Politecnico di Torino

Master's Degree in Computer Engineering

A.Y. 2024/2025

Graduation Session: December 2025

Enhancing Financial Crime Identification in Traditional Banking through Semi-Supervised Anomaly Detection

Supervisor:

Prof. Flavio Giobergia

Co-supervisors:

Prof. Danilo Giordano

Dr. Giordano Paoletti

Dr. Claudio Savelli

Candidate:

Giuseppe Marinacci

Abstract

Financial crimes pose significant risks to financial institutions, national economies, and society as a whole. The major threats in this domain are addressed through specific regulatory frameworks, such as Anti Money Laundering (AML), Combating the Financing of Terrorism (CFT) and opposing the proliferation of weapons of mass destruction. To mitigate these risks, supervisory authorities impose substantial penalties on institutions that do not adopt effective measures of prevention. Consequently, banks are required to comply with strict regulations designed to detect and prevent such crimes. Although these tasks have traditionally been handled by rule-based Transaction Monitoring (TxM) systems, recent advances in Machine Learning have introduced new paradigms to address this complex challenge.

This thesis presents a real-world case study on the adoption of unsupervised and semi-supervised Machine Learning techniques for anomaly detection in the context of a traditional banking system, a sector typically characterized by strong conservatism. In the next pages, the "Multicriteria Anomaly Detection" (MAD) project is described in detail: using approximately 3.3 billion anonymized transactions collected over 12 months by an Italian financial institution, a Machine Learning pipeline for TxM was developed, optimized, and deployed in production.

The work spans the entire process: from the presentation of the datasets, to the extraction of 98 aggregated features describing each account's monthly transactional behavior, to the training and hyperparameter tuning of a semi-supervised AutoEncoder, and finally to the evaluation of its performance with dedicated metrics. Comparative analyses were also carried out against other unsupervised anomaly detection models, specifically One-Class Support Vector Machine and Isolation Forest. This exploration is particularly relevant given the increasing demand for more effective approaches to counter financial crimes and highlights how the application of Machine Learning can contribute to the evolution of monitoring systems in this critical sector.

Acknowledgements

I would like to express my sincere gratitude to Prof. Flavio Giobergia for his expert guidance throughout the development of this thesis. His contribution has been essential not only for the successful completion of this work, but also for my broader academic and professional growth.

I am equally grateful to Prof. Danilo Giordano, Dr. Giordano Paoletti and Dr. Claudio Savelli for their remarkable dedication and constant support. Their daily supervision and patient assistance in facing every challenge were invaluable.

Finally, I would like to thank Alessandro La Ciura, who has shared this entire journey with me from day one, for his ability to lighten the atmosphere and make the whole experience far more enjoyable.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Financial Action Task Force (FATF) is an intergovernmental body created by the Paris G7 in 1989 with the aim of establishing global standards and strategies to combat money laundering of illicit origin ("Anti Money Laundering" or AML), terrorism financing ("Combating the Financing of Terrorism" or CFT) and the financing of the proliferation of weapons of mass destruction. In summary, the FATF is the world's leading and standard-setting body in the fight against financial crime, aiming to protect the global financial system.

The organization issues the 40 Recommendations[1]: international principles and measures that are used by countries to prevent and combat these crimes. As of September 2025, the 20th recommendation states that

"if a financial institution suspects or has reasonable grounds to suspect that funds are the proceeds of a criminal activity, or are related to terrorist financing, it should be required, by law, to report promptly its suspicions to the Financial Intelligence Unit (FIU)"[1].

Financial institutions rely on Transaction Monitoring (TxM) systems to comply with this regulation: these systems process hundreds of millions of transactions to model accounts' behaviors in search of the suspicious ones. TxM systems are often deployed as a set of deterministic rules, causing alerts when an account triggers one of them. However, in recent years, a new approach has been applied to make these systems more accurate and, most importantly, able to evolve in time, in order to constantly follow the evolution of criminal behavior: applying Machine Learning to TxM.

This thesis presents a real-world case study on using semi-supervised Machine Learning techniques to detect anomalies for AML. In the next pages, the "Multi-modal Anomaly Detection" (MAD) project will be presented: using approximately 3.3 billion anonymized bank transactions registered over 12 months one of the largest Italian financial institutions, a Machine Learning pipeline for TxM was developed, tuned, and deployed in production.

1.1 Subject of study

The project presented in this thesis is the follow-up of a previous work, developed between March and September 2024 by Politecnico di Torino and fan italian financial institution, and its aim is to expand the scope of the former one. By introducing MAD 2024 [2], some additional background on TxM and financial regulations will be provided, as well as some common vocabulary.

