classes of anomalies have been generated based

on examples of suspicious patterns suggested in [35] of The Financial Action Task Force (FATF), an inter-governmental body that promotes effective implementation of legal, regulatory and operational measures for combating money laundering.

Table 3.1: Dataset characteristics

Dataset size	Nominal Transactions	Anomalous Transactions	Ratio	Originator
29,704,090	29,622,822	81,268	0.274%	400

3.1.1 Features Description

In this section, we are going to explain and to show the features that compose our dataset. Since the dataset was forged synthetically by NapierAI, any feature cleaning or feature engineering was required.

The following are the features in our dataset:

- **Transaction ID**: is a unique value that identifies the transaction in the dataset. It is included in the dataset to guarantee compatibility with a real dataset but for our research, it does not provide any useful information, so it is never used;
- **Originator**: is the name of the users who performed the transaction. In our dataset, there are 400 distinct UserIDs. This means that there are 74,260.225 transactions per user, but the distribution is very skewed. Figure 3.1 shows how many transactions have been executed by each user. Circa 90% of the users made at least 50,000 transactions while 10% of the users performed circa 400,000. It means that 10% of the clients executed almost 50% the transactions;
- **Originator_ID**: is the identifier of the user. We do not need this feature because the *Originator* is sufficient to identify a customer;
- **InputOutput**: indicates if an asset is bought or sold with the transactions. Figure 3.2 shows that transactions are almost evenly distributed between the two categories;
- **EntryDate**: is the date and time when the transaction is executed. Figure 3.3 shows the distribution of the transactions over the days and over the hour. Since the dataset is synthetic, any seasonal trend is characterizing the data. It is possible to see that the transactions are evenly distributed between the

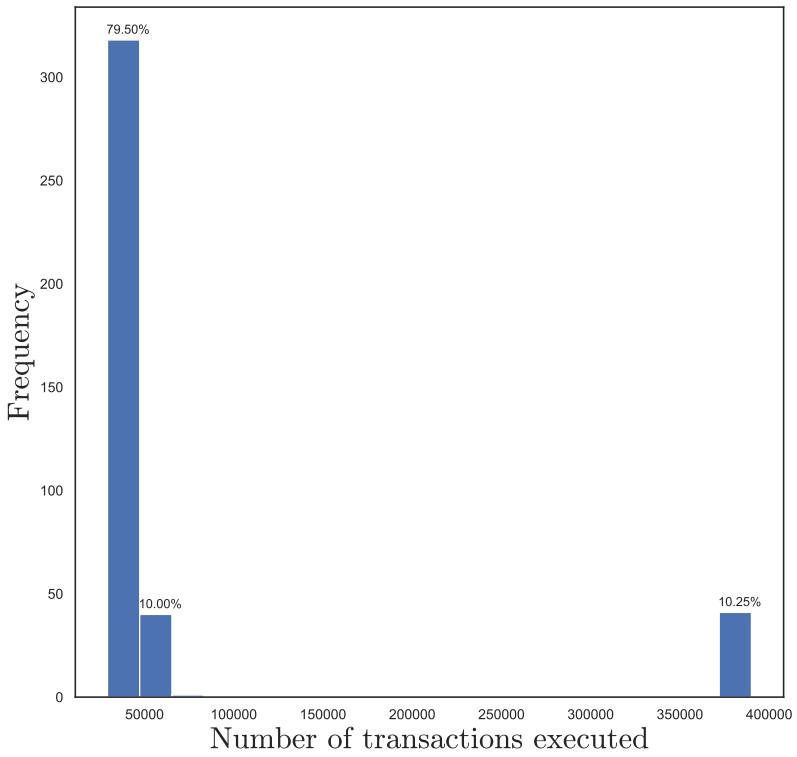


Figure 3.1: Histogram of number of transactions per user

12 weeks and that most of the operations are executed during market opening hours while only a small percentage of the transactions is done during the early hours of the morning and at the end of the day. Also, any particular trend can be observed looking at the distribution of the transactions over the date (Figure 3.3(a));

- **Market**: is the market where the transaction is executed. NapierAI forged the dataset using insights provided by one of their partners, so the market included in the dataset are not randomly chosen. Figure 3.4 shows how the transactions are distributed between the different markets.

- **Product ISIN**: contains information characterizing financial instruments but

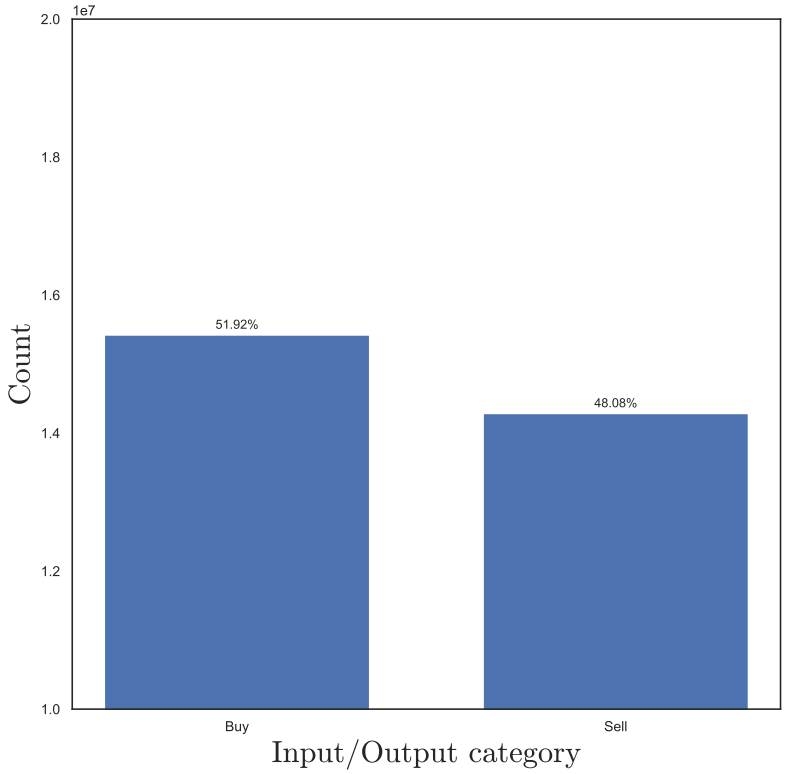


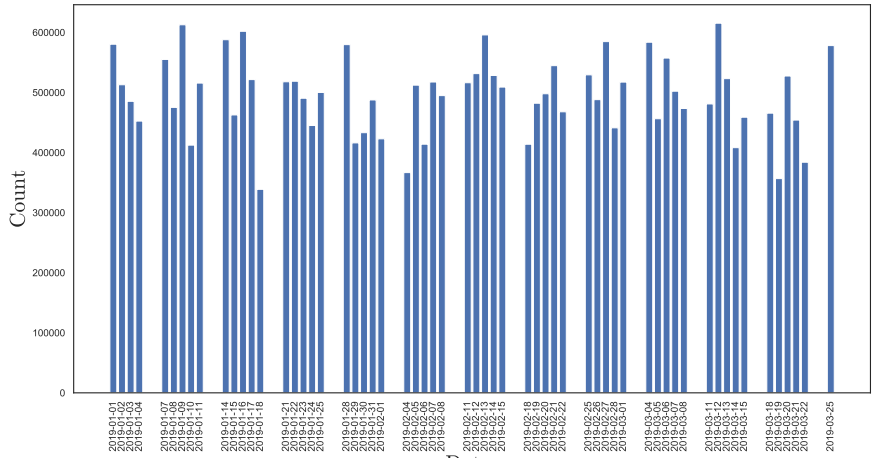
Figure 3.2: Histogram of number of transactions bought or sold

rather serves to uniformly identify a security for trading and settlement purposes. Since our dataset is synthetic, this feature is a random alphanumeric string that don't bring any insight for our challenge; so for this reason, it is not used;

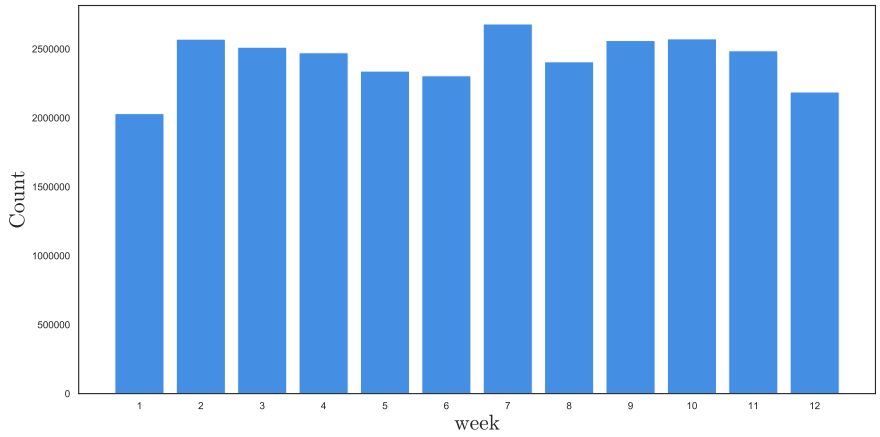
- **Product Type**: is the category of the product that is traded in the transaction. There are 17 different products, representing the main product traded in the capital market. Figure 3.5 shows how these categories are distributed;
- **Product Class**: is the procedure by the product type is traded in the transaction. Figure 3.5 shows how these categories are distributed;
- **Normalized Amount**: represents the amount of the transaction in USD. It is normalized since the transactions are performed in different markets and so in

Figure 3.3: Transactions distribution over time

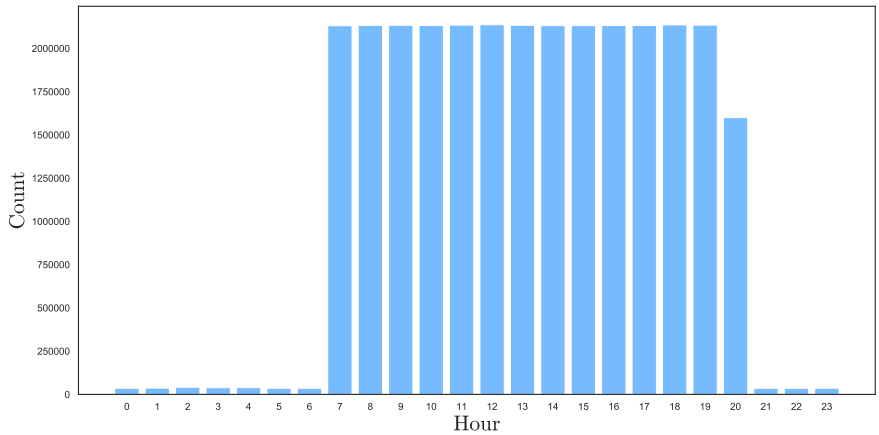
(a) Transactions distribution over date



(b) Transactions distribution over week



(c) Transactions distribution over hour



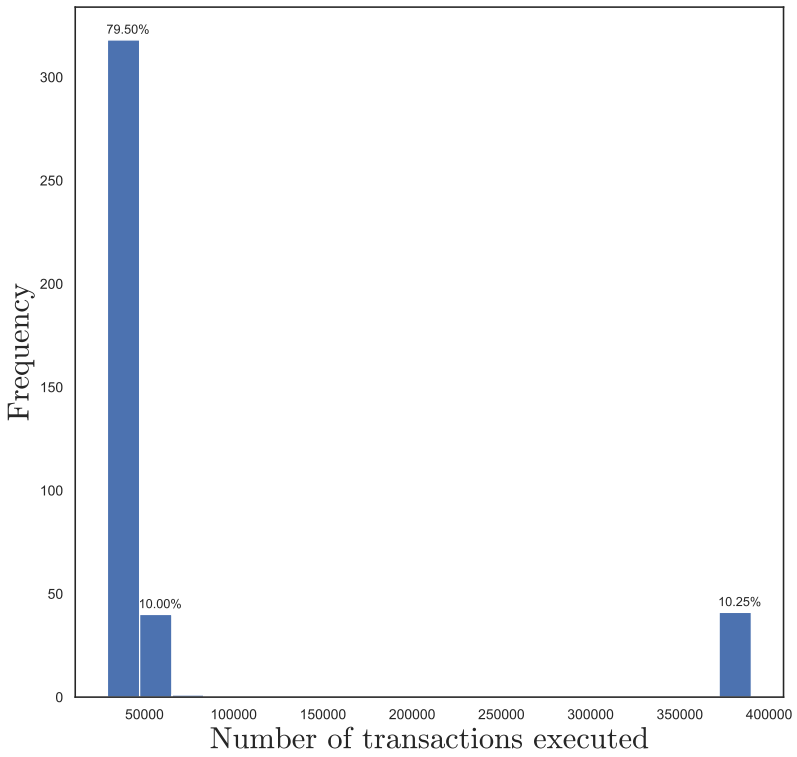


Figure 3.4: Number of transactions in the different markets

different currencies. Figure 3.6(a) and Figure 3.6(c) shows two histograms for the normalized amount of genuine transactions and anomalous transactions. It is possible to see a high spike indicating that 98.43% of the transactions have an amount that is less than 1M USD and focusing on these transactions 40% have an amount less than 10K USD. While Figure 3.6(b) and Figure 3.6(d) show the trend of the amount for the anomalous transactions. From the figures, it is evident that the anomalous transactions follow the same trend and this reinforces the thesis that anomalous transactions are very well hidden with genuine ones.

- **Currency**: is the currency in which the transactions are made. There are two possible values: RUB and USD;

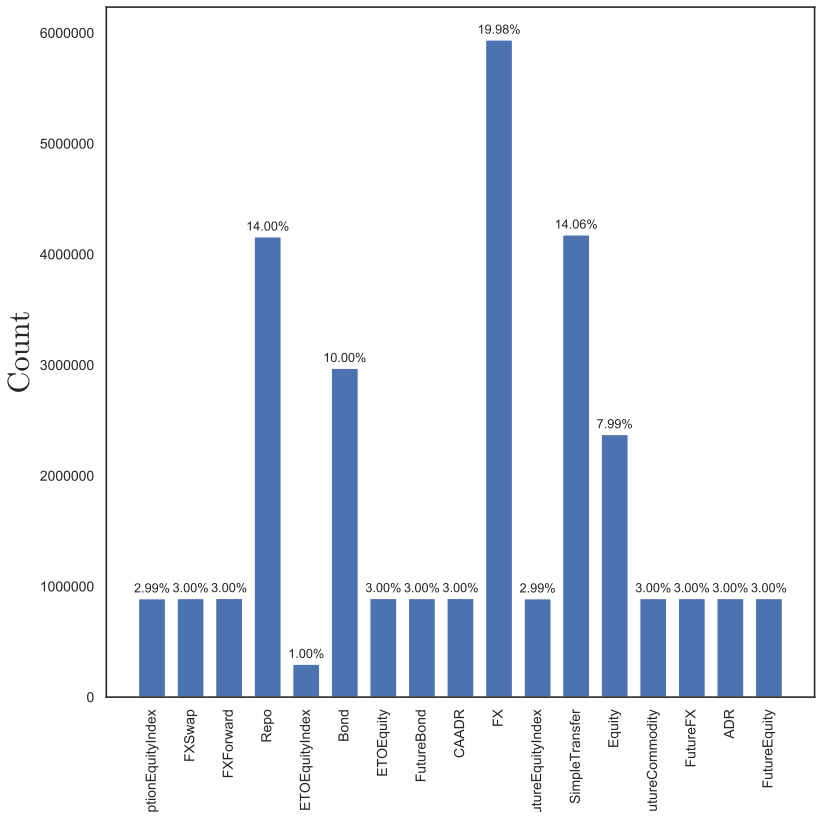
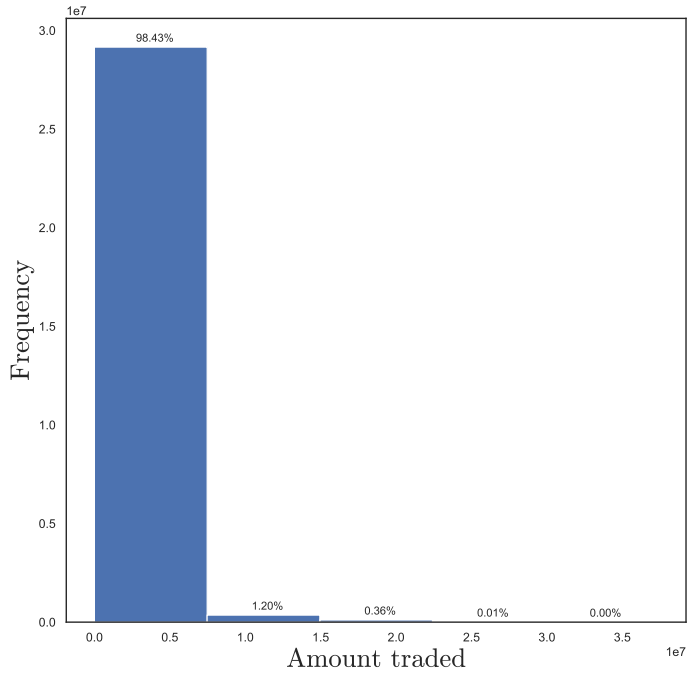
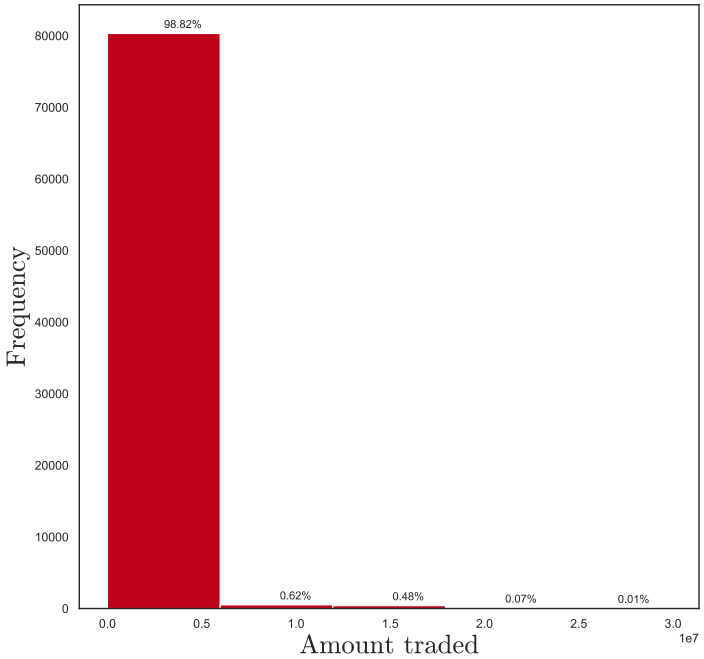


Figure 3.5: Distribution of product types

- **Anomaly**: indicates if a transaction is anomalous or not. It is only used in for test purposes. It represents an important feature so it is described in detail in Section 3.1.2. Figure 3.7 shows the trends of anomalous transactions: it is possible to see that fraudsters operate following genuine trends, camouflaging illegal transactions. The only difference can be spotted in Figure 3.7(c) that shows anomalous transactions also in early morning hours.

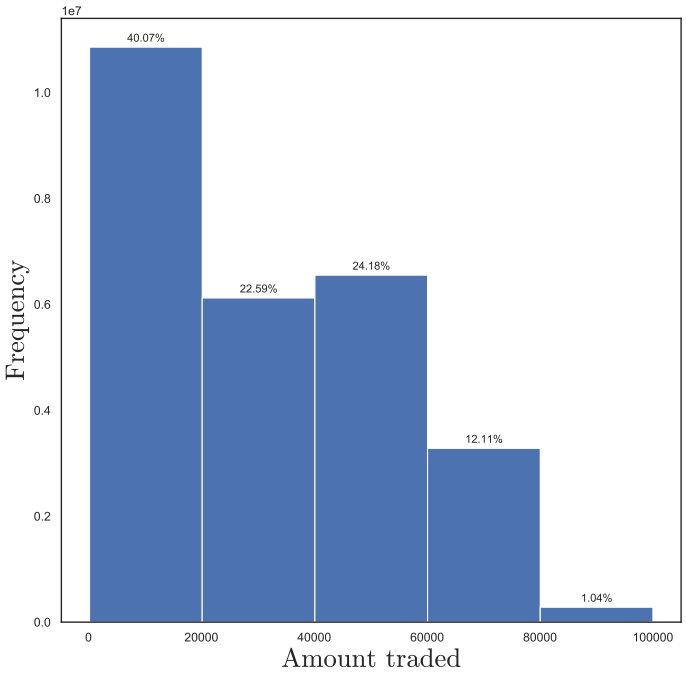


(a) Amount genuine distribution

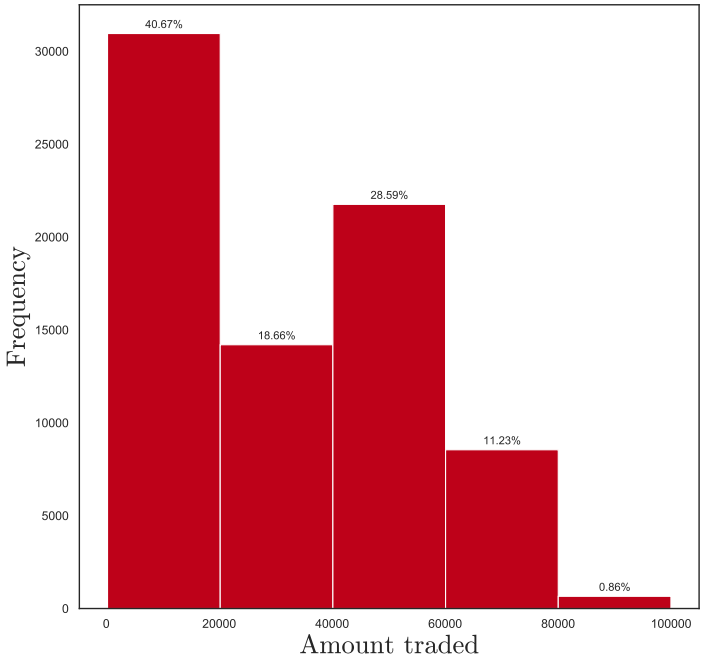


(b) Amount anomalous distribution

Figure 3.6: Amount transactions histograms

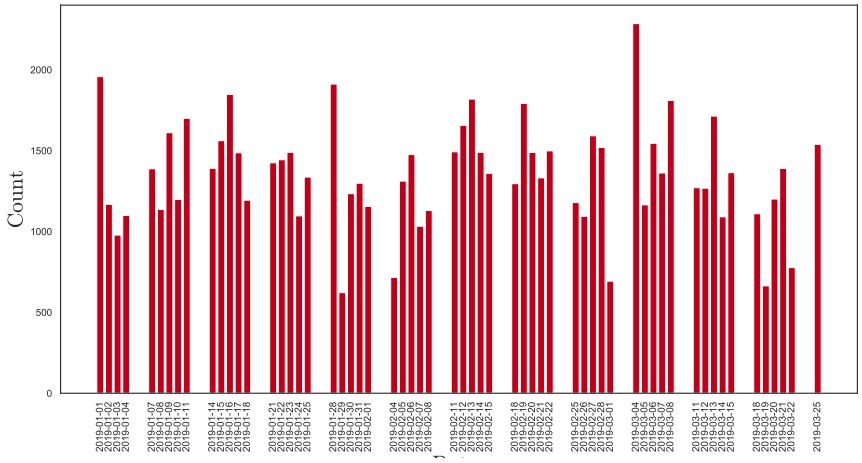


(c) Amount genuine distribution (<100k)

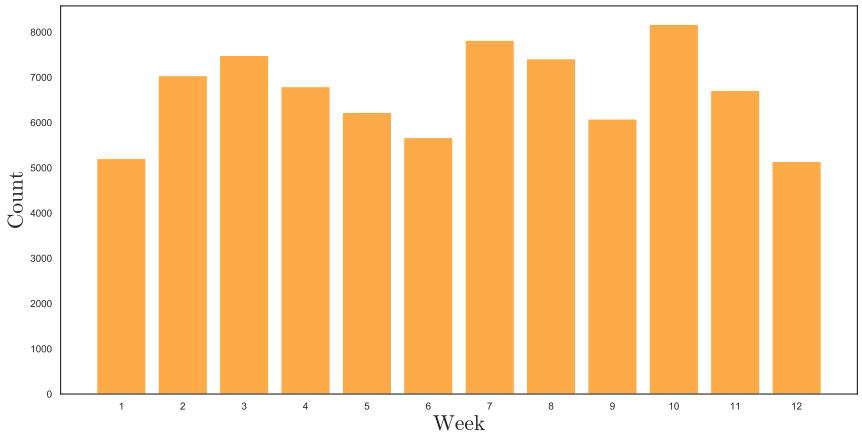


(d) Amount anomalous distribution (<100k)

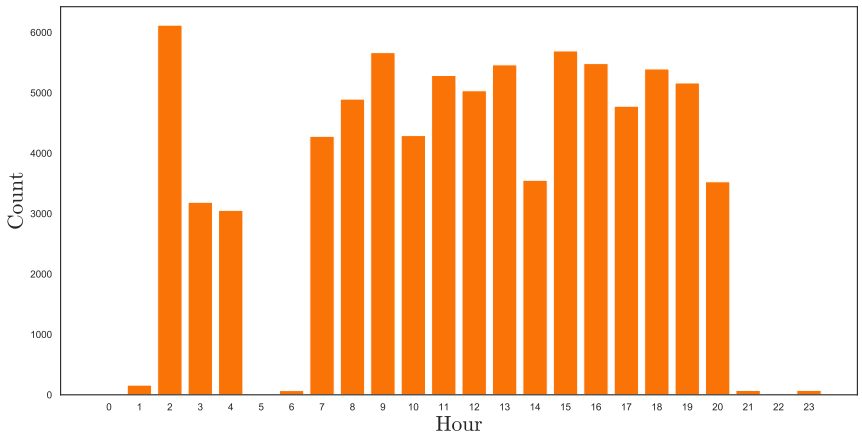
Figure 3.6: Amount transactions histograms, cont.



(a) Anomalous transactions distribution over date



(b) Anomalous transactions distribution over week



(c) Anomalous transactions distribution over hour

Figure 3.7: Anomalous transactions distribution over time

3.1.2 Money Laundering Patterns Analyzed

A total of 81,269 anomalous transactions have been generated in the dataset, this equates to circa 0.274% of the total population. The criteria used to generate the anomalies is based on mirroring realistic scenarios highlighting unusual behaviours in an account or customer activity. 5 classes of anomalies have been generated based on examples of suspicious patterns suggested in [35] of The Financial Action Task Force (FATF), an inter-governmental body that promotes effective implementation of legal, regulatory and operational measures for combating money laundering. Below, we describe the classes of anomalies injected in the dataset:

1. **Small but highly frequent transactions generated within a short timeframe**: a transaction pattern indicating a value of transactions just below any applicable reporting threshold;

2. **Transactions with rounded normalised amounts bought or sold within an account**: it is unusual for transactions in capital markets to have rounded amounts (unless they occur in markets where foreign exchange conversion causes rounding errors)

3. **Security bought or sold at an unusual time**: it is unusual for a customer trading a specific security, to trade outside of a specific timeframe (for example, outside of the opening and closing times of a stock exchange);

4. **Large asset withdrawal**: a sudden spike in transaction amount withdrawn from an account or transferred out, which deviates from previous transactional activity absent any commercial rationale or related corporate action event;

5. **An unusually large amount of collateral transferred in and out of an account within a short period**: this behaviour is unusual as a client would not be able to invest by simply trading collateral, or at least such a strategy would be unusual;

Figure 3.8 indicates the number of anomalous transactions belonging to each category of money laundering pattern. Over each bar is possible to see the percentage of each category related to the total number of anomalies. The figure shows that the majority of the anomalies (74.71%) are distributed between category 4 and 5, while less than half of the anomalies are divided among the other categories.

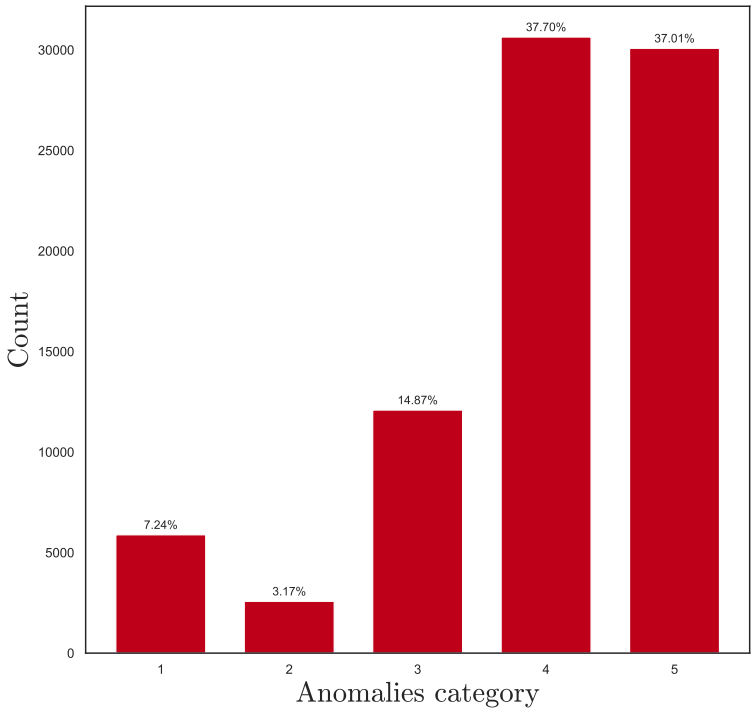
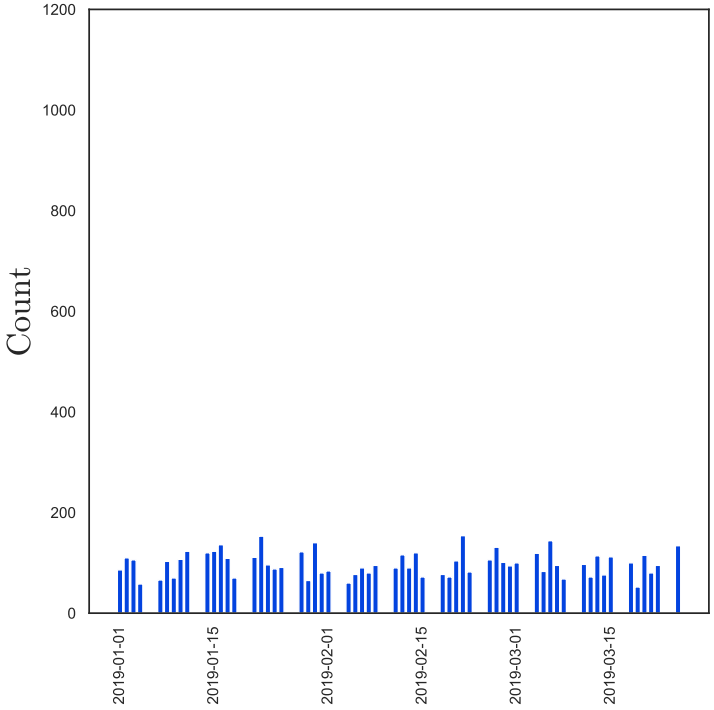
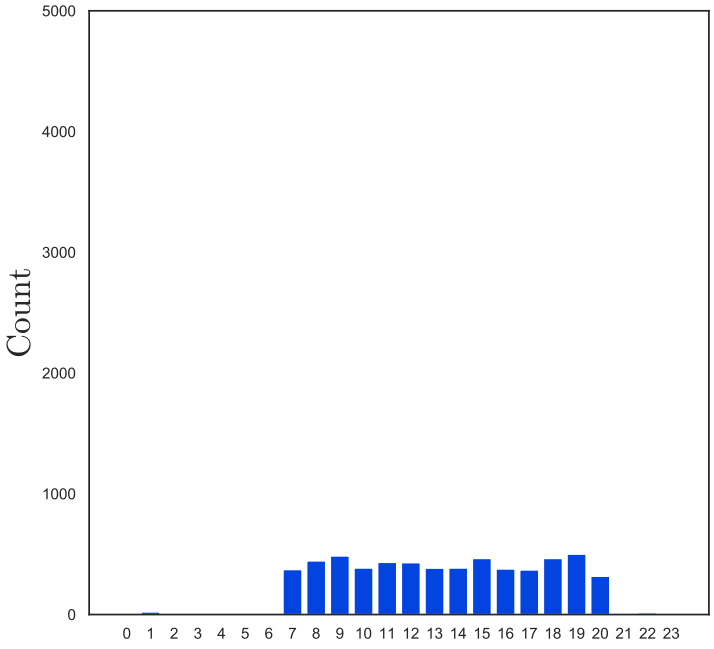


Figure 3.8: Anomalies count

Figure 3.9 shows the distribution of each category over the time (distribution over the date on the left column, distribution over hours on the right column). It is possible to see, on the left column, how the anomalies follow the same distribution of the genuine transactions shown in Figure 3.6(a); every category is distributed almost uniformly over the days without any particular peak. Furthermore, the same considerations can be drawn on the hourly distribution. Except for category 3 which by definition occurs only at unusual times: in fact from Figure 3.9(f), it is possible to notice how this type of anomaly occurs only during the night.