A customer of a financial institution is defined as an entity that performs interactions with the mentioned institution. A customer can be a physical person or a legal entity, and can interact with the institution through an account, like a savings or a checking account. Each account is identified by an account ID, for example, the American Bankers Association ("ABA") routing transit number in the USA, or the International Bank Account Number ("IBAN") in Europe. A single customer can operate several accounts for distinct purposes, while a single account can also be shared among multiple customers.

Each account enables the execution of transactions. These can be performed directly between the bank and the customer (for instance, withdrawing funds from an ATM) or between two distinct parties (such as a bank transfer). If a set of transactions is considered potentially malicious by a financial institution, it is mandatory that the institution report the suspicious activities to FIUs, by means of a so-called "Suspicious Activity Report" (SAR). SARs are required to contain information on the customer who performed the suspicious activity (the "subject" of the SAR) and the set of financial operations that characterize the suspicious activity (the "object" of the SAR).

As mentioned above, financial institutions fulfill this requirement by implementing TxM systems: a set of rules that analyze hundreds of millions of transactions to catch the potentially suspicious accounts and customers. Since the system is not required to work in real-time, TxM systems typically run every month.

The alerts generated by the automated part are then subject to several layers of investigation, carried out by different human operators within what is known as the competence center. These centers usually function on two tiers: L1 and L2. At the L1 stage, a preliminary assessment is conducted to filter out clearly irrelevant alerts. Those considered potentially significant are escalated to L2, where further in-depth investigations are performed. The result of this process is that the alert is either classified as a false positive (meaning it is judged to be not suspicious) or as sufficiently concerning to justify the creation of a SAR, which is then forwarded to the FIU.

The FIU proceeds with its own investigation and takes appropriate actions, which may include involving law enforcement. Typically, the FIU does not provide explicit feedback on the investigation's final outcome, but only indicates whether the SAR has been archived (deemed unimportant) or left open. In this thesis, non-archived SARs are evaluated as true positives.

1.2 Formal problem formulation

The set of all existing customers is referred to as $C = \{c_1, c_2, \ldots\}$, and the set of all existing accounts is referred to as $A = \{a_1, a_2, \ldots\}$. By representing as $\mathcal{P}(X)$ the power set of X, the relationship between customers and accounts can be modeled with two functions:

- $\mathit{owns}(\cdot) : C \to \mathcal{P}(A) \setminus \emptyset$, indicating that a customer is the owner of an account (that is, the owner uses the account to perform transactions);
- $\mathit{isOwned}(\cdot) : A \to \mathcal{P}(C) \setminus \emptyset$, indicating that an account is owned by a customer.

In both cases, a power set must be used, as a customer can own multiple accounts and an account can be owned by more than one customer, in which case a primary owner is identified for the account (this relationship can be referred to as $\mathit{primaryOwns}(\cdot) : A \to C$.

At the beginning of each month, each account is also associated to the available amount of money, the so called *balance*: $\mathit{balance}(\cdot) : A \to \mathbb{R}$.

Each customer is associated with a customer type, representing a general description of the customer, which is determined during the Know Your Customer (KYC) due diligence phase. The set $\mathcal{S}$ of known customer types is made only of two possible values: *Physical Person* and *Legal Entity.* A customer is associated to its type via $\mathit{customerType}(\cdot) : C \to S$.

Transactions can also be divided into different categories: wire transfers, cash (such as deposits and withdrawals) or none of these two. Cash transactions are usually performed between one customer and the financial institution, while wire transfers generally involve two customers as parties, with the institution being just and intermediary. However, for this thesis, the counterpart of a wire transfer was not made available.

Also a set R of reasons for each transaction is provided: it contains a description of the reason motivating a transaction.

Thus, each transaction is described by the 4-tuple (a, m, t, d), where $a \in A$ is the account performing the operations, $m \in \mathbb{R}$ is the amount of the operation (positive if the amount is deposited on the account, negative otherwise), $t \in R$ is the reason and d represents the timestamp (that is, date and time) of the operation. Further details about accounts, customers and transactions descriptions will be provided in the section dedicated to the description of the datasets.

In this thesis, reference to specific attributes of a transaction will be made by means of functions. For instance, $a(t)$ will be used to refer to the account of transaction t, $m(t)$ will be the amount associated with that transaction, and so on. Furthermore, T_{CASH} and T_{WIRE} will refer to the sets of all CASH and WIRE transactions processed by the financial institution, respectively.