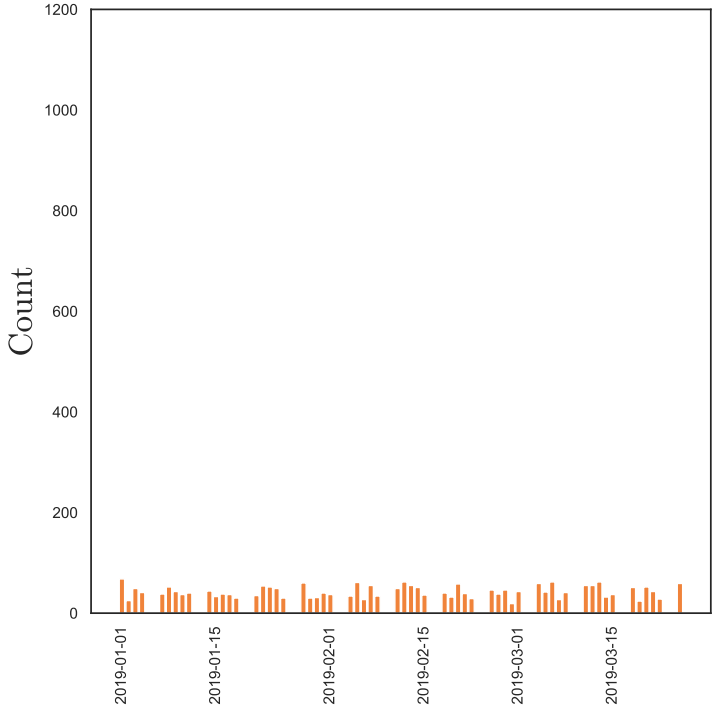


(a) Anomalies 1 distribution over the date

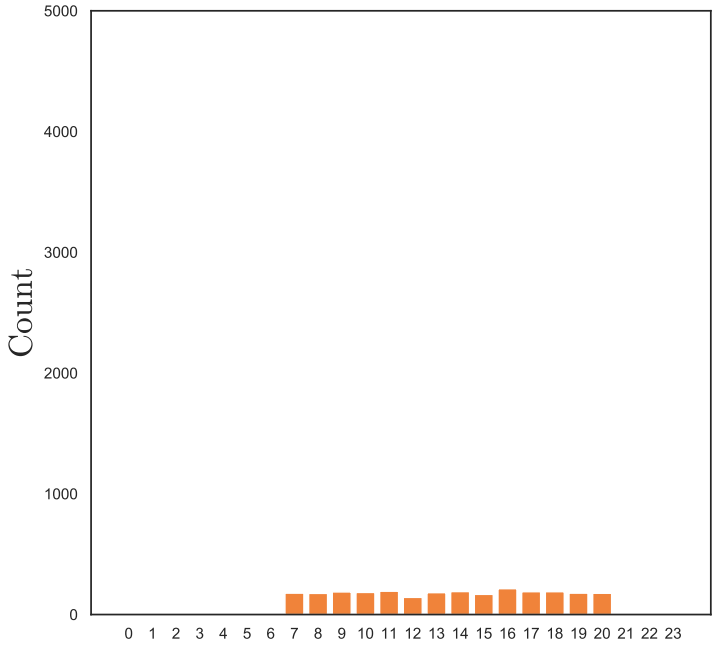


(b) Anomalies 1 distribution over hours

Figure 3.9: Money laundering pattern distribution over time

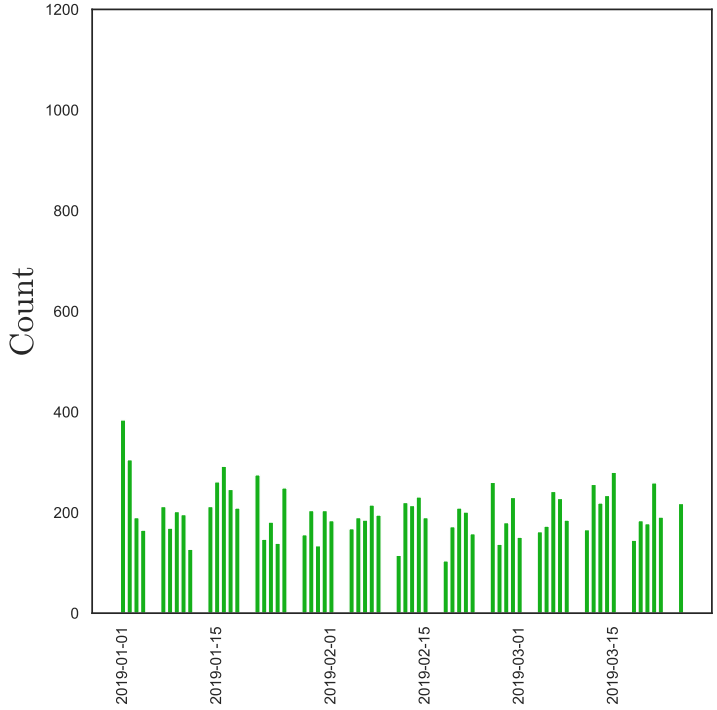


(c) Anomalies 2 distribution over the date

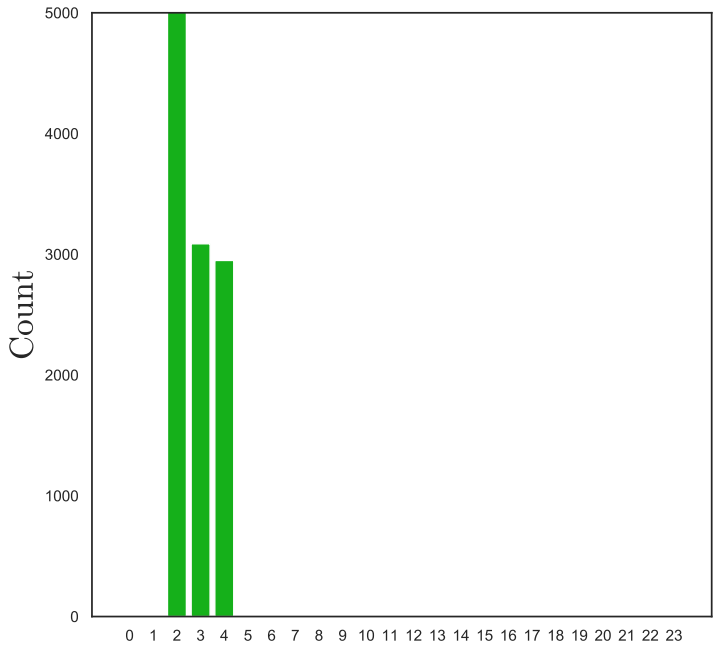


(d) Anomalies 2 distribution over hours

Figure 3.9: Money laundering pattern distribution over time, cont. 1

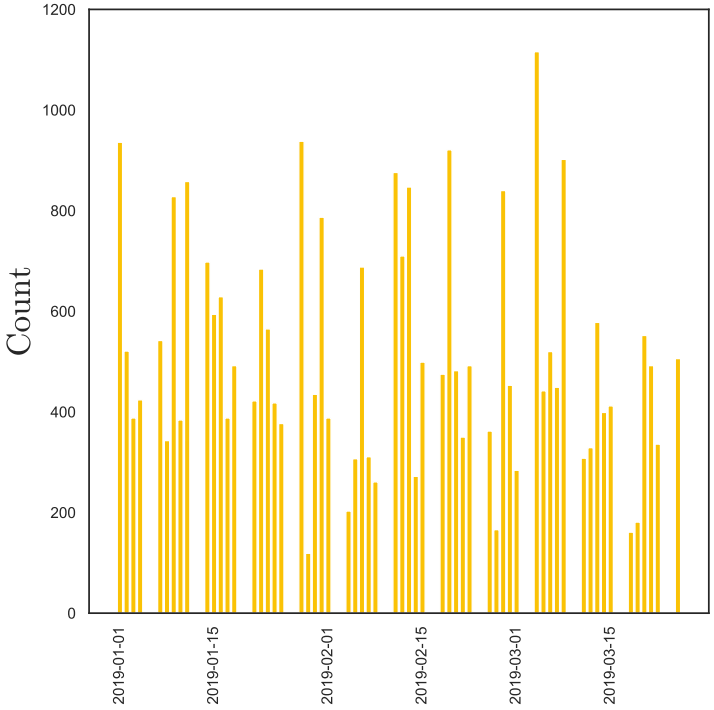


(e) Anomalies 3 distribution over the date

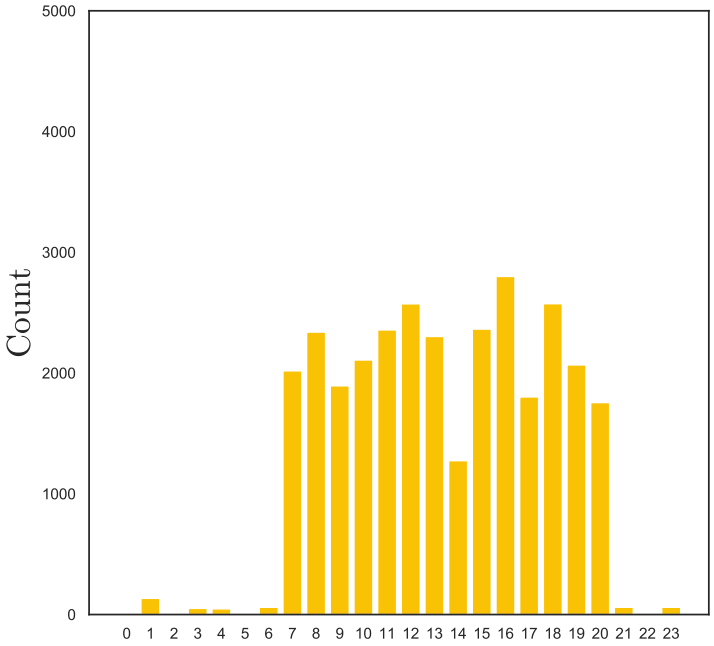


(f) Anomalies 3 distribution over hours

Figure 3.9: Money laundering pattern distribution over time, cont. 2

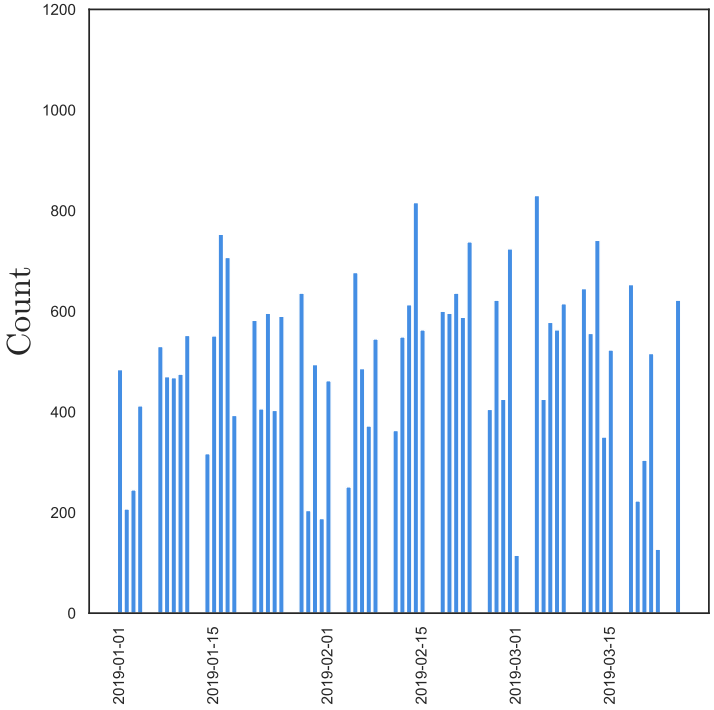


(g) Anomalies 4 distribution over the date

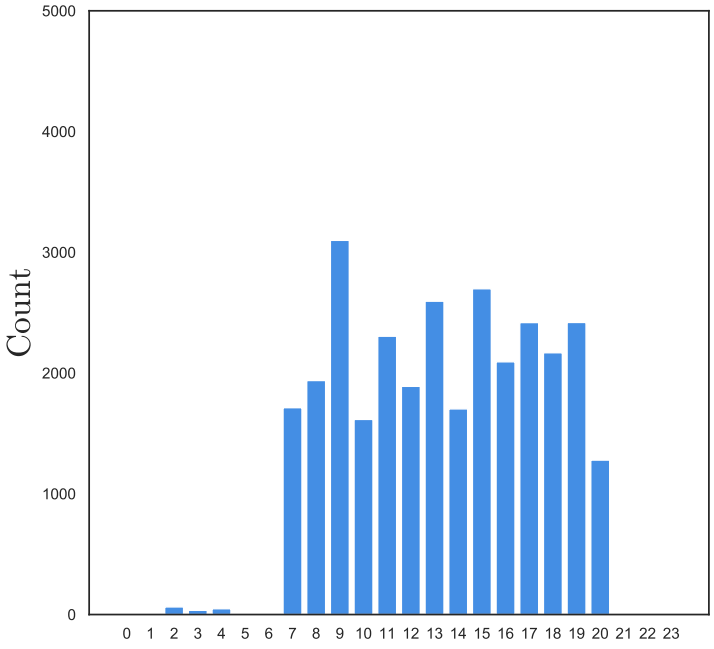


(h) Anomalies 4 distribution over hours

Figure 3.9: Money laundering pattern distribution over time, cont. 3



(i) Anomalies 5 distribution over the date



(j) Anomalies 5 distribution over hours

Figure 3.9: Money laundering pattern distribution over time, cont. 4

Chapter 4

Approach

In this chapter, we describe the design of Amaretto. We present all the single procedures that the system performs, then we show how they are combined to create an active learning framework that merges unsupervised and supervised techniques together to address all the challenges introduced in Section 2.3. In particular, the active learning framework will enable Amaretto to be deployed in situations where there is no past labelled data and where fraudulent patterns are rapidly evolving, which is a typical situation for money laundering detection.

4.1 Approach Overview

In this section, we introduce our active learning system for transaction monitoring, Amaretto.

To enable the detection of money laundering through supervised models, there is a requirement for sufficient labelled data. Unfortunately, the only way to have reliable labels is to have all transactions manually reviewed by an analyst, which is not feasible. For this reason, we opted for a hybrid solution, using active learning as described in [36]. This consists of using both unsupervised and supervised techniques, to overcome their respective limitations, combined in an analyst in-the-loop framework. The main goal of the system is to reduce the cost of transactions monitoring for a bank institution, supporting the analyst in his daily job routine. The system will focus on analysing the most suspicious activities using the `anomaly_score` assigned by the system, reducing the number of reviews that have to be done by the analyst. Within the active learning framework, the supervised models are trained on the analysts' feedback which constitutes the labelled dataset. The contribution of this procedure is to constantly increase the performances of the supervised model, integrating also new

anomalous patterns discovered by the unsupervised model.

The first step in the AMARETTO workflow is to aggregate the raw transactional data across a specific timeframe to produce features representing high-level vectors that capture the behavioural profile of a customer. The models employed in AMARETTO are trained with these high-level vectors generated from historical data. This is done to detect anomalies in customers behaviours inside an extended period, rather than just in the characteristics of the individual transactions they perform. After the training phase, AMARETTO computes an `anomaly_score` for each new vector using both unsupervised and supervised modules, if enough data is available to train the latter. Then a selection of vectors is done using the `anomaly_score`. The number of transactions that will be sent for review to the analyst each day (`daily_k`) is a parameter of our system, based on the resources that a financial institution can allocate on this task. At this point, the potential unusual behaviours are manually inspected by an analyst who assesses whether they are truly anomalous or not, hence saving this information as labels in the dataset building a knowledge base which grows every day by `daily_k` records. The reviewed labels contribute to a historical set of labelled data that is the input for the supervised component of the system. The supervised component is then used alongside the unsupervised model to continuously sample and select the data to be reviewed by the analyst.

The key component of AMARETTO is the selection strategy. The system selects the vectors to be shown to the analyst, and that will form the training set for the supervised model, considering the `anomaly_score` returned by the two models. This process is divided into three stages that can be configured to use different strategies. The first stage involves only the `anomaly_score` computed by the unsupervised module; this allows the system to rank each vector. Using the first strategy, the system samples the topmost anomalous vectors; on the contrary, the second strategy clusters the vectors and then samples the most anomalous vectors from each cluster to ensure the system learns different types of anomalies. In the second stage, the vectors are queried using only the `anomaly_score` of the supervised model, following these criteria: high scores, indicating a high probability of being an anomalous set of transactions and low scores, indicating high probability of being a normal set of transactions. In the third and last stage, data points considered *uncertain* are selected. To do so, the vectors can be ranked using the uncertainty of the score generated by the supervised model, this can be done by either sampling a set of transactions with an `anomaly_score` close to 0.5, or taking into account the difference of the `anomaly_score` computed by the unsupervised and the supervised model (i.e., when the two models disagree in scoring vectors).

In Figure 4.1 is possible to observe a scheme of the general design of AMARETTO.

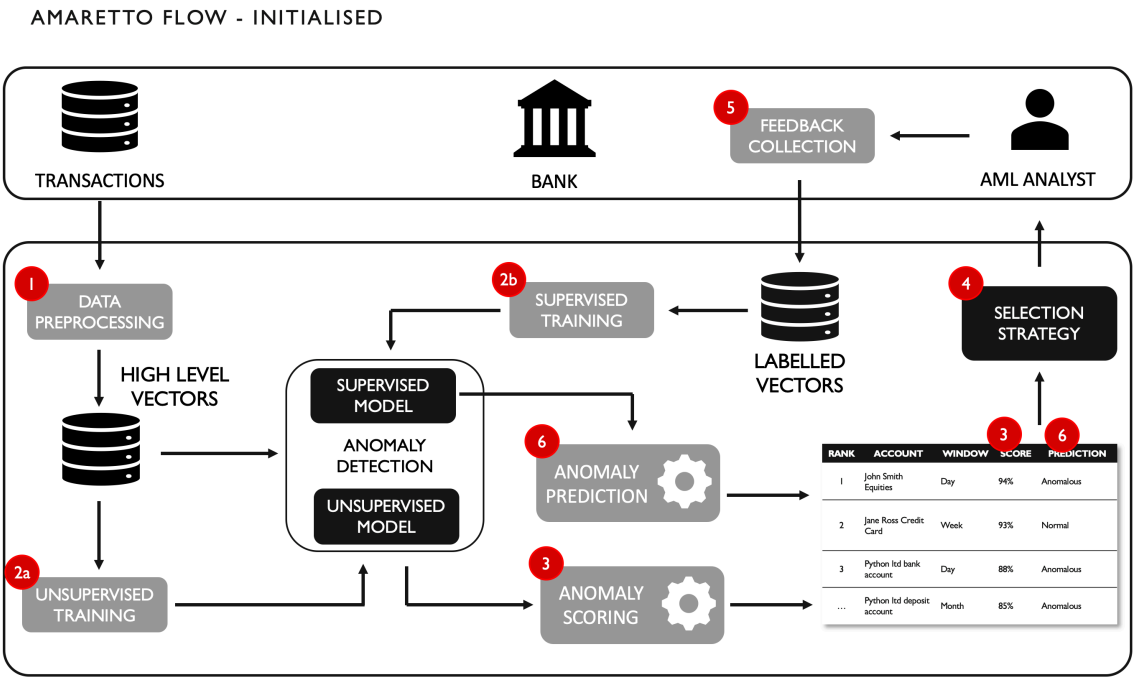


Figure 4.1: AMARETTO design

While in Algorithm 1 the pseudocode of the workflow is shown.

Summarizing, the use of the active learning pattern allows us to define a precise number of transactions to be labelled every day that solves the budget issue posed by the fully supervised approach. At the same time, it allows us to store and use the precious information received by the human analyst feedback, that otherwise, using a fully unsupervised approach, would be wasted. Moreover, we have a labelled training set that grows every day with new records; in this way, the supervised module that is trained with the labelled training set will always be up to date and capable of catching new fraudulent patterns. This would not be possible if the supervised module is only trained with an initial labelled training set and then used to predict, without the unsupervised module beside it that provides additional alerts of possibly new fraudulent patterns.

Algorithm 1: AMARETTO

Input: $\mathcal{L} = \emptyset, \mathcal{U}$, mod_{sup}, mod_{unsup}, $strat_{first}$, $strat_{second}$, $strat_{third}$, $\mathcal{K}$, $\mathcal{T}$
for $t = 0, \ldots, T$: **do**
  if $t = 0$ **then**
    Train mod_{unsup} on $\mathcal{U}^{t-1}$
    Compute the scores with mod_{unsup} for $S(x_i)$ where $x_i \in \mathcal{U}^t$
    Query $\mathcal{K}$ samples from $\mathcal{U}^t$ using the sampling strategy $strat_{first}$
    Collect analyst feedback. Call it $sample^t_{unsup}$
    Add the selected points to $\mathcal{L}^{t-1}$ i.e.
    $\mathcal{L}^t = \mathcal{L}^{t-1} \cup (x_i, y_i) \in sample^t_{unsup}$
  else if $t > 0$ **then**
    Train mod_{unsup} on $\mathcal{U}^{t-1}$
    Train mod_{sup} on $\mathcal{L}^{t-1}$
    Compute the scores with mod_{unsup} and mod_{sup} for $S(x_i)$ for where $x_i \in \mathcal{U}^t$
    Query $\frac{\mathcal{K}}{2}$ samples from $\mathcal{U}^t$ using the sampling strategy $strat_{first}$
    Collect analyst feedback. Call it $sample^t_{unsup}$
    $\mathcal{U}^t = \mathcal{U}_t \setminus sample^t_{unsup}$
    Select $\frac{\mathcal{K}}{2}$ samples from $\mathcal{U}$ using the sampling strategy $strat_{second}$ and $strat_{third}$
    Collect analyst feedback. Call it $sample^t_{sup}$
    Add the selected points to $\mathcal{L}^{t-1}$ i.e.
    $\mathcal{L}^t = \mathcal{L}^{t-1} \cup sample^t_{unsup} \cup sample^t_{sup}$

4.1.1 AMARETTO Workflow

In this section we will briefly describe the workflow involved in the daily routine of AMARETTO once it's fully operational.

1. **Data Processing Module**: the system retrieves new raw transactions (*low-level vectors*) from the bank to compute the customers' profiles than a behavioural modelisation (*high-level vectors*) is performed. The system computes aggregate features, capturing the customers' behaviours signature within a period, ready to be pipelined to the learning algorithms; (Section 4.2)

2. **Unsupervised Module**: high-level vectors are passed to the unsupervised model which outputs the `anomaly_score` for each one; (Section 4.3.1)

3. **Supervised Module**: the high-level vectors are fed into the supervised model which gives a *probability* or a *binary prediction* as output; (Section 4.3.2)

4. **Selection Strategies Module**: at this point, `daily_k` vectors detected by the system are presented to the human analyst, which will explore them and label them as *fraud* or *genuine*. The feedback is then stored into the local labelled database, that will be used in future to train the supervised model; (Section 4.4)

5. at the end of each day, the **feedbacks** are collected from the analyst and models are retrained:

   - the labelled database is given as input to the supervised model training algorithm;
   - the unsupervised model is trained (in an online fashion) with all the high-level vectors of the day, excluding the ones labelled as frauds by the analyst.

6. at the end of the month, with the aim to keep the tool up to date, an optional **hyper-parameters optimization** operation is performed. (Section 4.5)

4.2 Data Preprocessing

In this section, we present the steps performed by AMARETTO to transform the raw transactions data into the high-level vectors used for models training, also providing an insight on why this process is necessary to perform money laundering dection.

4.2.1 Purpose of the Aggregation Process

The starting point for performing money laundering detection is raw transactional data, i.e., a dataset of historical transactions performed by customers which can represent single person, an holding company or any entity able to perform financial movements in capital markets. Each raw transactions data sample is a vector representing a finanancial transaction, i.e., with at least a sender customer ID, the amount of money transferred and the timestamp of the transaction. In this domain, applying machine learning with the purpose of anomaly detection directly on raw data is either completely impossible, or destined to lead to utterly bad results, for a variety of reasons:

- **features' domain**: in general, most learning algorithms work with numeric data. However, almost all the raw features in transactions data are nominal: there are account IDs, countries, products codes and currencies identifiers, timestamps and many others. Therefore, preprocessing steps are necessary to extract usable information;
- **lack of perspective**: after mapping all the features into numeric ones, it is hard to learn anything useful from vectors corresponding to individual transactions for our purpose. The reason is that it is impossible to understand a customer's habits and, therefore, to detect variations in them, looking exclusively at how each individual transaction is shaped. To capture customers' behaviour, the system needs to compute statistics on number of transactions performed, product bought/sold or amount of money traded within a specific timeframe. To obtain these meaningful features, the system need to aggregate the information of individual transactions;
- **volume of data**: finally, even if the system would be able to learni directly from individual transactions, the volume of the data to handle would be huge, leading to practical problems in models' training, in term of both memory and time required. Also, too much data pointing to even slightly different directions in terms of customer behaviour information can result in losing the ability of capturing an modelling what is common, recurrent and standard for that customer or group of customers. In a domain where the amount of data produced worldwide daily is enormous, having one vector containing condensed, meaningful information about one day or few hours of transactions data instead of thousands of individual transaction vectors may be the only solution to apply machine learning on that data.

Considering the aforementioned reasons, a data preprocessing and aggregation is necessary for the succes of the system.

4.2.2 Data Preprocessing Steps

AMARETTO generates a set of high-level features that can be derived from the information contained in the *transactional data*. To derive such aggregated features, an *aggregation window* is chosen: this value represents the time over which transactions are aggregated (e.g., hours, days, etc.) and is used for computing each set of aggregated features. Aggregating transactions over a period of time is useful in the anti-money laundering use case since it can be used to capture correlations over time across multiple transactions.

1. **Data parsing**: the data is parsed to obtain raw transaction vectors, containing the original features. In this step, the raw data is converted into a format that can be easily read by the system. The raw features' handled by the system are described in detail in Section 3.1.1;

2. **High-level features derivation**: AMARETTO implements a set of high-level features that can be derived from the information contained in the *transactional data*. To derive such aggregated features, an *aggregation window* is chosen: this value indicates the width of the time window to consider for computing each set of aggregated features. For example, as shown in Figure 4.2, if a daily window is chosen, one aggregated features' set will be produced for each day, by aggregating all the transactions the customer performed in that day. This functionality has been introduced with a particular focus on money laundering detection since it is a very complex scenario in which studying anomalies and laundering processes can be detected only analysing patterns of transactions.

4.2.3 Modelling Strategies

AMARETTO offers two option of developing the anomaly detection models of customers to allow its usage in different conditions of data availability:

- **individual users' modelling**: if a single user profile contains enough information to allow them to be modelled individually, then this is the best to perform anomaly detection. What 'enough information' means depends on many factors: the specific subdomain of the data, how diversified the user's behaviour is and therefore for how long it needs to be observed to learn its regular patterns, how many high-level features are being used, and several others. It is, therefore, something that varies from application to application, and needs to be investigated in place;

- **global modelling**: a single, global model could also be trained using all the vectors from all the customers. However, this is usually not a good solution in any scenario in which behavioural patterns of the customers are diversified because one model would not be able to capture all the different behavioural patterns. So, unless necessary due to lack of data or to the opposite problem of a massive amount of data and limited computational resources available, this solution should not be adopted, or not as the only one.