To keep the results consistent with the current regulations, as well as to lighten the computational load by reducing the volumes of data, TxM is performed over batches of data belonging to a specific time span. Specifically, a single month will be adopted as time span. Thus, we can define a window of time, delimited by a beginning d_{begin} and an end d_{end} timestamp, and then consider the transactions occurring within that time frame, for instance: $T_{WIRE}|_{d_{begin}}^{d_{end}} = \{t \in T_{WIRE} \mid d_{\text{begin}} \leq d(t) < d_{\text{end}}\}$. Finally, the set of transactions performed by a specific account $a = \alpha$ will be referred to as T_α; so, for instance, the set of cash transactions performed by user α will be: $T_{CASH,\alpha} = \{t \in T_{CASH},\ a(t) = \alpha\}$.

Having defined this notation, the goal of the work will be to produce a set of subjects of interest, A_{pred}, i.e. a set of accounts which behavior seems suspicious and that are consequently selected to undergo further evaluation by the competence center and, eventually, become part of a SAR.

For the semi-supervised part, a ground truth will be used, consisting of past cases that have been reported as suspicious by either the L1 or L2 stages or that have been inserted into a forwarded SAR. As mentioned, further details about the ground truth will be given in the datasets' description section.

1.3 Machine Learning approaches to AML

The use of Machine Learning methods for identifying suspicious activities is extensively documented in the literature. Several studies have investigated different strategies aimed at improving the performance and reliability of anti-money laundering (AML) systems.

However, the fast-paced evolution of money laundering methods and the difficulty of handling incomplete and continuously changing data in real time, often makes it challenging for existing models to remain effective. With the clear goal of obtaining a system that is proactive (rather than just reactive) in addressing emerging threats, MAD 2024[2] adopts an unsupervised approach.

In the AML domain, unsupervised learning techniques are primarily employed for anomaly detection and clustering. Methods such as Isolation Forest, One-Class SVM, and Local Outlier Factor aim to identify instances that deviate from the general data distribution. Nevertheless, one of the main challenges is that anomalies do not always correspond to suspicious or fraudulent behaviors, which often leads to high false positive rates. An additional approach that has gained attention in this context is the use of autoencoder-based models, which offer a flexible and scalable alternative for uncovering hidden irregularities, potentially enabling the detection of future anomalies that differ from those currently observed.

Another critical challenge lies in the scale of real-world data, where the volume often reaches billions of transactions, yet only a tiny fraction (typically around 0.01%) turns out to be malicious. This scenario underscores the necessity for robust and scalable approaches that can effectively identify rare events within highly complex and realistic environments.

In the following chapter, a detailed overview of the machine learning models that form the foundation of MAD 2024 is presented. These models constitute the methodological basis explored and analyzed in this thesis. By revisiting and applying the same approaches, this work aims not only to reproduce their effectiveness, but also to critically assess their applicability, limitations, and potential for improvement.

Chapter 2

Related works

From this point onward, the focus will be on the unsupervised learning techniques already adopted in MAD 2024 and further explored in this project. These methods have proven to be effective and, most importantly, feasible in real-world situations, where scalability and computational efficiency are of top priority. These traits will be essential to ensure seamless integration and full functionality in production deployments.

The following sections will show the aforementioned ML models as well as some supervised models that will exploit the available labeled data to improve the performance of the TxM system.

2.1 One-Class Support Vector Machine

Support Vector Machines (SVMs) are a highly successful class of supervised learning algorithms for classification and regression. The central idea of SVMs is to find the optimal hyperplane that can maximally separate data points belonging to different classes. This principle of maximization of the margins provides strong generalization capabilities, making SVMs particularly effective in spaces of high dimensionality and in cases where the number of features exceeds the number of samples. Over the years, SVMs have been successfully applied in numerous applications from image classification to text classification and bioinformatics.

Based on the idea of basic SVMs, the One-Class SVM (OC-SVM) was introduced as an extension to novelty detection and anomaly detection problems. Unlike traditional SVMs that require labeled data from multiple classes, the OC-SVM is trained only on a single class of data and learns a decision boundary that encloses the majority of the training instances. New samples can then be classified as either belonging to the learned distribution or as outliers. This approach has been shown to perform well in cases where negative examples are scarce or difficult to obtain, such as fraud detection, network intrusion detection, and fault diagnosis.

The One-Class SVM, following the formulation of Schölkopf et al.[3], frames novelty or anomaly detection as support estimation: given samples from an unknown distribution P, its goal is to estimate a region S in the input space such that the probability that a new point falls outside it is less than a predefined threshold ν. The method to approach this problem is to try to estimate a function f which is positive on S and negative on the complement. In other words, OC-SVM is an algorithm which computes a binary function which is supposed to find the regions of the input space where the probability density lives (its support), i.e. a function such that most of the data will be found in the region where the function is non-zero.