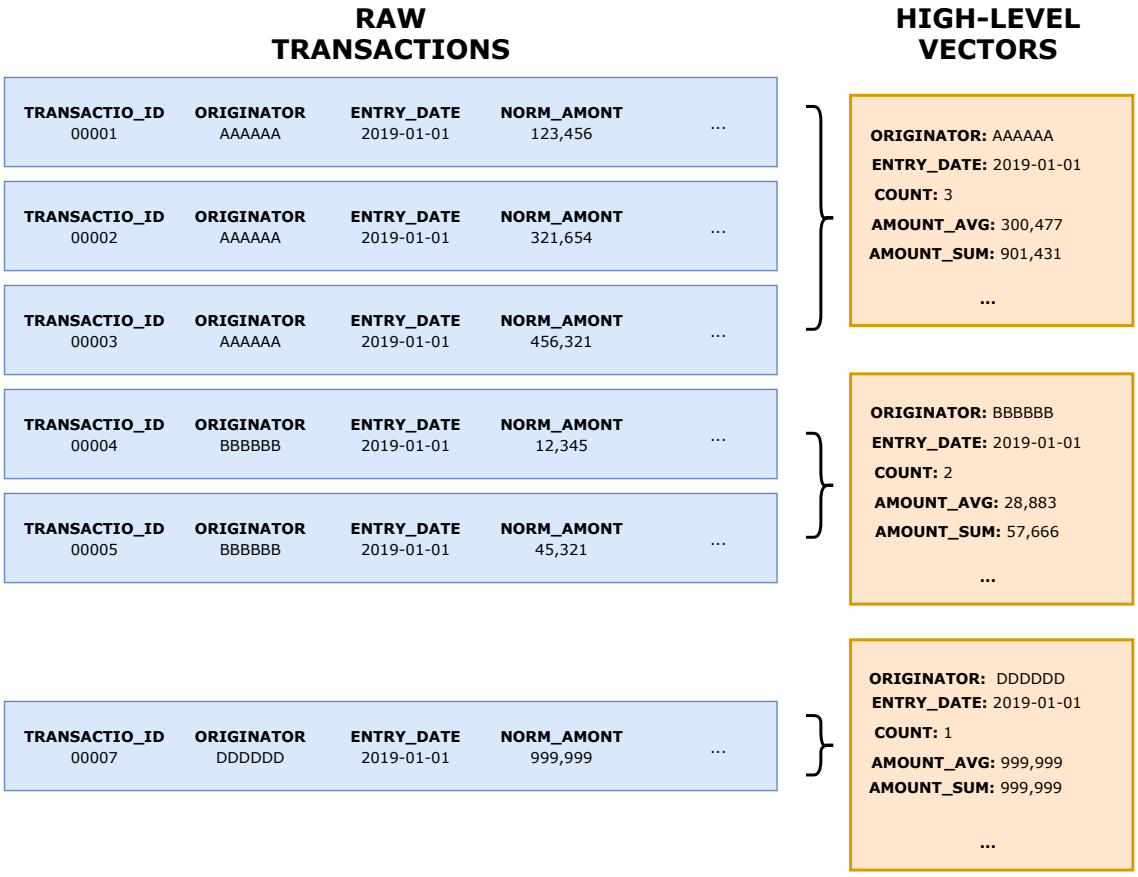


Figure 4.2: Aggregation example

4.3 Anomaly Detection

The anomaly detection models are the true core of AMARETTO and use high-level vectors to train the models used to detect anomalies. In this section we present the algorithms' that compose it and the technique employed, explaining first the characteristics fulfilled by these algorithms and the choice of using an ensemble. We present then the three different options, available to the analyst, for modelling customers' behaviours.

The ensemble is composed of the two of the best machine learning algorithms, *Isolation Forest* and *Random Forest*. An unsupervised model is essential to detect new anomalous patterns never seen before, while a supervised model improves the ability of the system to make future predictions. The choice of such algorithms

has been carried out with the aim of implementing a solution with the following characteristics:

- **robustness to outliers**: the fundamental characteristic of any good unsupervised anomaly detection algorithm. The presence of a few outliers in the training set should not compromise the efficacy of the solution, as the will is to model the behaviour of the majority of the data samples;
- **flexibility and domain independence**: AMARETTO should be capable of working on any kind of numeric dataset, independently from what those data represent and from the meaning of each data feature;
- **computational efficiency**: knowing that, in the financial domain, the algorithms might be used to train models individually for tens of thousands of users, we want them to be fast. This aspect acquires even more importance considering the general need for frequent retraining, to keep the models updated with the dynamic nature of the scenario. Also for what concerns generic anomaly detection, the computation time performance is important since even if only one model has to be trained, it is quite frequent to have hundreds of thousands of training vectors, many more than the ones required for training individual users' models. Therefore, we are required to put particular attention into the computation time performance of the algorithms, to obtain a solution that is not only theoretically but also practically suitable for the widest number of applications.

The *Isolation Forest* and the *Random Forest* fulfil all the listed requirements. The domain independence and noise robustness are intrinsic characteristics of these two learning algorithms. Within the Selection Strategies Module, the scores of the two modules are combined together. Since the two scores have different natures and meanings, we need to map them in a common domain. For our solution, we employ a technique called Weibull Probabilities Combination that transforms all the `anomaly_score` produced by each model into probabilities in the interval [0, 1]. It is later described after the individual algorithms are presented.

4.3.1 Isolation Forest

The Isolation Forest algorithm [37] is based on the isolation principle: it tries to separate data points from one another, by recursively randomly splitting into two partitions the dataset. The idea is simple: if a point is an outlier, it won't be

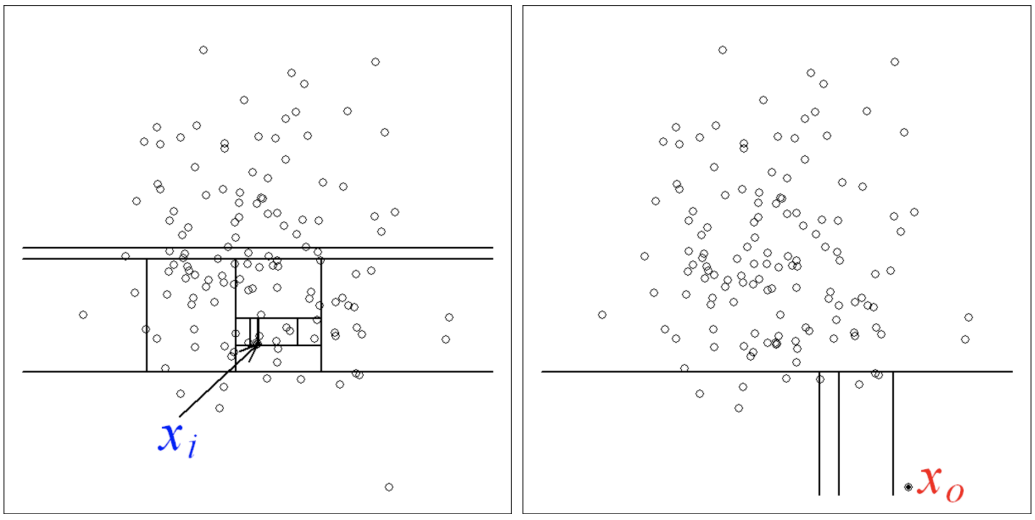


Figure 4.3: Isolation Forest splitting examples

surrounded by many other points, and therefore it will be easier to separate it from the rest of the dataset with random partitioning.

The algorithm uses the training set to build a series of isolation trees, which constitute the isolation forest; each isolation tree is built on a subset randomly sampled from the original training set. The isolation tree is built starting from the whole subset, which is split into two partitions and then recursively repeating the binary splitting on each pair of partitions generated at each step. The splitting is performed as follows: the algorithm randomly selects a feature and a splitting value - lying between the minimum and maximum values for that feature among the data points in that partition - and separates the data points that have that feature value above the splitting value from the ones that have it below. Recursively splitting the data points in this way will lead to isolating all the points sooner or later, i.e., to reach partitions with only one sample. As shown in Figure 4.3, the number of splits required by an outlier to remain isolated is likely to be much smaller than the one needed by a regular point, because of there's more space around it.

First, the concept of *path length* for a sample $\tilde{x}$ must be defined. The path length $h(\tilde{x})$ of $\tilde{x}$ is measured by the number of edges $\tilde{x}$ traverses in the *Isolation Forest* starting from the root node and until the traversal is terminated at an external node. Each edge corresponds to a binary split over a feature, and reaching an external node of the tree means that the point got isolated from all the others.

The anomaly score s of an instance x is defined as

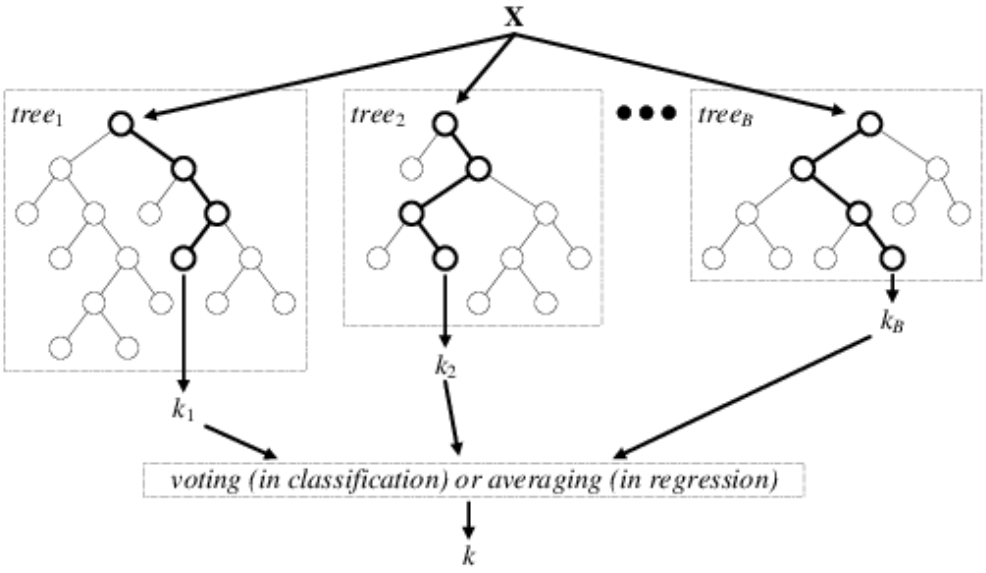


Figure 4.4: Random Forest example

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{4.1}$$

where n is the number of instances contained in a subset of the training set used to build an *Isolation Forest*, $E(h(x))$ is the average path length of x in all the trees of the forest, and $c(n)$ an estimate of the general average path length of a tree built with n instances, representing the average path length of an unsuccessful search in a Binary Search Tree, calculated as

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \tag{4.2}$$

where $H(i)$ is the harmonic number, i.e., the truncation of the harmonic series at the i-th term

4.3.2 Random Forest

In this section, we describe what a *Random Forest* [38] (shown in Figure 4.4) is and how it works, starting from its basic component, the *Decision Tree* [39].

A *Decision Tree* is a structure that allows the categorization of data points into different classes. Starting from the root node, each data point is deviated through different branches of the tree, depending on each node rule, until a leaf node is reached.

The node rules are simple conditions verified by a given attribute of the data point (e.g., is attribute $a_1 \geq K$? Or, for categorical features, is attribute a_2 equal to C_0?). In the leaf node, it will then be possible to determine the class of an input point, by simply looking for the class presenting maximum probability with respect to the leaf's samples.

A major advantage of this technique is the possibility to analyze, branch after branch, every decision made by the tree. This allows for a much better understanding of the decision taken and motivation driving the classification of points towards one or the other class.

DTs hold several good properties, however, they suffer a major issue: overfitting. There are several ways to cope with this problem, although, the most promising one in literature is to use an ensemble of several decision trees, called *Random Forest*. It consists of growing N trees from bootstrapped samples of the original dataset, thus growing many different weak learners, characterized by low bias and high variance. Nonetheless, the bagging ensemble of these weak learners will be a robust model, since the overall prediction is made averaging the prediction of the single trees. For binary classification tasks, the prediction can be interpreted as the probability of the sample to belong to the positive class. In usual applications, the desired output is a class, so you have to set a *threshold* value: if the predicted probability is greater than the threshold, the sample will be labelled as positive, otherwise as negative. However, in our case, the desired output is not a binary classification. Since the final output of our system is a ranking for transactions, we prefer the continuous value of probability more than a binary value 0 or 1. For example, if we have a threshold at 0.5, two samples predicted respectively as 0.6 and 0.9 are both classified to 1, but we prefer keeping that difference so that they get ranked differently. For this reason, we keep the probability prediction without applying any thresholding function.

4.3.3 Ensembling Technique

If you want to use several different models, you face a challenging issue: how can you combine the prediction of models that don't have the same output?

In AMARETTO we use a *Random Forest*, which outputs class probabilities $\in [0, 1]$, and an *Isolation Forest*, which outputs an `anomaly_score` $\in [-1, +1)$. Even if the models yield outputs in the same range (e.g., probabilities in $[0, 1]$), their prediction distribution could be significantly different, so the sum of the predictions could be misleading.

In order to cope with this problem, we use a score ensembling technique based on Weibull distribution. To perform the score transformation for each model, we

perform three steps:

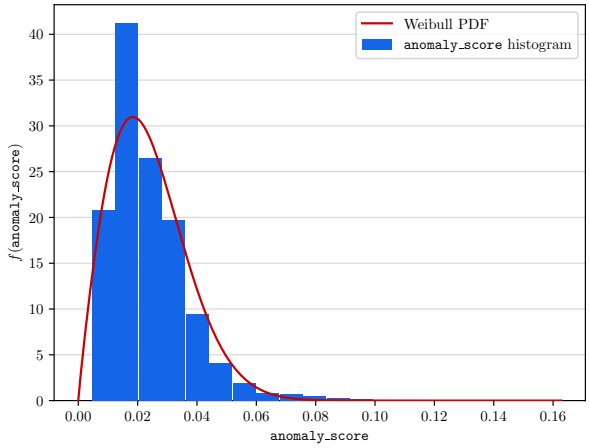
1. Weibull PDF fitting: we fit a Weibull distribution to the observed model's score distribution (i.e., $f($`anomaly_score`$)$); the choice of a Weibull curve is performed because of its shape flexibility (Figure 4.5(a)).

2. Weibull CDF derivation: once we have the Weibull PDF, we compute the corresponding CDF through integration (Figure 4.5(b)).

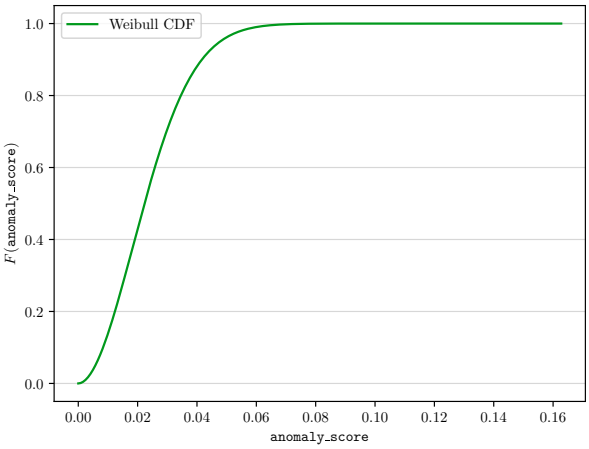
3. Scores conversion: for each new prediction $s_{\vec{x}}$, we redefine the `anomaly_score` as $F(s_{\vec{x}}) = P(\vec{x} \leq s_{\vec{x}})$. This is performed by plugging the old `anomaly_score` into the Weibull CDF (Figure 4.5(c)).

Figure 4.5: Anomaly score standardization

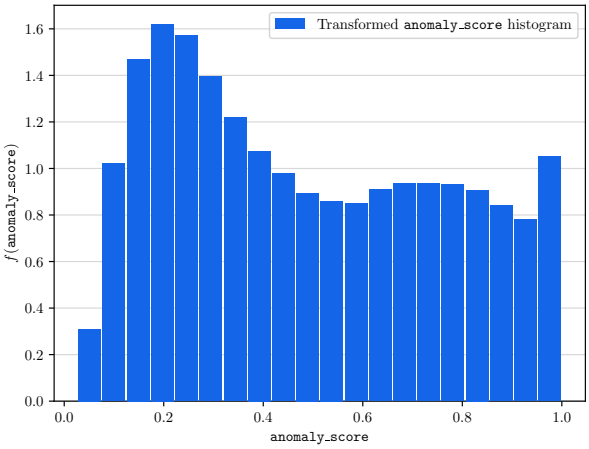
(a) Weibull distribution fitted `anomaly_score`.



(b) Fitted Weibull CDF: $F_X(\vec{x}) = P(X \leq \vec{x})$.



(c) Histogram of the `anomaly_score` after the conversion is performed.



4.4 Selection Strategies

This section defines the processes used to select a subset of high-level vectors for the subject matter expert to review as part of the active learning process. Taking into account the resources allocated by a financial institution on transaction monitoring, a fixed number (`daily_k`) of high-level vectors is shown to the subject matter expert. This process is divided into three stages.

4.4.1 First Stage: Unsupervised Selection Strategies

The purpose of the first stage is to detect new anomalous patterns as well as common anomalous patterns. As previously mentioned, the `anomaly_score` computed by the Isolation Forest is fundamental to detecting new anomalous patterns. For this stage two possible strategies are available.

In the firsts strategy, high-level vectors are ranked in decreasing order based on the `anomaly_score` generated by the Isolation Forest, and the topmost anomalous vectors are selected. However, this may not guarantee all types of anomalies are covered (i.e. selecting the top anomalies may result in having samples of the same anomaly type).

As previously evidenced in Section 2.2 by [27], it is important to diversify the type of unusual patterns that are selected. For this reason, the second active learning strategy uses clustering to group high-level vectors and drawing samples from each cluster based on the `anomaly_score`. Samples are drawn from each clusters starting from the least dense cluster until the desired number of samples has been reached. The decision of starting from the least dense cluster is motivated by the following assumption: given that the number of non-anomalous high-level vectors is greater than the number of anomalous vectors, the latter should form less dense clusters.

The clustering algorithm used for this strategy is HDBSCAN and the technical details are presented in the following section.

HDBSCAN

This algorithm is based on the work by [40] and [41]. The first step of the algorithm is to build a weighted graph, where each data point represents a node of the graph. The weights of this graph are computed using a metric called *mutual reachability distance* between two points, defined as:

$d(a, b) = \max\{core_k(a), core_k(b), d(a, b)\}$.

$core_k(x)$ is the core distance for a point x which is the distance between that point and its k-th farthest neighbour. The *mutual reachability distance* defines the

density of the areas around each point and it is used for spreading apart isolated points. A minimum spanning tree is constructed from the resulting graph using Prim's algorithm, which aims to connect every point in the graph whilst minimising the total weight of the edges in the resulting graph.

The next part of the algorithm focuses on building a hierarchy of clusters. This is achieved by removing all edges sorted by decreasing weight. This split process is recursively performed starting with the edges of the tree that have the lowest weight. This is defined by a parameter "minimum cluster size". The first step in cluster extraction is condensing down the large and complicated cluster hierarchy into a smaller tree. The key point is to consider points that are split close to a cluster, belonging to this single persistent cluster. To do so, the notion of minimum cluster size is applied. Again, a different measure than distance is defined to measure the persistence of clusters: $\lambda = \frac{1}{\text{distance}}$. For a given cluster, values λ_{birth} and λ_{death} represent the value when the cluster split off and became its cluster and the lambda value (if any) when the cluster split into smaller clusters respectively. In turn, for a given cluster, for each point p in that cluster we can define the value λ_p as the lambda value at which that point 'fell out of the cluster' which is a value somewhere between λ_{birth} and λ_{death} since the point either falls out of the cluster at some point in the cluster's lifetime, or leaves the cluster when the cluster splits into two smaller clusters. Now, for each cluster computes the stability as $\sum_{p \in \text{cluster}}(\lambda_p - \lambda_{\text{birth}})$. If the sum of the stabilities of the child clusters is greater than the stability of the cluster, then we set the cluster stability to be the sum of the child stabilities. If, on the other hand, the stability of the is greater than the sum of its children then we declare the cluster to be selected and unselect all its descendants. Once we reach the root node we call the current set of selected clusters our flat clustering and return that.

4.4.2 Second Stage: Random Forest Selection

The second stage of the selection phase relies on the probability score generated by the Random Forest. This process comprises two steps: the first one is the selection of the most anomalous aggregated transactions; the second one is the selection of the least anomalous aggregated transactions. The purpose of this stage is to take advantage of the higher accuracy of the Random Forest to reinforce the information contained in the labelled dataset.

4.4.3 Third Stage: Uncertain Datapoints Selection

In the third and last stage of the selection phase, the high-level vectors, for which the two models show the most uncertainty, are selected. To assess the level of uncertainty, two approaches can be followed:

- the first approach uses the probability scores generated by the supervised model. Samples whose probability is close to 0.5, have a high chance of being selected due to their high entropy and hence uncertainty;
- the second approach takes into account the difference between the scores generated by the supervised and unsupervised model. A discrepancy the in score for each set of high-level vectors indicates that the outputs of the two models disagree; for this reason, samples, where the score discrepancy is close to 1.0, are selected (i.e. the models are completely in disagreement on whether the vectors are anomalous or not).

Both the strategies provide a varied selection of samples that allows the supervised model to be trained not only anomalous aggregation but also on nominal one. Furthermore, the selection of samples for which the models are uncertain leads the analyst to inspect transactions that are well hidden within the nominal ones.

4.5 Automatic Hyper-Parameters Optimization

The two models we use in AMARETTO, *Isolation Forest* and *Random Forest*, have several hyper-parameters that need to be tuned to achieve optimal performances. There are several ways to do this, from manual to automatic activity. The hyper-parameters optimization could be done by hand, but it's a very tedious operation. For this reason, among the automatic methods, we adopt *Bayesian Optimization* [42] because it allows us to save an enormous amount of time.

Bayesian Optimization is a probabilistic model-based approach for finding the minimum of any function that returns a real-value metric. In particular, it finds the input value or set of values to an objective function that yields the lowest output value, called a "loss". Typically, in machine learning, the objective function is multidimensional because it takes in a set of model hyperparameters. For simple functions in low dimensions, it is possible to find the minimum loss by creating a grid of input values (*grid search*) and seeing which one yields the lowest loss. Or it is possible to pick random values (*random search* [43]). As long as evaluations of the objective function are cheap, these uninformed methods might be adequate.

However, for complex objective functions, it could require a huge amount of time and computational power. *Bayesian Optimization*, also called *Sequential Model-Based Optimization (SMBO)*, builds a probability model of the objective function that maps input values to a probability of a loss: $p(loss|input_values)$. The probability model also called the *surrogate* or *response surface*, is easier to optimize than the actual objective function. Bayesian methods select the next values to evaluate by applying a criterion (usually *Expected Improvement*) to the surrogate. The concept is to limit evals of the objective function by spending more time choosing the next values to try, focusing on previous values that return a lower loss. *Bayesian Reasoning* means updating a model based on new evidence, and, with each eval, the surrogate is re-calculated to incorporate the latest information. The longer the algorithm runs, the closer the surrogate function comes to resembling the actual objective function. Bayesian Optimization methods differ in how they construct the surrogate function: common choices include *Gaussian Processes*, *Random Forest Regression* and the *Tree Parzen Estimator (TPE)*.

The hyper-parameters that need to be set are the following:

- *Isolation Forest*:
  - `n_estimators`: the number of base estimators in the ensemble;
  - `max_features`: the number of features to draw from the training set to train each base estimator;
  - `contamination`: the amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function;
- *Random Forest*:
  - `num_trees`: the number of decision trees that compose the forest;
  - `max_depth`: the maximum depth to grow each decision tree;
  - `min_samples_split`: the minimum number of samples required to split an internal node;
  - `min_samples_leaf`: the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches;
  - `max_features`: the number of features to consider when looking for the best split.

Chapter 5

Implementation Details

In this chapter, we describe all the details regarding the implementation of AMARETTO. The entire framework is developped using `Python v3.7.2`. Firstly, we present an overview of the main modules that compose our tool, then we describe each one of them in detail, together with some algorithms.

5.1 AMARETTO Architecture

AMARETTO is composed of three main modules:

- the **Data Preprocessing Module** is the module that new raw transactions (low-level vectors) from the bank to compute the customers' profiles than a behavioural modelisation (high-level vectors) is performed. The system computes aggregate features, capturing the customers' behaviours signature within a period, ready to be pipelined to the learning algorithms (Section 5.2);
- the **Anomaly Detection Module** is the module that employes the algorithm using to detect the anomalous patters. It is divided in two submodules:
  - the **Unsupervised Module** that is used to detect anomalous patterns and, above all, discover new anomalous patterns. It returns an `anomaly_score` to each new high-level vectors (Section 5.3.1);
  - the **Supervised Module** is used to perform the supervised scoring and the binary predictions. It uses the labeled feedbacks provided by the analyst to train itself (Section 5.3.2);

- the **Selection Strategies Module** is the module in charge of selecting the high-level vectors to be shown to the analyst which will explore them and label them as fraud or genuine. The feedback is then stored into the local labelled database, that will be used in future to train the supervised model (Section 5.4);
- the **Hpyer-parameters Optimization Module** that automatically finds the optimal value for the hyper-parameters of our framework (Section 5.5).

A full schema of AMARETTO architecture is depicted in Figure 4.1. In the following sections, we will deeply describe each of these modules, what models we chose for them and how they work.

5.2 Data Preprocessing Module

We present here the characteristics of the **Data Preprocessing Module**, whose funcionalities have been designed in collaboration with NapierAI experts to answer all the expected needs of an analyst in the capital market domain.

5.2.1 Raw Data Parsing

The raw data parsing is the first key task performed by the module. The process is performed starting from data shaped as a `.csv` (comma-separated values) file, where each column corresponds to a base feature. The file containing the raw transactions is then transformed into a `DataFrame` provided by `Pandas v0.23.4`. The DataFrame columns are mapped in the following `dtype`:

- **Transaction ID**: `str`;
- **Originator**: `str`;
- **Originator_ID**: `str`;
- **InputOutput**: `str`;
- **EntryDate**: `pandas.DateTime`
- **Market**: `str`;
- **Product ISIN**: `str`;
- **Product Type**: `str`;

- **Product Class**: `str`;
- **Normalized Amount**: `float`;
- **Currency**: `str`;
- **Anomaly**: `int`

5.2.2 High-Level Features Derivation

After parsing the raw data, the aggregation process is carried out using a fixed, defined aggregation window for all the customers. For each one, the aggregation window starts sliding from his oldest transaction performed, and all the activities inside the window at each iteration are aggregated into a single DataFrame containing all the high-level vectors. When aggregating labeled data, necessary to test the efficacy of the solution (remember that labels are not used in any way for the anomaly detection task), if at least one of the raw transaction vectors included in the aggregation is marked as anomalous, then the high-level vector itself is marked as anomalous too.

Extracting financial information First of all, the `EntryDate` column is used to extract temporal features like `Weekday`, `Month`, `Hour`. The DataFrame containing the transactional data is grouped by using the `Originator` and the temporal features mentioned before. Then, the financial features are extracted from the `DataFrameGroupBy` object. For each window in which the user performed at least one transaction, a raw in the final Dataframe is created, collecting all the activities of the customer. These records are uniquely indexed through the features used to create the `DataFrameGroupBy`.

The financial features extracted in this phase are designed to model the behaviourial signature of the user in each window, capturing the spending patterns. The features of interest comprises combination between `Currency`, `Product Class`, `Product Type`, and `InputOutput` columns. These high-level features are carefully selected exploting the domain expertise of the NapierAI team, to be able to detect all kinds of behavioral variations that might indicate a fraudulent activity.

The first step is to transform the `Product Class`, `Product Type` into new features called `Cash` and `Collateral` that indicate if a transaction is performed through a *simple transfer* using *cash* or other types of *security*. Then, a first aggregation, called `Amount_IO_Aggregation`, is carried out extracting information about the amount of the transactions included in the aggregation window as statistical features like

`mean_amount`, `sum_amount` or `code_small_amount`, `code_round_amount`. Furthermore, during this step, `InputOutput_delta` and `Collateral_delta` are determined, indicating the difference between bought and sold operations or the difference between collateral and the other secutiries. Afterwards, another `DataFrameGroupBy` object, called `Product_Currency_Aggregation`, is created aggregating by `Currency`, `Product Class`, `Product Type`, and `InputOutput` columns and computing the `mean_amount`, `sum_amount`, and `count` for each different value of the pivot columns. Finally, the two aggregation, `Amount_IO_Aggregation` and `Product_Currency_Aggregation`, are merged together and indexed used the `Originator` and the temporal columns. This is the final `DataFrame` that contains the high-level vectors used to train or be analysed by the *Anomaly Detection Module*.

An example of high-level vector is shown in Table 5.1, while in Algorithm 2 is shown the pseudocode of the *high-level features derivation*.

Algorithm 2: High-level features derivation

Input: `transaction_df`, `window_width`, `client_flag`
Output: `high_level_df`

drop useless columns from `transaction_df`

extract temporal features from `transaction_df`

extract features regarding `Cash` and `Collateral`

if *client_flag* == *True* **then**
    `group_df` = groupby temporal feautures and `Originator`
else
    `group_df` = groupby temporal feautures

`Amount_IO_Aggregation` = aggregate and compute 'mean', 'count', 'sum' on `[Amount, InputOutput and Collateral]`

`Product_Currency_Aggregation` = aggregate and compute 'mean', 'count', 'sum' on `[Cash, Collateral and Currency]`

`high_level_df` = merge `[Product_Currency_Aggregation and Amount_IO_Aggregation]`
return *high_level_df*

Feature name	Value
Originator	Client_304
EntryDate	2019-01-01 00:00:00
Weekday	1
Hour	7
Night	1
Morning	0
Evening	0
Anomaly	1
Transactions_count	73
Normalized Amount_sum	20886919.32000001
Transactions_count_Small_Amount	15.0
Transactions_count_Round_Amount	0.0
InputOutput_delta	17
Collateral_delta	-39
Transactions_count_Buy_Collateral_Cash_USD	0.0
Transactions_count_Buy_Collateral_Cash_RUB	1.0
Transactions_count_Buy_Collateral_Security_USD	3.0
Transactions_count_Buy_Collateral_Security_RUB	3.0
Transactions_count_Sell_Collateral_Cash_USD	1.0
Transactions_count_Sell_Collateral_Cash_RUB	3.0
Transactions_count_Sell_Collateral_Security_USD	2.0
Transactions_count_Sell_Collateral_Security_RUB	4.0
Normalized Amount_sum_Buy_Collateral_Cash_USD	0.0
Normalized Amount_sum_Buy_Collateral_Cash_RUB	1952.2
Normalized Amount_sum_Buy_Collateral_Security_USD	34888.65
Normalized Amount_sum_Buy_Collateral_Security_RUB	835052.6699999999
Normalized Amount_sum_Sell_Collateral_Cash_USD	16782.43
Normalized Amount_sum_Sell_Collateral_Cash_RUB	94323.17
Normalized Amount_sum_Sell_Collateral_Security_USD	27584.75
Normalized Amount_sum_Sell_Collateral_Security_RUB	191164.09000000003
Normalized Amount_mean_Buy_Collateral_Cash_USD	0.0
Normalized Amount_mean_Buy_Collateral_Cash_RUB	1952.2
Normalized Amount_mean_Buy_Collateral_Security_USD	11629.550000000001
Normalized Amount_mean_Buy_Collateral_Security_RUB	278350.88999999996
Normalized Amount_mean_Sell_Collateral_Cash_USD	16782.43
Normalized Amount_mean_Sell_Collateral_Cash_RUB	31441.056666666667
Normalized Amount_mean_Sell_Collateral_Security_USD	13792.375
Normalized Amount_mean_Sell_Collateral_Security_RUB	47791.02250000001

Table 5.1: High-level vector example

Modeling Strategies Due to the lack of the a sufficient amount of data in a customer profile or because the analyst wants a difference point of view, the high-level vectors can be model via a *individual profile modeling* or a *global profile modeling*. Through a `boolean` variable, the high-level features derivation can take into account the `Originator` column or not. Obviously, the final DataFrame generated to globally model the profiles contains less instances than a the individual one, since in the latter are generated $n_{customers} \times n_{windows}$ instances.

5.3 Anomaly Detection Module

Here, we present the main processes performed by the two models included in the *Anomaly Detection Module* of AMARETTO.

5.3.1 Unsupervised Module

The *Unsupervised Module* is one of the core components of AMARETTO. It is in charge of performing anomaly detection in an unsupervised way, using the high-level vectors. The algorithm running within this module is the *Isolation Forest* [37], developped by [44]. In order to fulfill our requirements, we built a wrapper class around the main algorithm that implements the methods we need, receiving and returning pandas `DataFrames` and `Series`. The principal functions implemented are the following:

1. **decision function**: computes the `anomaly_score` using given the depth of the leaf containing the observation, which is equivalent to the number of splittings required to isolate this point;

2. **fit Weibull parameters:** at training time, we first train the *Isolation Forest*, then we fit a Weibull PDF to the `anomaly_score` distribution. For the Weibull we used the implementation provided by the statistical module of `SciPy v1.1.0` [45];

3. **anomaly scoring:** it's the method used for transactions scoring. It basically calls the `decision function`, then passes the `anomaly_score` through the Cumulative Distribution Function (CDF) of the Weibull fitted in training phase, thus obtaining a normalized `anomaly_score` in the $[0, 1]$ space that can be compared to the supervised one.

5.3.2 Supervised Module

The *Supervised Module* is the other main component of AMARETTO *Anomaly Detection Module*. It's composed of a *Random Forest* that gets trained with the labeled high-level vectors database created by collecting human analyst feedbacks. To develop the wrapper class around the *Random Forest*, we used the implementation provided by the ensemble module in `scikit-learn v0.20.3` [44]. The main custom functions we added are the following:

1. **fit Weibull parameters:** during the training phase, we first train the *Random Forest*, then we fit a Weibull PDF to the probability distribution of the just fitted *Random Forest*;

2. **probabily estimation**: computes the *probabily* that an instance is anomalous or not computed as the mean predicted class probabilities of the trees in the forest. The class probability of a single tree is the fraction of samples of the same class in a leaf;

3. **anomaly scoring:** it's a custom method to score the transactions based on the *probability estimation* of the *Random Forest*. Instead of directly using the predicted probability for class 1, we pass it through the CDF of the Weibull fitted during training phase, in order to have results that can be compared with the *Isolation Forest*'s `anomaly_score`;

4. **binary prediction:** it's the function called to return the *binary prediction* computed by the *Random Forest*. The purpose of this functions is to create a wrapper that returns a `pandas DataFrame` merging the *predictions* and the *high-level vectors*.

5.4 Selection Strategies Module

In this section, we detailed the the processes and the algorithms used to select a subset of high-level vectors for the analyst to review as part of the active learning process. After the scoring phase, the `DataFrame` containing the `anomaly_score` and the high-level vectors features is the input of this module and through every stage of the selection process is modified by the submodule related to each stage. Since the process is divided in three stages, the `daily_k` samples are equally shared between the stages.

5.4.1 First Stage: Unsupervised Selection Strategies

The *Unsupervised Module* returns an `anomaly_score` for each high-level vector. This score is fundamental because it indicates how much a vector is anomalous and therefore it is used to select the most interesting ones for the analyst. In this stage, AMARETTO leverages the ability of the *Isolation Forest* to detect, in addition to common anomalies, new anomalous patterns. As explained in Section 4.4.1, two approaches can be employed in this stage.

In the first one, the `DataFrame` is sorted in decreasing order by `anomaly_score` and the $ratio_{unsup}$ most anomalous vectors are selected, as show in Algorithm 3.

Algorithm 3: First stage: *SELECT-TOP*

Input: $\mathcal{U}^t$, $ratio_{unsup}$
Output: $sample^t_{unsup}$

Sort $\mathcal{U}^t$ by unsupervised score
Select $ratio_{unsup}$ most anomalous aggregations. Call it $\mathcal{C}$
Append $\mathcal{C}$ to $sample^t_{unsup}$ i.e.
$sample^t_{unsup} = sample^t_{unsup} \cup \mathcal{C}$

return $sample^t_{unsup}$

The second approach employes more steps but it allows the system to select different types of anomalies, providing a wider knowledge in the training set for the *Supervised Module*. Also in this approach, the `DataFrame` is ranked using the `anomaly_score` and all the vectors with an `anomaly_score` higher than the p_{unsup} percentile are then considered. Afterwards, the cluster process is applied on this subset of vectors in order to obtain different groups of vectors. Then, `cluster_ratio` is defined taking into account $ratio_{unsup}$ and indicating the number of samples to be selected from each cluster. Starting from the least populated cluster, `cluster_ratio` vectors are selected untill a `DataFrame` containing $ratio_{unsup}$ samples is obtained. The Algorithm 4 presents the pseudocode of this approach.

5.4.2 Second Stage: Random Forest Selection

The second stage of the selection phase relies on the *probability score* generated by the *Random Forest*. This process comprises of two steps: the first one is the selection of the most anomalous aggregated transactions; the second one is the selection of the least anomalous aggregated transactions. In this stage, the previously selected

Algorithm 4: First stage: *SELECT-DIVERSE*

Input: $\mathcal{U}^t$, $ratio_{unsup}$, p_{unsup}
Output: $sample^t_{unsup}$

Sort $\mathcal{U}^t$ by unsupervised score
Find p_{unsup} percentile. Call it x_p
Select $x_i \cup \mathcal{U}^t$ with score $s_i > x_p$. Call it $anoms_{unsup}$
Run HDBSCAN algorithm on $anoms_{unsup}$
Define $cluster_{ratio} = \max\{1, \frac{ratio_{unsup}}{n_{clusters}}\}$
Sort cluster by $cluster_{density}$
for $i = 0, \ldots, n_{clusters}$: **do**
    Select $cluster_{ratio}$ samples from $cluster_i$. Call it $\mathcal{C}_i$
    Append $\mathcal{C}_i$ to $sample^t_{unsup}$ i.e.
    $sample^t_{unsup} = sample^t_{unsup} \cup \mathcal{C}_i$
    if $|sample^t_{unsup}| \geq ratio_{unsup}$ **then**
        Break

return $sample^t_{unsup}$

vectors are filtered out from the input `DataFrame` in order to not overlap the selected vectors.

5.4.3 Third Stage: Uncertain Datapoints Selection

In the third and last stage of the selection phase, we exploit the `anomaly_score` of both models in order to select the high level vectors for which the two models show contraddictory `anomaly_score`.

As explained in Section 4.4.3, two approaches can be employed in this stage.

In the first one, only the *probability estimation* provided by the *Random Forest* is considered. A *probability* close to 0.5 means that the *Random Forest* is completely uncertain if the high-level vector is anomalous or not. For this reason, with this approach, we focus on these vectors. From the `anomaly_score` in the input `DataFrame`, the *distance* of each vectors to 0.5 is computed. Then, the vectors are sorted in ascending order by *distance* and $ratio_{sup}$ samples are selected. Algorithm 5 shows the pseudocode of the algorithm.

In the second approach, the `anomaly_score` computed by the two models are taken into account. In this case, the score provided by the *Isolation Forest* and the

Algorithm 5: Third stage: *SELECT-ENTROPY*

Input: $\mathcal{U}^t, ratio_{sup}, p_{center}$
Output: $\mathcal{C}^{center}$

Compute the distance of the score for each aggregation i to the center $= 0.5$.
$dist_i = |s_i - 0.5|$
Sort $dist_i$ in ascending order
Select $ratio_{sup} \times p_{center}$ least distant aggregations. Call it $\mathcal{C}^{center}$

return $\mathcal{C}^{center}$

Random Forest are normalized using the Weibull techinque in order to be compared. With this approach, AMARETTO focuses on the high-level vectors which shows a different rating between the two models. In this case, a *difference* between the two `anomaly_score` is computed. The derived `DataFrame` is then sorted by the *difference* and $ratio_{sup}$ vectors are selected. Algorithm 6 shows the pseudocode of the algorithm.