The One-Class SVM problem is defined as follows. It takes some unlabeled data $x_1, x_2 \dots x_l \in X$ and a feature map $\Phi(x)$ which maps $X \to F$, that is, a function that maps X into a dot product space F such that the dot product in the image of Φ can be computed by evaluating a simpler kernel:

$$k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$$

The goal is to develop an algorithm which returns a function f that takes the value $+1$ in a "small" region capturing most of the data points, and -1 elsewhere. In other words, the strategy is to seek an hyperplane that separates the mapped data from the origin with maximum margin, yielding a decision function that is positive on the estimated support and negative outside it. This turns the problem into a large-margin, kernelizable optimization well-suited to high-dimensional settings.

In its primal form, the determination of the f function is a problem of minimizing the norm of the weight vector $w = \sum_i \alpha_\mathbf{i} \Phi(\mathbf{x_i})$ while allowing slack for violations, controlled by a parameter $\nu \in (0, 1]$. The canonical dual yields a sparse kernel expansion:

$$f(x) = \sum_{i=1}^{\ell} \alpha_i \, k(x_i, x) - \rho$$

where ρ represents the bias term. This problem is subject to the constraints $0 \leq \alpha_i < \frac{1}{\nu l}$ and $\sum_i \alpha_\mathbf{i} = 1$. Points with $f(x) \geq 0$ are considered inliers while those with $f(x) < 0$ are outliers. The threshold ρ is recovered from any "free" support vector with $0 < \alpha_i < \frac{1}{\nu l}$ by enforcing $f(x_i) = 0$.

From a practical perspective, OC-SVM is attractive because (1) it requires only positive (normal) data at training time, (2) it inherits SVMs' robustness in high dimensions via margin control, and (3) it presents two easily interpretable hyperparameters: ν (expected outlier rate, or model sparsity) and the kernel scale (e.g., RBF width), which shapes the granularity of the support. Proper feature scaling and a simple grid over ν and the kernel width are typically enough for effective models.

2.2 Isolation Forest

Decision Trees represent one of the most fundamental and widely used methods in machine learning, offering a simple and powerful way to model decision-making processes. Their hierarchical structure, based on a recursive partitioning of the feature space, makes them extremely intuitive to human interpretation and very efficient for classification or regression tasks. Thanks to their transparency, Decision Trees have often been employed as both standalone predictive models and as building blocks for more advanced methods.

With this assumption in mind, the Isolation Forest algorithm extends the idea of tree-like structures to the domain of anomaly detection. Instead of modeling normal data distributions in an explicit manner, Isolation Forest isolates anomalies by randomly partitioning the data space, based on the intuition that anomalous points are easier to separate than normal points. This tree-based ensemble approach combines the interpretability of decision trees with the robustness and scalability required by high-dimensional datasets, making it a widely adopted method in the field of unsupervised anomaly detection.

Here, a brief description of the Isolation Forest (iForest) algorithm is proposed, following its original formulation by Liu et al.[4].

The core intuition behind Isolation Forest is that anomalies are rare and significantly different from the other points. Unlike traditional anomaly detection techniques that attempt to create a profile of normal instances, this method proposes a different approach: it focuses on isolating points that deviate from most of the data, without the need to profile the mass. The algorithm constructs a set of binary trees, called isolation trees (iTrees), by recursively partitioning the data through a random selection of features and split values. At each node, a feature is selected at random together with a split value, chosen uniformly between the minimum and maximum values of the features. This recursive process continues until either all points are isolated or a predefined tree height is reached.

Since anomalous points are typically rare and exhibit attribute values that are significantly different from normal ones, they tend to be isolated with fewer splits. Consequently, the average path length from the root node to a given data point becomes a natural indicator of its degree of abnormality: shorter paths suggest a higher likelihood of being an anomaly.

The notion of path length plays a central role in the detection process. For a given data point x, the path length $h(x)$ is defined as the number of edges traversed from the root node to the terminating node in the iTree. As already mentioned, anomalies tend to be easier to isolate: since they can be separated using less random splits, and that is why they show a shorter average path length across the set. On the other hand, normal instances, which are more densely clustered, typically require deeper partitions to be isolated.

To normalize the expected path length across datasets of different sizes, Liu et al.[4] introduce the average path length $c(n)$ of unsuccessful searches in Binary Search Trees, defined as:

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

where $H(i)$ is the i-th harmonic number, approximated by $ln(i) + \gamma$, where γ is the Euler–Mascheroni constant. This quantity represents the average path length for a dataset of size n.