Algorithm 6: Third stage: *SELECT-CONFLICT*

Input: $\mathcal{U}^t, ratio_{sup}, p_{center}$
Output: $\mathcal{C}^{center}$

Compute the difference between the supervised score and the unsupervised score for each aggregation i. Call it unc_i
Sort unc_i in descending order
Select $ratio_{sup} \times p_{center}$ most uncertain aggregations. Call it $\mathcal{C}^{center}$

return $\mathcal{C}^{center}$

Both the strategies provide a varied selection of samples that allows the supervised model to be trained not only anomalous aggregation but also on nominal one. Furthermore, the selection of samples for which the models are uncertain leads the analyst to inspect transactions that are well hidden within the nominal ones.

5.5 Hyper-Parameter Optimization Module

The two models we use in AMARETTO, *Isolation Forest* and *Random Forest*, have several hyper-parameters that need to be tuned in order to achieve optimal performances. We adopt *Bayesian Optimization* [42] that is a probabilistic model based

approach for finding a set of input values to an objective function that yields the lowest loss. The objective function implemented in our case takes into account the $AUROC$ because an higher $AUROC$ means a high value for TPR and a low value for FPR.

$f(x) = 1 - AUROC$

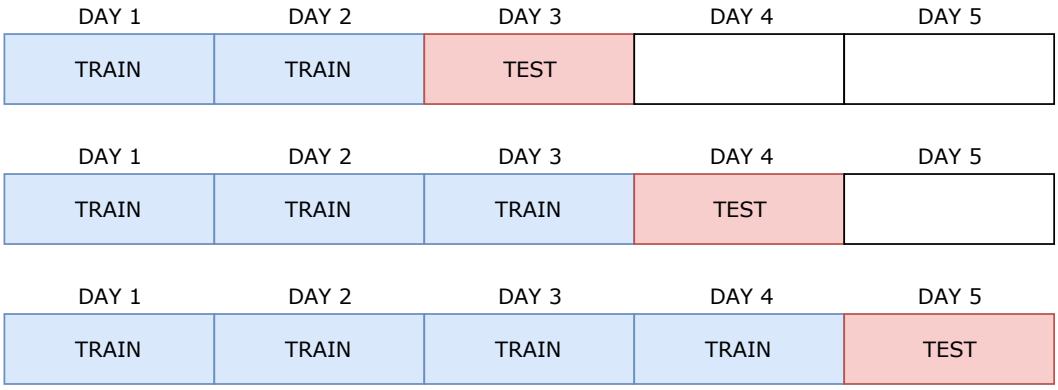


Figure 5.1: Walk-forward example

Since the framework is designed to be employed in a real-world scenario on a daily basis, the hyper-parameters optimization tests are simulating a walk-forward approach [46], as showed in Figure 5.1.

The library allows us to choose different search spaces for each parameters that has to be tuned. The following search spaces are defined:

- *Isolation Forest*:

  `n_estimators`: uniform distribution between $[50, 150]$;

  `max_features`: uniform distribution between $[0.5, 1.0]$;

  `contamination`: uniform distribution between $[0.0001, 0.15]$;

- *Random Forest*:

  `num_trees`: uniform distribution between $[50, 150]$;

  `max_depth`: uniform distribution between $[3, 12]$;

  `min_samples_split`: loguniform distribution between $[log(3), log(6)]$;

  `min_samples_leaf`: loguniform distribution between $[log(2), log(4)]$;

  `max_features`: uniform distribution between $[0.7, 1.0]$.

Chapter 6

Experimental Validations

In the following chapter, we exhibit several experiments conducted to assess the performance and effectiveness of AMARETTO. Firstly, the goals of the experiments are described. Secondly, we present the base approach used in our experiments and metrics used to perform our evaluations. Then, we show the experimental setup and libraries used in our experiments. Afterwards, we show the experiment conducted on to asses the performances of the components of AMARETTO. Finally, we will assess the performance of the system proving its effectiveness specifically in the detection of money laundering patterns recognized in real-world scenarios.

First, we compare *Isolation Forest* used in AMARETTO with state-of-the-art unsupervised solutions, as outlined in [26], [21], to confirm that our choice is the best for an anomaly detection system (Section 6.4). Then, we test the same unsupervised techniques to assess their prediction ability with different daily budgets (Section 6.5). Afterwards, we evaluate *Random Forest* against the supervised solutions cited in Section 2.2 to prove that the algorithm we adopt is the best classifier for anomaly detection (Section 6.6). Furthermore, we prove the importance of an unsupervised model in combination with a supervised one in detecting new anomalous patterns (Section 6.7). Finally, we show the evaluation tests conducted on AMARETTO: initially, we compare the different selection strategies of AMARETTO (Section 6.8); finally, we compare AMARETTO with AI^2, a state of the art active learning framework, in a real-world scenario (Section 6.9).

6.1 Goals

The major goal for this thesis is to implement a framework capable of learning over time an efficient way to recognize fraudulent patterns, even starting from a

situation where there is no historical labelled data, with a limited human analysis budget. Analysing these goals, we can see that these specifications are in contrast with themselves because it is possible to achieve a very high detection rate only with a massive labelled dataset to train the system with and this requires a significant effort in term of human analysis. Keeping this in mind, we face this trilemma choosing the best trade-off possible for our tool, trying to achieve the following goals:

1. ability to start in a setting where no past labelled data is available;
2. high detection rate, even with a limited human analysis budget;
3. produce an acceptable number of false positives and false negatives;

6.2 Experimental Setup

6.2.1 Hardware

All the experiments are executed on a remote machine with the following characteristics:

- 377GB of RAM;
- Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz x40;

6.2.2 Software

The following software with the described characteristics is employed:

- Ubuntu 16.04.6 LTS
- Python 3.7.2

The following libraries are employed within the developing of the code:

- `catboost v0.13.1` for the implementation of the CatBoost classifier in Section 6.6;
- `copulas v0.2.3`, for the implementation of the Copula-based model used in Section 6.4;

- `hdbscan v0.8.23`, for the implementation of the HDBSCAN algorithm in Section 4.4.1;
- `hyperopt v0.1.2`, for the implementation of *Bayesian Optimization* in Section 4.5;
- `keras v2.2.4` [47] with `tensorflow v1.13.1` backend, for the implementation of *Auto Encoder* (AE) in Section 6.4;
- `lightgbm v2.2.3`, for the implementation of *eXtreme Gradient Bosting* (XGB) in Section 6.6;
- `numpy v1.16.2`, for mathematical computations;
- `pandas v0.24.2` [48], for managing data in Section 4.2;
- `scikit-learn v0.20.3` [44], for implementing the algorithm of *Isolation Forest*, *Random Forest* and the tecnhiques used in Section 5.3, in Section 6.4 and in Section 6.6

6.3 Evaluation Approach and Metrics

The data contained in our dataset can be considered as time-series data. For this reason, we split the dataset into two sets: the first one contains the first 7 weeks of transactions which is used for training the models and for the hyper-parameter optimization; the second set is used to evaluate the model performance by running tests and it includes 5 weeks of transactions. Given the temporal link of the data, we used a walk-forward testing approach [46] for evaluating the models. For the hyper-parameter optimization, we used *Bayesian Optimization* [42], explained in Section 4.5 because of its ability to achieve accurate parameter selection within a reasonable amount of time.

In this section, we introduce the metrics used for evaluating AMARETTO. A *True Positive* (TP) is an anomalous high-level vectors correctly classified as anomalous, *False Positive* (FP) is a legitimate high-level vectors wrongly ranked as anomalous, a *False Negative* (FN) is an anomalous vector wrongly ranked as legitimate, and a *True Negative* (TN) is a legitimate vector correctly ranked as non-anomalous. The common evaluation metrics used to assess the system performance are:

- *Accuracy* that measures the percentage of high-level vectors correctly classified:

$$ACC = \frac{TN + TP}{TN + FP + FN + TP}; \tag{6.1}$$

- *Precision* that is the proportion of TP over the vectors considered as anomalies:

$$Precision = \frac{TP}{TP + FP}; \tag{6.2}$$

- *True Positive Rate* or *Recall* that computes the percentage of correctly identified anomalous vectors:

$$TPR = \frac{TP}{TP + FN}; \tag{6.3}$$

- *False Positive Rate* that computes the percentage of legitimate vectors that are wrongly identified as anomalous:

$$FPR = \frac{FP}{TN + FP}; \tag{6.4}$$

- *FScore* that measures the harmonic mean between the Recall and the Precision:

$$FScore = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}; \tag{6.5}$$

- *Matthews Correlation Coefficient* that measures the quality of the detection rate in terms of the correlation coefficient between the observed and predicted classifications; a coefficient of $+1$ represents a perfect ranking, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation:

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \tag{6.6}$$

- *Area Under the Receiver Operating Characteristic (ROC) Curve* is the area under the *ROC curve*, obtained by plotting the TPR against the corresponding FPR at various threshold settings; the $AUROC$ gives a measure of the solution performance, where a perfect model has an $AUROC$ of 1.

The test data is very imbalanced (0.27% of anomalous transactions), so metrics like accuracy, are not very meaningful. However, to make a fair comparison with the state of the art solution, they are included as a reference. The *AUROC* is a useful indicator for benchmarking algorithms; if the *ROC curve* of a model is consistently higher than the curve of other estimators, this indicates the former achieves better performance. For these reasons, we use the *AUROC* and the *ROC curve* to assess the performance of various unsupervised models and the common metrics described above for assessing the performance of the supervised models.

We also considered an additional metric to account for class imbalance and different classification costs. This cost metric is described in [49] as:

$$Cost = FP + C\_R \times FN \tag{6.7}$$

A normalization process can be applied to obtain a value that is independent from the number of transactions:

$$Norm\_Cost = \frac{FP + C\_R \times FN}{TN + FP + C\_R \times (TP + FN)} \tag{6.8}$$

As suggested in [49], 100 is a reasonable estimation of C_R, that is the *cost ratio* between *FN* and *FP*, and this was the value used to assess the optimal operating condition of our system, however, it could be set to reflect real costs of anomalous transactions based on scenarios. This metric takes into account the cost of false positives for an institution. A unit cost is applied to a *FP*, whilst a higher cost is applied for a *FN*, since the cost of allowing a money launderer in the system is hundreds of time higher than the cost of false positives, and it may result in fines for the institution.

6.4 Experiment 1: unsupervised models comparison

With this test, we compared models used in state of the art solutions with *Isolation Forest*. In [21], AE are used for outlier detection; we also tested Variational AE as proposed in [50]. In [26], the unsupervised models used comprise of a Matrix-decomposition model, a Density-based model and an AE, using PCA as a Matrix decomposition model [51] and using a Copula distribution as a Density-based model. We also tested a threshold-based model that uses mean and standard deviation computed for each feature of the high-level vector. Given these descriptive statistics, we compute a one-sided threshold as the sum of mean and standard deviation. In order to score new samples, all features that exceed their respective threshold, add the surplus to the risk score, while features below the threshold yield a risk score of 0.

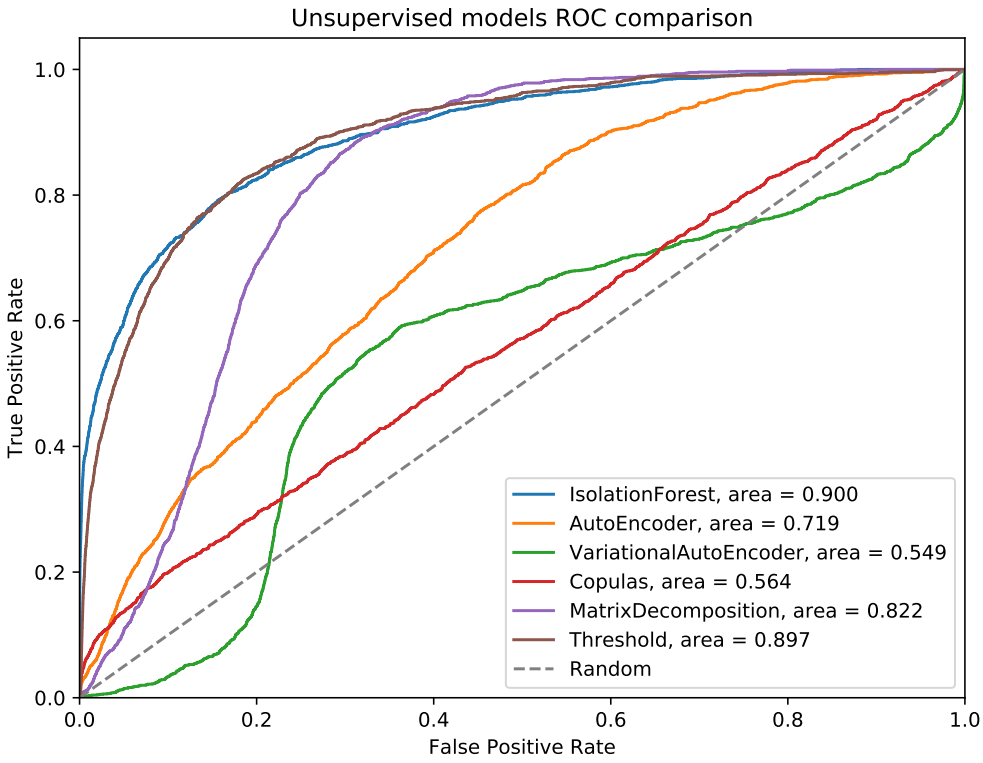


Figure 6.1: Experiment 1: ROC unsupervised algorithms

As shown in Figure 6.1, *Isolation Forest* is the model that exhibits the best

performance with an *AUROC* of 0.9. Surprisingly, the threshold-based model and the matrix decomposition-based model outperformed the AE, which is considered one of the best models for outlier detection.

6.5 Experiment 2: daily budget K estimation

For this experiment, we benchmark the performances of all the unsupervised models analysed in the previous experiments when varying the number of samples reviewed each day by the analyst. For every day existing in the test set, each model computes the `anomaly_score` for the high-level vectors which is then used to rank the vectors. Then the K top anomalous vectors are considered anomalous e.g., for $K = 10$ the first 10 vectors with the highest scores are selected for the review. The purpose of this experiment is to assess the best `daily_k` that allows the system to achieve a suitable detection rate and to reduce the cost for a financial institution and the effort of the subject matter expert in reviewing the high-level vectors.

Please note that we split the table of the result in 4 parts, for the sake of readability. The metrics presented in Table 6.1, Table 6.2, Table 6.3 and Table 6.4 are the mean metric computed for each technique and for each budget. As shown in the tables, the *Isolation Forest* is the model that achieves the best results for every budget K, achieving an average *Precision* of 0.904 and an average *FPR* of 0. This means that the *Isolation Forest* allows the analyst to focus only on the most anomalous vectors. The matrix decomposition-based model achieves worse results in this experiment compared to the Isolation Forest, achieving comparable performance only with a higher budget. The daily budget values considered in this experiment represent a small percentage of the daily vectors that are generated, for this reason, the *FNR* is high for small daily budget; while it is reducing as the budget increases.

Table 6.1: Experiment 2: daily budget K estimation (I)

K	Metric	AutoEncoder	Copulas	IsolationForest
10	**AUC**	0.612	0.558	**0.897**
	Accuracy	0.991	0.991	**0.993**
	FScore	0.005	0.055	**0.246**
	False Negative Rate	0.997	0.968	**0.858**
	False Positive Rate	0.001	0.001	**0.0**
	Matthews	0.003	0.078	**0.357**
	Normalized Cost	0.448	0.435	**0.385**
	Precision	0.015	0.208	**0.904**
	True Positive Rate	0.003	0.032	**0.142**
20	**AUC**	0.612	0.558	0.897
	Accuracy	0.989	0.99	**0.993**
	FScore	0.007	0.061	**0.313**
	False Negative Rate	0.996	0.96	**0.794**
	False Positive Rate	0.003	0.002	**0.001**
	Matthews	0.003	0.067	**0.364**
	Normalized Cost	0.448	0.432	**0.357**
	Precision	0.013	0.129	**0.656**
	True Positive Rate	0.004	0.04	**0.206**
30	**AUC**	0.612	0.558	0.897
	Accuracy	0.988	0.989	**0.992**
	FScore	0.015	0.068	**0.328**
	False Negative Rate	0.989	0.95	**0.758**
	False Positive Rate	0.004	0.003	**0.002**
	Matthews	0.01	0.068	**0.348**
	Normalized Cost	0.446	0.428	**0.342**
	Precision	0.022	0.108	**0.513**
	True Positive Rate	0.011	0.05	**0.242**

Table 6.2: Experiment 2: daily budget K estimation (II)

K	Metric	MatrixDecomposition	Threshold-base
10	**AUC**	0.676	0.898
	Accuracy	0.991	0.991
	FScore	0.037	0.049
	False Negative Rate	0.979	0.972
	False Positive Rate	0.001	0.001
	Matthews	0.05	0.068
	Normalized Cost	0.44	0.437
	Precision	0.131	0.181
	True Positive Rate	0.021	0.028
20	**AUC**	0.676	**0.898**
	Accuracy	0.99	0.991
	FScore	0.051	0.128
	False Negative Rate	0.967	0.916
	False Positive Rate	0.002	0.002
	Matthews	0.055	0.146
	Normalized Cost	0.435	0.412
	Precision	0.104	0.269
	True Positive Rate	0.033	0.084
30	**AUC**	0.676	**0.898**
	Accuracy	0.989	0.99
	FScore	0.057	0.178
	False Negative Rate	0.958	0.869
	False Positive Rate	0.003	0.003
	Matthews	0.056	0.187
	Normalized Cost	0.432	0.391
	Precision	0.088	0.279
	True Positive Rate	0.042	0.131

Table 6.3: Experiment 2: daily budget K estimation (III)

K	Metric	AutoEncoder	Copulas	IsolationForest
50	**AUC**	0.612	0.558	0.897
	Accuracy	0.986	0.987	**0.99**
	FScore	0.017	0.07	**0.312**
	False Negative Rate	0.985	0.937	**0.721**
	False Positive Rate	0.006	0.006	**0.004**
	Matthews	0.01	0.064	**0.309**
	Normalized Cost	0.446	0.424	**0.326**
	Precision	0.018	0.081	**0.355**
	True Positive Rate	0.015	0.063	**0.279**
100	**AUC**	0.612	0.558	0.897
	Accuracy	0.98	0.981	**0.985**
	FScore	0.016	0.067	**0.284**
	False Negative Rate	0.979	0.914	**0.634**
	False Positive Rate	0.013	0.012	**0.01**
	Matthews	0.007	0.059	**0.284**
	Normalized Cost	0.446	0.417	**0.29**
	Precision	0.013	0.055	**0.233**
	True Positive Rate	0.021	0.086	**0.366**
200	**AUC**	0.612	0.558	0.897
	Accuracy	0.967	0.968	**0.974**
	FScore	0.015	0.055	**0.223**
	False Negative Rate	0.968	0.888	**0.537**
	False Positive Rate	0.025	0.025	**0.022**
	Matthews	0.004	0.05	**0.251**
	Normalized Cost	0.449	0.412	**0.253**
	Precision	0.01	0.036	**0.147**
	True Positive Rate	0.032	0.112	**0.463**

Table 6.4: Experiment 2: daily budget K estimation (IV)

K	Metric	MatrixDecomposition	Threshold-base
50	**AUC**	0.676	**0.898**
	Accuracy	0.986	0.989
	FScore	0.057	0.229
	False Negative Rate	0.949	0.795
	False Positive Rate	0.006	0.005
	Matthews	0.05	0.225
	Normalized Cost	0.429	0.36
	Precision	0.065	0.261
	True Positive Rate	0.051	0.205
100	**AUC**	0.676	**0.898**
	Accuracy	0.98	0.984
	FScore	0.05	0.237
	False Negative Rate	0.935	0.696
	False Positive Rate	0.012	0.01
	Matthews	0.042	0.236
	Normalized Cost	0.427	0.318
	Precision	0.041	0.195
	True Positive Rate	0.065	0.304
200	**AUC**	0.676	**0.898**
	Accuracy	0.968	0.973
	FScore	0.044	0.197
	False Negative Rate	0.908	0.593
	False Positive Rate	0.025	**0.022**
	Matthews	0.038	0.219
	Normalized Cost	0.421	0.278
	Precision	0.029	0.13
	True Positive Rate	0.092	0.407

6.6 Experiment 3: supervised models comparison

With this test, we compare the *Random Forest* in AMARETTO, with state of the art supervised models, cited in Section 2.2, including *Gradient Boosting* and *Category Boosting*. The *Gradient Boosting* framework [52] is considered one of the best algorithms for classification tasks. *Category Boosting* [53] is an alternative boosting algorithm based on decision trees, that offers computational and efficiency improvements compared to Gradient Boosting frameworks. The experiment is done by running predictions daily. Table 6.5 and Table 6.6 presents the average metrics for each technique.