Based on this, the anomaly score for a point x is given by:

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

where $E[h(x)]$ is the average path length of x across all trees. Scores close to 1 indicate anomalies, while scores significantly lower than 0.5 suggest normal observations. The formalization of the average path length provides a clear and intuitive criteria for distinguishing between normal and anomalous data points.

The key strengths of Isolation Forest are its efficiency and scalability. The algorithm operates with linear time complexity relative to the size of the dataset and requires a memory footprint significantly smaller than distance-based or density-based methods. This makes it especially suitable for high-dimensional data or large-scale applications where traditional approaches, such as K-Nearest Neighbors or clustering-based methods, become computationally challenging. Furthermore, because the partitions are created randomly, Isolation Forest avoids assumptions about data distribution and maintains robustness across diverse domains, including fraud detection, intrusion detection in cybersecurity, and fault detection in industrial systems.

2.3 Local Outlier Factor

The Local Outlier Factor (LOF) algorithm, introduced by Breunig et al.[5], is one of the most widely used methods for unsupervised anomaly detection. Unlike global approaches that rely on a single notion of distance or density, LOF identifies anomalies by comparing the local density of a data point to that of its neighbors. Specifically, while other algorithms treat being an outlier as a binary property, it tries to assign to each object a degree of being an outlier. This degree is called *outlier score* and it reflects how isolated the point is with respect to the surrounding neighbors: points that have a substantially lower density than their neighbors are considered outliers. This local perspective makes LOF particularly effective in datasets with heterogeneous density distributions, where global threshold-based methods often fail. Since its introduction, LOF has been extensively applied in domains such as fraud detection, network intrusion detection, and fault diagnosis, and it has inspired several variants and extensions aimed at improving scalability and robustness.

The core concept of the Local Outlier Factor algorithm is that of *local density*, which is estimated using the notion of *reachability* distance between points. For a given object, its local reachability density is defined as the inverse of the average reachability distance with respect to its k-nearest neighbors. This density measure captures how closely a point is surrounded by others in its neighborhood. Given a point p and one of its k-nearest neighbors o, the reachability distance is defined as:

$$\text{reach\_dist}_k(p, o) = \max\{k\text{-distance}(o),\ d(p, o)\}$$

where $d(p, o)$ is the Euclidean distance between p and o, and k-distance(o) is the distance between o and its k-th nearest neighbor. This definition ensures that very close neighbors do not dominate the density estimation, which provides robustness against noise.

Using this measure, the *local reachability density* (LRD) of a point p is expressed as:

$$\text{lrd}_k(p) = \left(\frac{\sum_{o \in N_k(p)} \text{reach\_dist}_k(p, o)}{|N_k(p)|} \right)^{-1}$$

where $N_k(p)$ denotes the set of the k-nearest neighbors of p. Intuitively, $lrd_k(p)$ reflects how densely p is surrounded by its neighbors: a higher value means a denser region around p, and vice versa.

The LOF score of an object is then computed as the average ratio of the local reachability densities of its neighbors to its own. Analytically, the Local Outlier Factor of a point p is defined as:

$$\text{LOF}_k(p) = \frac{\sum_{o \in N_k(p)} \frac{\text{lrd}_k(o)}{\text{lrd}_k(p)}}{|N_k(p)|}$$

Intuitively, a score close to 1 indicates that the point has a density comparable to its neighbors (that is, p lies in a region of similar density as its neighbors), while a value significantly greater than 1 indicates that the point is in a relatively sparse region, and thus it is likely to be an outlier.

One of the most important characteristics of LOF is its relative nature. Instead of relying on fixed global thresholds, the algorithm adapts to the local structure of the data, which allows it to detect anomalies even in regions with varying densities. For example, in datasets containing both dense and sparse clusters, traditional distance-based outlier detection methods tend to misclassify points in sparse clusters as outliers. LOF avoids this issue by evaluating each point with respect to its local context, ensuring that points in naturally sparse regions are not automatically flagged as anomalous.

Breunig et al.[5] also demonstrated that LOF is highly effective in distinguishing between true outliers (points that are meaningfully different from their neighbors) from simple borderline points at the edges of clusters. This distinction is important in practical applications, where mislabeling boundary points as anomalies could lead to the detection of false positives. However, the choice of the parameter k, which indicates the size of the neighborhood, plays a critical role: smaller values make the algorithm sensitive to noise, while larger values may blur local structures.

2.4 Autoencoder

Following the discussion on traditional machine learning approaches for anomaly detection, it is now worth considering the advances introduced by deep learning. Deep learning techniques extend traditional neural network models by incorporating multiple layers of abstraction, which allow them to capture complex patterns in data, which are often inaccessible to shallow models or conventional algorithms. This capability has made deep learning a powerful paradigm for tasks involving high-dimensional or unstructured data.

One of the core concepts of deep learning is Artificial Neural Networks (ANNs): computational models inspired by the functioning of biological neural systems. ANNs are designed to learn non-linear relationships directly from data and they can be employed to adapt to a large variety of problem domains. Their ability to extract meaningful representations makes them particularly suitable for anomaly detection scenarios, where subtle irregularities may not be easily captured by traditional methods.

Within this context, Autoencoders have been one of the most widely adopted neural network architectures for anomaly detection. Designed to learn compressed representations of the input data, Autoencoders attempt to reconstruct the original input as accurately as possible. Although they are mainly used for tasks such as dimensionality reduction, data compression, and denoising, they have also proven valuable for anomaly detection. Given the reconstructed data as output, anomalies can be detected by analyzing the reconstruction error: data points that significantly deviate from the learned representation are flagged as potential outliers. This property has led to a growing interest in Autoencoder-based methods across multiple domains, including cybersecurity, financial crime detection, and industrial monitoring. Here is presented a brief introduction to Autoencoder models inspired by the description made by Hinton and Salakhutdinov[6].

2.4.1 Architecture

Autoencoders are a specific family of feedforward neural networks that are trained to reproduce their input at the output through a bottleneck of reduced dimensionality.

Conceptually, an autoencoder is designed to take an input, compress it into a lower-dimensional representation, and then reconstruct the original input from that compressed form. This process forces the model to learn the most important features of the data. The architecture of an autoencoder consists of three main components:

1. **Encoder** The encoder is the component of the network responsible for mapping the input data into a compact, lower-dimensional representation. By applying successive layers of non-linear transformations, the encoder is able to extract the most informative features while discarding redundancy. Formally, the encoder can be described as a function $f(\cdot)$ that projects the input $\mathbf{x}$ into a latent representation $\mathbf{z} = f(\mathbf{x})$.

   In practice, the encoder is implemented as a stack of fully connected layers. Each layer progressively reduces the dimensionality of the data, so that both the number of layers and the width of each layer (i.e., the number of neurons) act as hyperparameters to be tuned during model design and training.

   Denoting by $a_0 = m, \ldots, a_E = n$ the widths of the decoder layers, a generic layer update can be written as

   $$\mathbf{h}_{e+\mathbf{1}} = \phi_e(\mathbf{W}_e \cdot \mathbf{h}_e + \mathbf{b}_e) \tag{2.1}$$

   where $e = 0, \ldots, E-1$. Here

   - $\mathbf{W}_e \in \mathbb{R}^{a_{e+1} \times a_e}$ is the weight matrix between layer e and $e+1$,
   - $\mathbf{b}_e \in \mathbb{R}^{a_{e+1}}$ is the bias term,
   - ϕ_e denotes the activation function that must be applied to the output vector of layer e before passing it to layer $e+1$ (e.g. ReLU, tanh, sigmoid).

   It is important to notice that the first layer corresponds to the input $\mathbf{h}_0 = \mathbf{x}$ and the final one is the latent representation $\mathbf{h}_E = \mathbf{z}$.

2. **Latent space (or code)** At the center of the autoencoder is the latent space, also called the bottleneck. The latent representation $\mathbf{z}$ is a compressed version of the input, typically with much lower dimensionality than the original data. Its purpose is to retain only the most relevant features while discarding noise or redundant information. Ideally, the latent space captures the essential structure of the data, providing a compact encoding that can later be used for reconstruction or for tasks such as anomaly detection.

   Formally, the code is commonly denoted by $Z \subseteq \mathbb{R}^m$, where m is the latent dimension. If the input space is $X \subseteq \mathbb{R}^n$, with n being the input dimensionality, typically $m \ll n$.

The dimensionality of $\mathbf{z}$ is a key hyperparameter: if it is too large, the network may simply learn to copy the input without extracting useful features; if it is too small, the model may fail to preserve important information. The hyperparameter m is hence tuned during model design and training.

3. **Decoder** The decoder is the final part of the network and is responsible for reconstructing the input from its latent representation. Realized as a stack of fully connected layers, it symmetrically mirrors the encoder's structure and gradually expands the compressed code back to the original dimensionality of the data.

   Formally, the decoder can be expressed as a function $g(\cdot)$ that takes the latent vector $\mathbf{z}$ and produces a reconstruction $\hat{\mathbf{x}} = g(\mathbf{z})$.