Table 6.5: Experiment 3: supervised techniques comparison (I)

Metric	RandomForest	CatBoost	DecisionTree
Accuracy	**0.998**	0.998	0.996
Precision	0.899	**0.907**	0.77
True Positive Rate	**0.793**	0.781	0.771
False Positive Rate	0.001	0.001	0.002
False Negative Rate	**0.207**	0.219	0.229
FScore	**0.842**	0.838	0.77
Matthews	**0.843**	0.84	0.768
Normalized Cost	**0.094**	0.099	0.104

Table 6.6: Experiment 3: supervised techniques comparison (II)

Metric	xGradient Boost	SVM	NaiveBayes
Accuracy	0.997	0.994	0.963
Precision	0.879	0.855	0.036
True Positive Rate	0.752	0.3	0.135
False Positive Rate	0.001	**0**	0.03
False Negative Rate	0.248	0.7	0.865
FScore	0.809	0.44	0.056
Matthews	0.811	0.501	0.054
Normalized Cost	0.112	0.314	0.405

Please note that we split our table into 2 parts, for the sake of readability. As shown in Table 6.5 and Table 6.6, the metrics are quite similar between Random Forest, Category Boosting (CatBoost) and Gradient Boosting (xGradient Boost)

models, whilst other supervised methods don't achieve equally good performance. The CatBoost model accomplishes the highest *Precision* although *Random Forest* achieves the highest *TPR* and the lowest *FNR*. Considering also the cost-related metric, we can conclude that Random Forest is more efficient than the other models.

6.7 Experiment 4: detecting new anomalies

The goal of this test is to assess the performance of supervised and unsupervised techniques to detect new anomalous patterns.

Several runs of this experiment are done in order to test each combination of the classes of anomalies existing in the dataset. For every run, a set of classess of anomalies is withheld from the training set is only introduced in the test set for evaluation. During the run, the models are trained using the high-level vectors obtained from the training set that contains the remaining class of anomalies excluding the withheld anomalies. After several iterations of the system (precisely after the 15th day of test), the withheld pattern is introduced in the test data to assess the behaviours of the two models. The results of each run are then averaged on a daily basis. For this test, we considered a daily budget of $K = 5$ in order to compute the same metrics for the two algorithms.

Figure 6.2 and Figure 6.3 demonstrate that *Isolation Forest* performance is consistent, while *Random Forest* exhibits a decay in performance when new anomalies are introduced. The increase of 100% in the FNR proves that *Random Forest* is unable to detect new anomalous patterns introduced in the dataset. On the other hand, the performance of *Isolation Forest* was not negatively affected by the new anomalous pattern introduced.

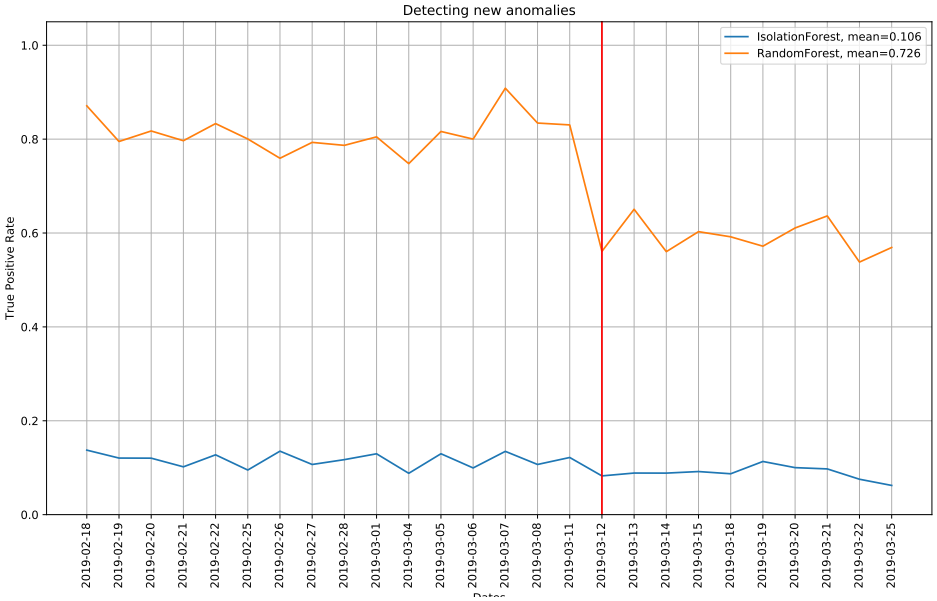


Figure 6.2: Experiment 4: True Positive Rates

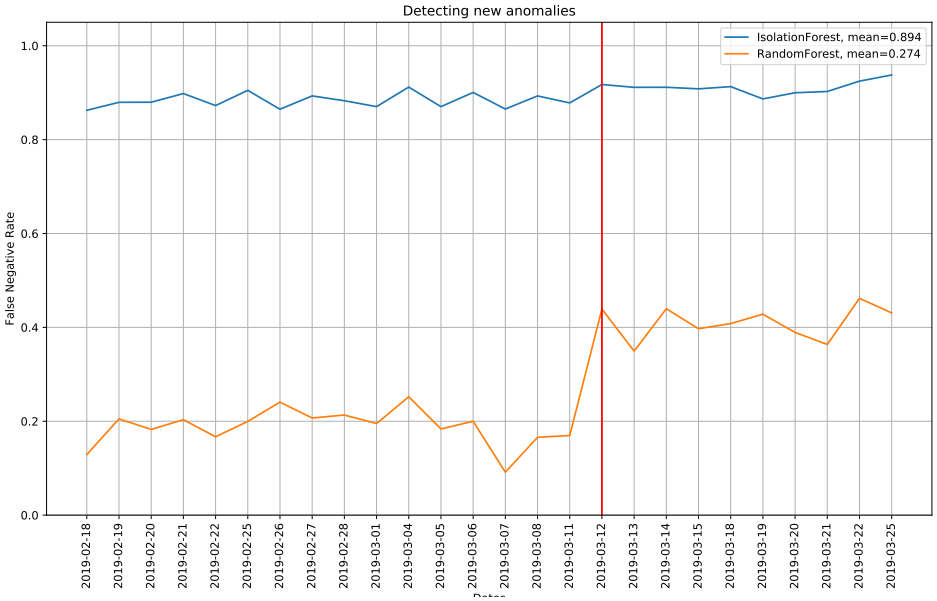


Figure 6.3: Experiment 4: False Negative Rates

6.8 Experiment 5: AMARETTO configurations

The purpose of this experiment is to test which of the strategies are the most suitable for our dataset since our active learning system is customisable with different selection strategies, as explained in Section 4.4. The experiment works as follow: the first day the labelled dataset is empty, therefore the supervised model is not used. After the first day, the labelled dataset contains the samples selected by the *Isolation Forest* which have been reviewed by an analyst. From this point onwards, the entire selection strategy can be employed (first stage, second stage and third stage). The mapping between the approaches adopted in the first and third stage and the names of the configurations are outlined below (and used in experiments described in the next section):

- $first\_stage = SELECT\text{-}TOP$
- $third\_stage = SELECT\text{-}ENTROPY$;

} $Amaretto\_0$

- $first\_stage = SELECT\text{-}DIVERSE$
- $third\_stage = SELECT\text{-}CONFLICT$;

} $Amaretto\_1$

- $first\_stage = SELECT\text{-}TOP$
- $third\_stage = SELECT\text{-}CONFLICT$;

} $Amaretto\_2$

- $first\_stage = SELECT\text{-}DIVERSE$
- $third\_stage = SELECT\text{-}ENTROPY$;

} $Amaretto\_3$

Figure 6.4 and Figure 6.5 show the *average Precision* and the average AUROC of the score generated by *Random Forest*. The performances of the 4 configurations are similar. We decided to focus on the system that employs *SELECT-DIVERSE* and *SELECT-ENTROPY* (*Amaretto_3*, red bar in the plots) because it provides the best *Precision* in the worst-case scenario where the daily budget K is equal to 10.

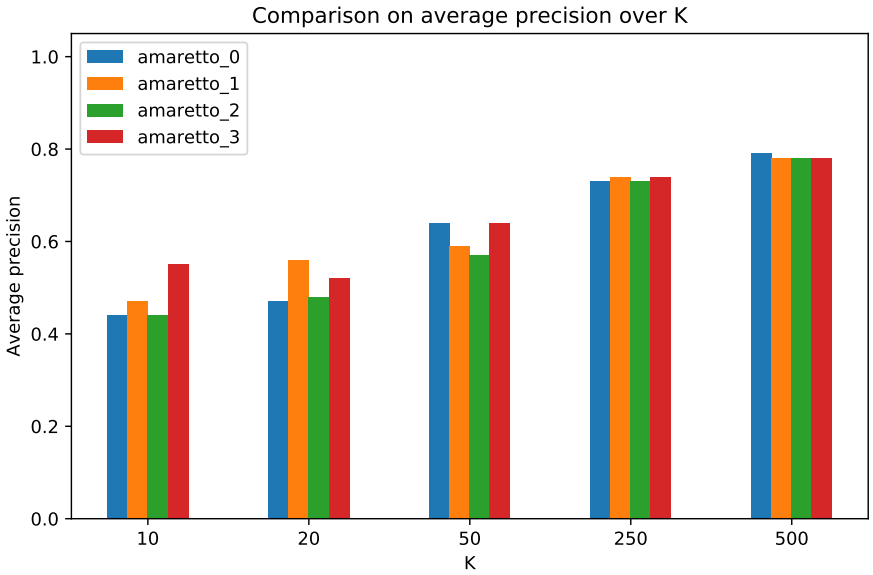


Figure 6.4: Experiment 5: average Precision

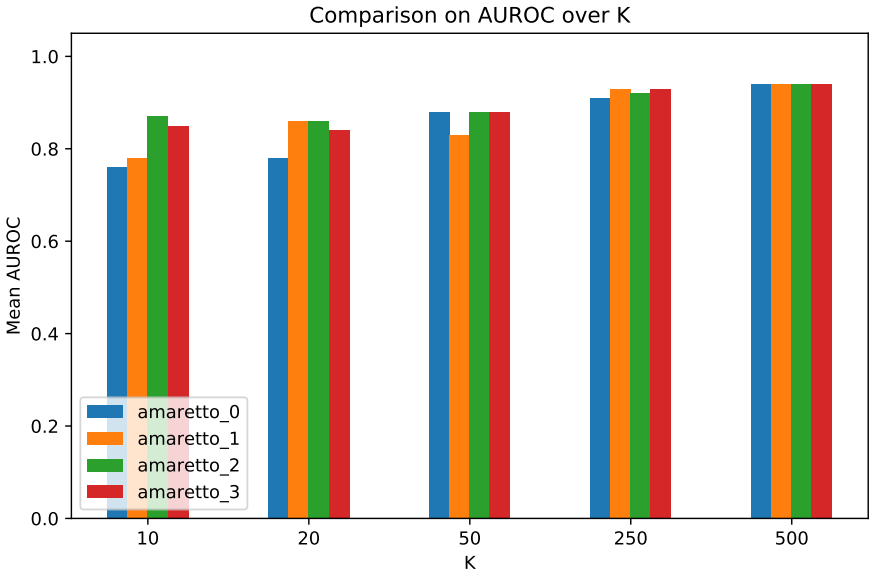


Figure 6.5: Experiment 5: AUROC

6.9 Experiment 6: state of the art comparison

In this final experiment, we compare AMARETTO with the state-of-the-art active learning framework AI^2 [26]. It includes an ensemble of three unsupervised models including a density-based model, using a *Copula-based multivariate* distribution, a matrix decomposition-based model, using a *PCA-based* model, and an *Autoencoder*. The combination of the `anomaly_score` computed by the three models is used to rank the most anomalous high-level vectors for review by an analyst; The feedback collected is then used to train a *Random Forest* that additionally analyse the high-level vectors. The experiment is divided into three parts:

I we compare the frameworks in a static scenario i.e., we collect 10 samples per day over a period of 10 days from each framework and then we use this labelled data to train the *Random Forest* and to predict all the remaining high-level vectors;

II we compared the frameworks in a real-world scenario, comparing the effective support to the daily routine of an analyst and the performances of the frameworks;

III we compare the frameworks in a real-world scenario, comparing the performance taking into account different risk profiles for a financial institution.

The purpose of the first part of the experiment is to verify the performance of the frameworks with a minimum amount of training data. During this part, we also assess the active learning inner modules, i.e., the components of the framework in charge of computing the `anomaly_score` and selecting the samples to be shown to the analyst.

Table 6.7: Experiment 6 (I): classification report

	Amaretto	**AI2**
Accuracy	0.9947	**0.9951**
Precision	0.7488	**0.9813**
True Positive Rate	**0.5159**	0.3896
False Positive Rate	0.0014	**0.0001**
False Negative Rate	**0.4841**	0.6104
FScore	**0.6109**	0.5578
Matthes	**0.6191**	0.6167
Normalized Cost	**0.2168**	0.2724

For the first 10 days existing in the test set, only the inner module are employed with a minimum daily budget ($K = 10$), collecting 100 samples. Afterwards, this labelled dataset is used to train a *Random Forest*. Subsequently, the trained *Random Forest* model computes the *probability score* and the *prediction* for all the remaining high-level vectors. Figure 6.6 and Figure 6.7 show the comparison of our system against AI^2 using the *probability score*. As exhibited by the ROC curve plotted in Figure 6.6, AMARETTO achieves an *AUROC* of 0.94 better than 0.88 obtained by AI^2. Furthermore, as shown in the *Precision/Recall* curve in Figure 6.7, AMARETTO reaches an *average Precision* of 0.62 that is 31% better than AI^2. Table 6.7 shows the classification report obtained using the *prediction* computed by the *Random Forest*. In spite of the better Precision and FPR obtained by AI^2, AMARETTO achieves comparable performances especially considering that TPR and cost.

In the second part of the experiment, we estimate, in a real-world scenario, the effective support provided by the framework and the practical decrease of the workload of the analyst. Initially, only the unsupervised techniques can be employed since no feedback was collected. After the first day, the *Random Forest* starts to work alongside the unsupervised models in the active learning loop and the prediction phase. For this test, we consider the worst-case scenario with a minimum daily budget of ($K = 10$). Figure 6.8 shows the average precision computed using the *probability score*. AMARETTO doubled its *Precision* in approximately 10 days, i.e with a dataset of 100 high-level vectors, constantly increasing its performance. During the tests, AMARETTO reaches a maximum *average Precision* of 0.78, while AI^2 0.57. As shown in Figure 6.9, AMARETTO achieves better performances also considering the *AUROC*. As in the previous test, the maximum *AUROC* is 0.94 and the *mean AUROC* is 0.847, improving AI^2 one by 14% circa.

In the last part, we test the frameworks considering different risk profiles that a financial institution can adopt. This is done considering different threshold values for the *probability score* corresponding to different use cases. For example, a lower threshold could be used where high financial risk is estimated, this way, more transactions will be considered as a candidate for review, hence reducing the *False Negatives* but increasing *False Positives* too. As shown in Figure 6.10, Figure 6.11, Figure 6.12, and Figure 6.13, the AMARETTO outperforms the AI^2 across all thresholds: in the high-risk use case, AMARETTO achieves a *TPR* of 0.428, while in the high-risk use case a *TPR* of 0.596 that represents an improvement of circa 50% w.r.t AI^2. AI^2 achieves a better *FPR* than AMARETTO only in a low-risk case. On the other hand, AMARETTO achieves a higher *TPR* and a lower *FNR* and a lower *Cost*, balancing the overall performance. Another import point is that in every scenario, the *Cost* of the AMARETTO system is lower than the *Cost* of AI^2.