   Denoting by $a_0 = m, \ldots, a_D = n$ the widths of the decoder layers, a generic layer update can be written as

$$\mathbf{h}_{d+\mathbf{1}} = \phi_d(\mathbf{W}_d \cdot \mathbf{h}_d + \mathbf{b}_d) \tag{2.2}$$

   where $d = 0, \ldots, D-1$. Here

   - $\mathbf{W}_d \in \mathbb{R}^{a_{d+1} \times a_d}$ is the weight matrix between layer d and $d+1$,
   - $\mathbf{b}_d \in \mathbb{R}^{a_{d+1}}$ is the bias term,
   - ϕ_d denotes the activation function that must be applied to the output vector of layer d before passing it to layer $d+1$.

   It is important to notice that the first layer corresponds to the code $\mathbf{h}_0 = \mathbf{z}$ and the final one to the reconstructed output $\mathbf{h}_D = \hat{\mathbf{x}}$.

   Particular care must be used when chosing the *output* activation ψ of the last layer, since it has to be consistent with the data distribution: the most relevant activation functions are the identity (i.e, no activation function) for unconstrained real-valued outputs (paired with Mean Squared Error), a sigmoid for binary data (paired with Binary Cross-Entropy), or a softmax for categorical outputs (paired with Categorical Cross-Entropy).

   If the encoder has succeeded in extracting meaningful features, the reconstructed output $\hat{\mathbf{x}}$ will be very close to the original input $\mathbf{x}$. The similarity between input and reconstruction is measured through a loss function, such as mean squared error or cross-entropy, which guides the training of both encoder and decoder.

Together, encoder, latent space and decoder define the autoencoder mapping $\mathbf{x} \mapsto \hat{\mathbf{x}} = (g \circ f)(\mathbf{x})$. Proper architectural choices and loss functions selection ensure that this mapping approximates the identity on the data while providing a compact, useful representation for the incoming tasks.

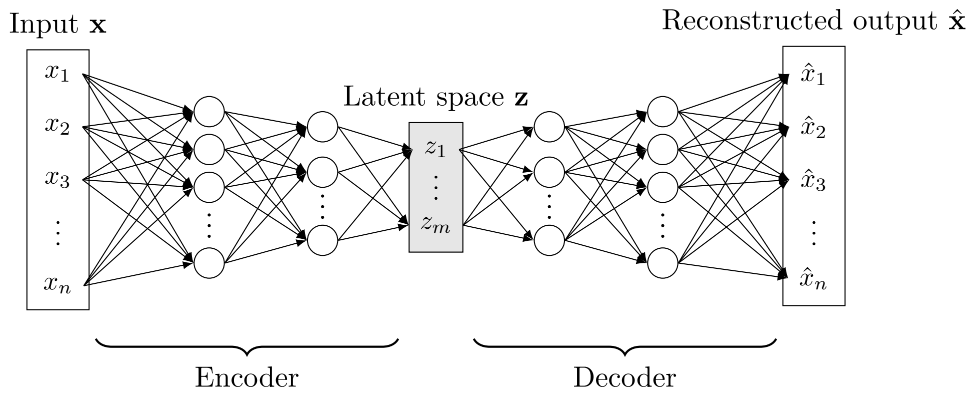


Figure 2.1: Simplified representation of an Autoencoder with fully connected layers.

2.4.2 Training

The training of an autoencoder consists in learning the parameters of the encoder $f(\cdot)$ and decoder $g(\cdot)$ such that the composition $\hat{\mathbf{x}} = (g \circ f)(\mathbf{x})$ approximates the identity mapping on the input data. In other words, the model is trained to minimize the discrepancy between the original input $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$, while simultaneously forcing the information to pass through the reduced latent representation $\mathbf{z}$.

Formally, given a dataset $\{x_i\}_{i=1}^{T} \subseteq \mathbb{R}^n$, where T is the cardinality of the set, the training objective is to minimize the reconstruction loss

$$\mathcal{L}(f,g) \;=\; \frac{1}{T}\sum_{i=1}^{T} L\Big(x_i, g(f(x_i))\Big), \tag{2.3}$$

where $L(\cdot,\cdot)$ is a suitable loss function that measures the distance between input and reconstruction. The choice of L depends on the data distribution and on the activation function ψ applied at the decoder output. Possible alternatives are:

- **Mean Squared Error (MSE)** for continuous, real-valued data, paired with linear outputs: $L(\mathbf{x},\hat{\mathbf{x}}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$;
- **Binary Cross-Entropy (BCE)** for binary or [0,1]-bounded data, paired with sigmoid output activations: $L(\mathbf{x},\hat{\mathbf{x}}) = -\frac{1}{n}\sum_{i=1}^{n}\Big[x_i \log \hat{x}_i + (1-x_i)\log(1-\hat{x}_i)\Big]$;
- **Categorical Cross-Entropy** for one-hot encoded categorical data, paired

with softmax activations: $L(\mathbf{x}, \hat{\mathbf{x}}) = -\sum_{i=1}^{C} x_i \log \hat{x}_i$, where C denotes the number of classes.