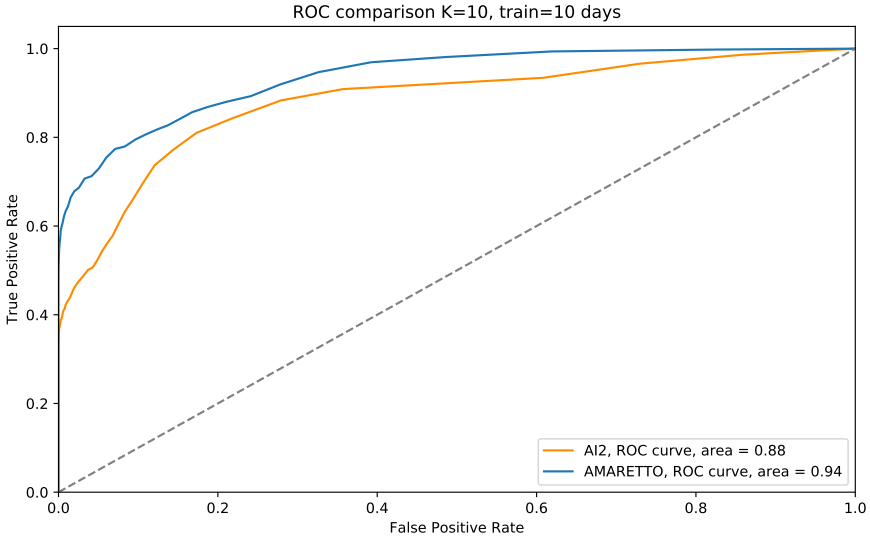


Figure 6.6: Experiment 6 (I): ROC

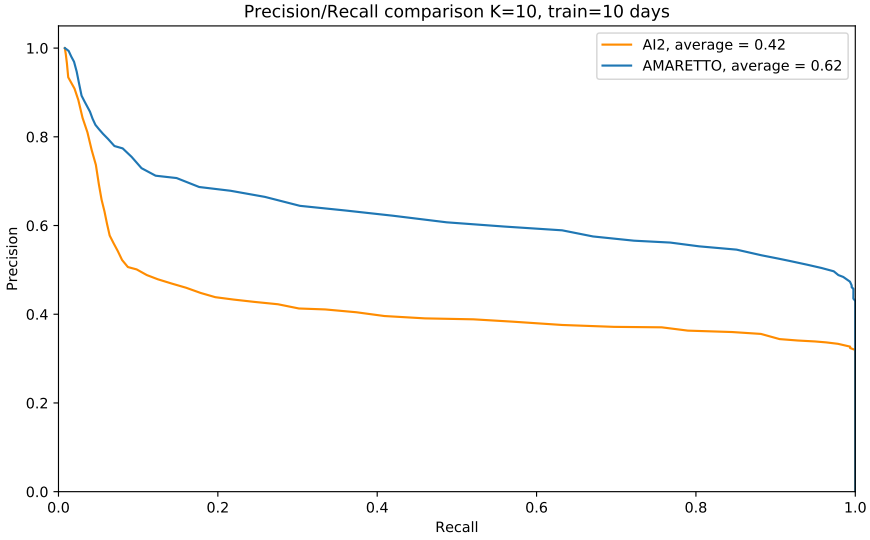


Figure 6.7: Experiment 6 (I): Precision/Recall curve

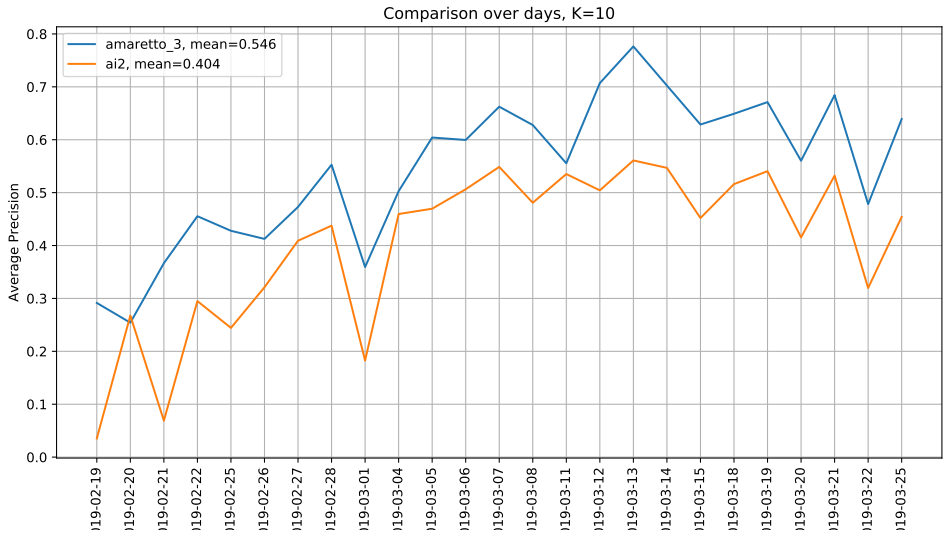


Figure 6.8: Experiment 6 (II): average Precision

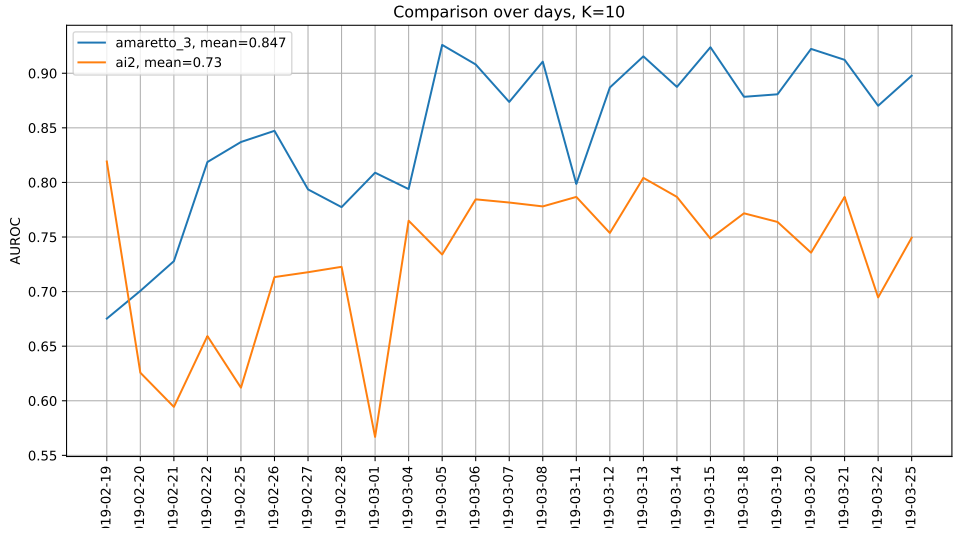


Figure 6.9: Experiment 6 (II): average AUROC

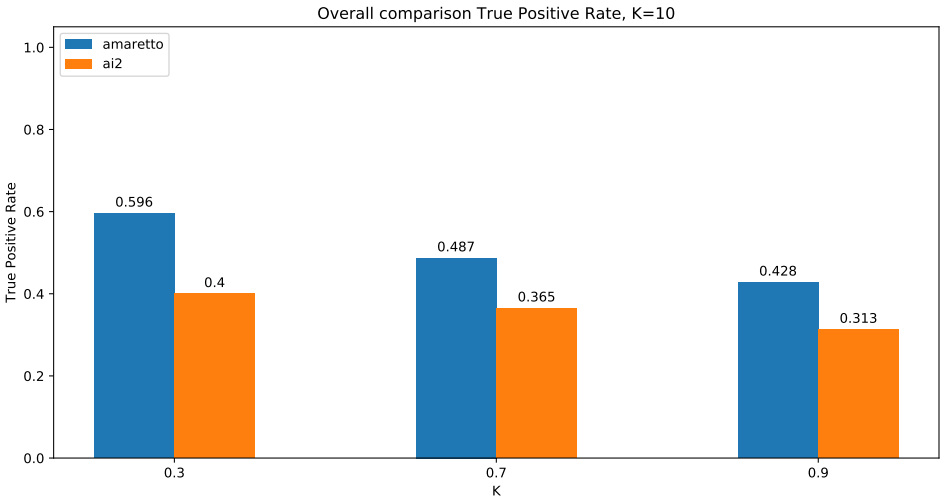


Figure 6.10: Experiment 6 (III): TPR over thresholds

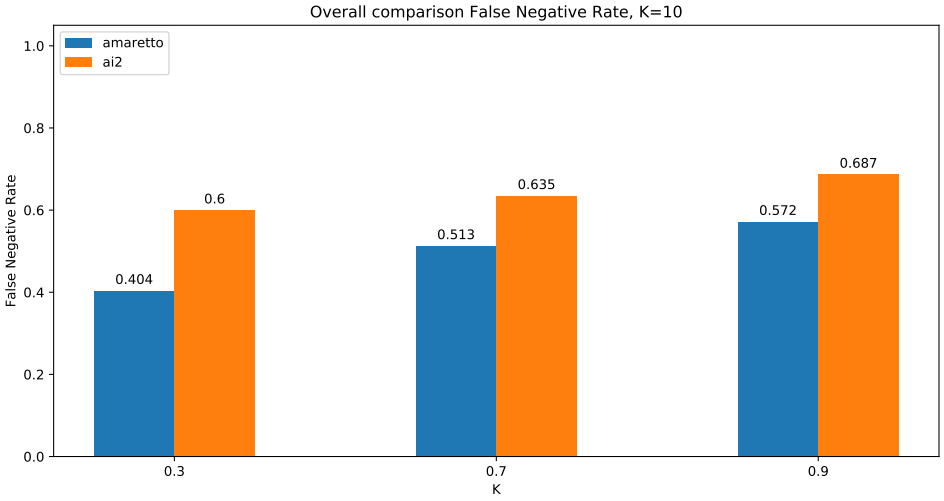


Figure 6.11: Experiment 6 (III): FNR over thresholds

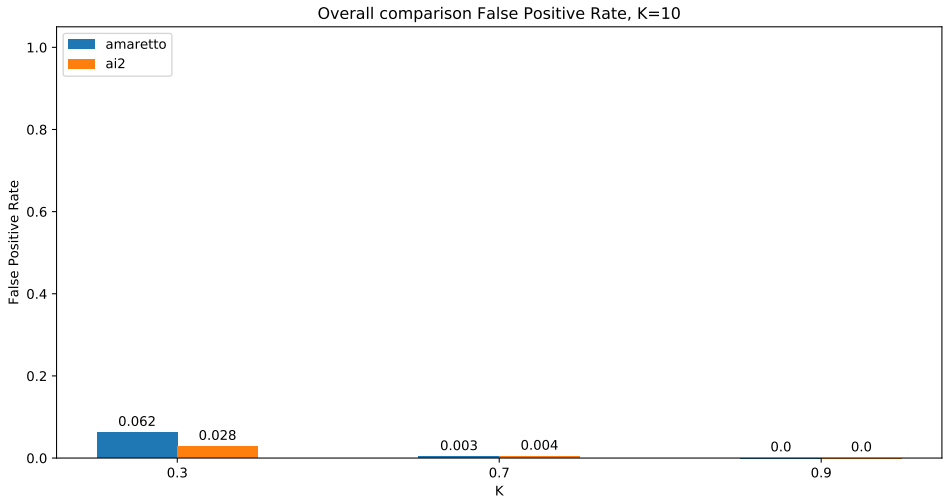


Figure 6.12: Experiment 6 (III): FPR over thresholds

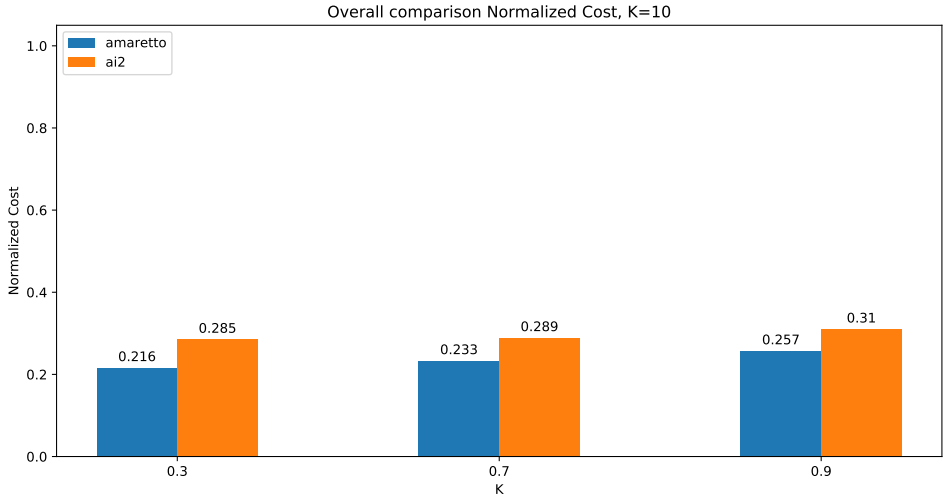


Figure 6.13: Experiment 6 (III): Cost over thresholds

Chapter 7

Limitations and Future Work

The first limitation encountered in this research work is the lack of an intuitive explanation for the `anomaly_score` returned by the two models which are used to rank the high-level vectors. The two algorithms are used in a black-box fashion. We know that both techniques are based on rule-based trees whose prediction can be explained following the path that led to the given classification. However, with a *Random Forest* and *Isolation Forest*, we are using the average result of several trees, so the resulting probability does not provide us with any useful insight about the choice. For this reason, the final `anomaly_score` can not be used to help the human analyst to better understand the result.

Another limitation of our work is the dataset we worked on. Even though NapierAI experts synthetically forged it on a real-world dataset, is very limited in terms of timespan, containing only 60 days of transactions. A wider dataset would have allowed us to perform a deep analysis of the system's evolution over time, especially it would have been possible to implement seasonal models or models for a specific anomalous pattern. Moreover, the dataset contains only a few patterns highlighted by the FATF [12]. It could be interesting to see how AMARETTO works on more complex scenarios. Despite the important advantage of being conducted in collaboration with NapierAI domain experts, exploiting their knowledge to target the evaluation on the detection of specific money laundering patterns, the experimental validation does not include a further evaluation on a real-world dataset.

For what concerns possible future works, there are many interesting directions in which this work can be improved:

- as part of this work, we assumed that the analyst always provided correct labels when reviewing the data; further research should be conducted to assess the

impact of incorrect labelling to the system and ensure that the models can accommodate for such errors;

- alternative unsupervised and supervised models should be tested. In this work, we didn't consider graph-based models or deep learning models. For example, LSTM neural networks could be employed due to the temporal correlations in money-laundering patterns;
- a library implementing explainability processes should be adopted in AMARETTO. For example, `SHAP` [54] is one of the newest and most interesting approaches for explaining models' output;
- finally, evaluating the tool on the detection of other kinds of financial frauds, like credit card fraud detection, to prove its flexibility.

Chapter 8

Conclusions

In this work, we present an active learning system for anomaly detection applied to assess money-laundering risk in capital markets. AMARETTO comprises of an unsupervised model for detecting known and unknown anomalous patterns, 4 distinct strategies to optimally sample the data for a subject matter expert to review and feed into a supervised learning model to continuously improve the performance of the system. AMARETTO was able to process over 29 million transactions, extract aggregated features highlighting customer's behavioural patterns over time to detect unusual correlations.

Finally, we present the experimental evaluation conducted on a synthetic dataset that was generated from a real-world scenario that resembles typical genuine and potential money laundering patterns. Firstly, we compare unsupervised techniques, commonly used in anomaly detection tasks and in state of the art solutions, and we demonstrate that Isolation Forest is the best algorithm. Then, we compare the supervised techniques and we assess that Random Forest outperforms the other techniques. Subsequently, we prove the contributions made by an unsupervised model to an anomaly detection made up of only supervised models. In the end, we conducted experiments with the aim of confirming the robustness of our design in a real-world scenario, demonstrating the best selection strategies between the ones proposed and showing that AMARETTO outperforms the state of the art solutions by improving both the detection rate and the precision by 25% and achieving an overall detection rate of 0.6 and an AUROC of 0.94.

The experimental validation proves that our system was able to achieve state of the art performance within a short timeframe, with minimal manual input from an analyst.

Bibliography

[1] U. N. O. on drugs and crime, "Estimating illicit financial flows resulting from drug trafficking and other transnational organized crimes." `https://www.unodc.org/documents/data-and-analysis/Studies/Illicit_financial_flows_2011_web.pdf`, 2011.

[2] FCA, "Financial crime: a guide for firms." `https://www.handbook.fca.org.uk/handbook/FCG.pdf`, 2016.

[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, 07 2009.

[4] V. Kumar, "Parallel and distributed computing for cybersecurity," *IEEE Distributed Systems Online*, vol. 6, no. 10, 2005.

[5] C. Spence, L. Parra, and P. Sajda, "Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model," *Proceedings of the Workshop on Mathematical Methods in Biomedical Image Analysis*, 10 2001.

[6] M. Carminati, R. Caron, F. Maggi, I. Epifani, and S. Zanero, "Banksealer: A decision support system for online banking fraud analysis and investigation," *Computers & Security*, vol. 53, pp. 175 – 186, 2015.

[7] N.-A. Le-Khac and T. Kechadi, "Application of data mining for anti-money laundering detection: A case study," pp. 577–584, 12 2010.

[8] N.-A. Le-Khac, S. Markos, and T. Kechadi, "A data mining-based solution for detecting suspicious money laundering cases in an investment bank," pp. 235–240, 01 2010.

[9] S. Viaene, R. Derrig, B. Baesens, and G. Dedene, "A comparison of state-of-the-art classification techniques for expert automobile insurance claim fraud detection," *Journal of Risk and Insurance*, vol. 69, pp. 373 – 421, 09 2002.

[10] B. Hoogs, T. Kiehl, C. Lacomb, and D. Senturk, "A genetic algorithm approach to detecting temporal patterns indicative of financial statement fraud," *Int. Syst. in Accounting, Finance and Management*, vol. 15, pp. 41–56, 06 2007.

[11] R. J. Bolton and D. J. Hand, "Statistical fraud detection: A review," *Statistical Science*, vol. 17, no. 3, pp. 235–249, 2002.

[12] T. F. A. T. F. (FATF), "Fatf report - professional money laundering." `http://www.fatf-gafi.org/media/fatf/documents/Professional-Money-Laundering.pdf`, 2018.

[13] T. L. Vic Barnett, *Outliers in Statistical Data, 3rd Edition*. Wiley, 1994.

[14] E. Ngai, Y. Hu, Y. Wong, Y. Chen, and X. Sun, "The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature," *Decision Support Systems*, vol. 50, pp. 559–569, 02 2011.

[15] J. West and M. Bhattacharya, "Intelligent financial fraud detection: A comprehensive review," *Computers & Security*, vol. 57, pp. 47 – 66, 2016.

[16] D. Yue, X. Wu, Y. Wang, Y. Li, and C. Chu, "A review of data mining-based financial fraud detection research," in *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 5519–5522, Sep. 2007.

[17] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets.," vol. 29, pp. 427–438, 06 2000.

[18] G. Williams, R. Baxter, H. He, S. Hawkins, and L. Gu, "A comparative study of rnn for outlier detection in data mining," in *in ICDM*, p. 709, 2002.

[19] M. Goldstein and A. Dengel, "Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm," *KI-2012: Poster and Demo Track*, pp. 59–63, 2012.

[20] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International*

Conference on Knowledge Discovery and Data Mining, KDD'96, pp. 226–231, AAAI Press, 1996.

[21] R. N. C. Ebberth L. Paula, Marcelo Ladeira and T. Marzagão, "Deep learning anomaly detection as support fraud investigation in brazilian exports and anti-money laundering," in *15th IEEE International Conference on Machine Learning and Applications*, 2016.

[22] A. D. Pozzolo and G. Bontempi, "Adaptive machine learning for credit card fraud detection," in *Adaptive Machine Learning for Credit Card Fraud Detection*, 2015.

[23] 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. On quantitative methods for detection of financial fraud.

[24] T. Senator, H. Goldberg, J. Wooton, M. Cottini, A. Khan, C. Klinger, W. Llamas, M. Marrone, and R. Wong, "The financial crimes enforcement network ai system (fais) identifying potential money laundering from reports of large cash transactions.," *AI Magazine*, vol. 16, pp. 21–39, 12 1995.

[25] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, "Toward supervised anomaly detection," *CoRR*, vol. abs/1401.6424, 2014.

[26] K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li, "ai^2: Training a big data machine to defend," in *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, pp. 49–54, April 2016.

[27] S. Das, M. R. Islam, N. K. Jayakodi, and J. R. Doppa, "Active anomaly detection via ensembles: Insights, algorithms, and interpretability," 2019.

[28] L. Akoglu, H. Tong, and D. Koutra, "Graph-based anomaly detection and description: A survey," *Data Mining and Knowledge Discovery*, vol. 29, 04 2014.

[29] W. Eberle and L. Holder, "Anomaly detection in data represented as graphs," *Intell. Data Anal.*, vol. 11, pp. 663–689, Dec. 2007.

[30] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, (New York, NY, USA), pp. 631–636, ACM, 2003.

[31] M. Bulloni, "Fraudcleaner : an unsupervised ensemble approach for money laundering and financial fraud detection." `https://www.politesi.polimi.it/handle/10589/140286`, Apr. 2018.

[32] A. Biondani, "Fraudhunter : a supervised fraud detection tool for internet banking transactions." `https://www.politesi.polimi.it/handle/10589/123569`, July 2016.

[33] M. Belhaj, "Fraudkiller: an online fraud detection system for digital marketplaces." `https://www.politesi.polimi.it/handle/10589/135601`, July 2017.

[34] A. Dignani, "Fraudsdigger : an active learning tool for online banking fraud detection." `https://www.politesi.polimi.it/handle/10589/144814`, Dec. 2018.

[35] FATF, "Guidance for a risk-based approach: securities sector." `https://www.fatf-gafi.org/media/fatf/documents/recommendations/pdfs/RBA-Securities-Sector.pdf`, 2018.

[36] H. S. Seung, H. Sompolinsky, and N. Tishby, "Statistical mechanics of learning from examples," *Phys. Rev. A*, vol. 45, pp. 6056–6091, Apr 1992.

[37] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 6, pp. 3:1–3:39, Mar. 2012.

[38] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct 2001.

[39] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, Mar. 1986.

[40] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Advances in Knowledge Discovery and Data Mining* (J. Pei, V. S. Tseng, L. Cao, H. Motoda, and G. Xu, eds.), (Berlin, Heidelberg), pp. 160–172, Springer Berlin Heidelberg, 2013.

[41] L. McInnes and J. Healy, "Accelerated hierarchical density based clustering," *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, Nov 2017.

[42] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 2951–2959, Curran Associates, Inc., 2012.

[43] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[45] E. Jones, T. Oliphant, and P. Peterson, "{SciPy}: open source scientific tools for {Python}," 2014.

[46] I. Kaastra and M. Boyd, "Designing a neural network for forecasting financial and economic time series," *Neurocomputing*, vol. 10, no. 3, pp. 215 – 236, 1996. Financial Applications, Part II.

[47] F. Chollet *et al.*, "Keras: The python deep learning library," *Astrophysics Source Code Library*, 2018.

[48] W. McKinney, "pandas: a foundational python library for data analysis and statistics," *Python for High Performance and Scientific Computing*, pp. 1–9, 2011.

[49] C. Whitrow, D. Hand, P. Juszczak, D. Weston, and N. Adams, "Transaction aggregation as a strategy for credit card fraud detection," *Data Mining and Knowledge Discovery*, vol. 18, pp. 30–55, 02 2009.

[50] D. Kingma and M. Welling, "Auto-encoding variational bayes," 12 2014.

[51] M. ling Shyu, S. ching Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," in *in Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop, in conjunction with the Third IEEE International Conference on Data Mining (ICDM'03*, pp. 172–179, 2003.

[52] T. Chen and C. Guestrin, "Xgboost," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016.

[53] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," 2017.

[54] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," 2017.