Optimization proceeds by adjusting the parameters $\theta = (f, g)$ through iterative updates based on the gradient descent:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta), \tag{2.4}$$

where $\eta > 0$ is the learning rate. In practice, variants such as mini-batch gradient descent and adaptive optimizers (e.g., Adam, RMSProp) are typically employed, as they accelerate convergence and improve stability.

Gradients of the loss with respect to the parameters of each layer are computed via the backpropagation algorithm. This procedure relies on the chain rule to efficiently propagate the error from the output layer backward through the decoder and then the encoder, updating weights $\mathbf{W}_e$ and $\mathbf{W}_d$ and biases $\mathbf{b}_e$ and $\mathbf{b}_d$ at each step.

Through this training process, the encoder learns to extract compact, informative features into the latent representation $\mathbf{z}$, while the decoder learns to use such features to reconstruct the input as accurately as possible.

2.5 k-Nearest Neighbors

The k-Nearest Neighbors (kNN) algorithm is one of the simplest yet most intuitive supervised learning methods, relying solely on the concept of similarity or distance between instances. The basic idea is that a new point to be classified (or predicted) will assume the most common class (or value) among its k nearest neighbors in the training dataset, according to a chosen metric, typically the Euclidean distance. Its simplicity makes it highly interpretable and often used as a baseline for a wide range of classification or regression tasks.

In the original work by Thomas M. Cover and Peter E. Hart, a fundamental theoretical analysis of the one-nearest neighbor rule was provided: the authors demonstrated that, as the number of training samples approaches infinity, the error rate of the nearest neighbor rule does not exceed twice the Bayes optimal error (i.e., the theoretical minimum error) [7]. This asymptotic guarantee established a crucial theoretical benchmark for pattern classification.

The algorithm operates straightforwardly: given a point x to predict, the distance between x and all points in the training set is computed, the k closest points are selected, and x is assigned the most frequent class among those neighbors. Although the mathematical formulation is simple, the choice of k and the distance metric, as well as the handling of noise and high-dimensional data, are critical factors for achieving good performance.

Among its limitations, the kNN algorithm particularly suffers from the so-called "curse of dimensionality": as the number of dimensions increases, distances between points tend to become similar, reducing the discriminative power of the notion of proximity. Moreover, being a lazy learning method, the entire training dataset must be stored in memory, and prediction requires a distance computation against every training sample, resulting in a computational complexity of $O(n)$ per query, where n is the number of training examples, resulting in a total complexity of $O(n^2)$.

Nonetheless, the strength of kNN lies in its versatility and its lack of strong assumptions about the data distribution: it does not require, for instance, the explicit estimation of a density function or a parametric model. For this reason, it is widely used not only in general classification tasks but also in fields such as pattern recognition, recommender systems, bioinformatics, and data mining. However, for large datasets or high-dimensional data, dimensionality reduction techniques are commonly applied before using kNN.

2.6 Logistic Regression

Logistic Regression is one of the most fundamental and interpretable algorithms for binary classification, serving as a bridge between traditional statistical modeling and modern machine learning. Originally formalized by David R. Cox in 1958 [8], it models the probability that an input $\mathbf{x}$ belongs to a particular class using a logistic (sigmoid) transformation applied to a linear combination of input features. The logistic function maps any real-valued input to the interval (0,1), making it suitable for probabilistic interpretation. Formally, the model predicts the conditional probability of class $y = 1$ as

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^\top \mathbf{x} + b)}}, \tag{2.5}$$

where $\mathbf{w}$ represents the weight vector and b the bias term. Training the model consists in estimating these parameters by maximizing the likelihood of the observed data, or equivalently, by minimizing the negative log-likelihood (cross-entropy) loss:

$$\mathcal{L}(\mathbf{w}, b) = -\frac{1}{N} \sum_{i=1}^{N} \Big[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \Big], \tag{2.6}$$

where $\hat{y}_i$ is the predicted probability for the i-th sample. Despite its simplicity, logistic regression provides strong theoretical guarantees and interpretable coefficients, making it a preferred baseline in many supervised learning pipelines. The linear decision boundary in the input space is given by $\mathbf{w}^\top \mathbf{x} + b = 0$, separating the two classes. However, this linearity also represents its main limitation, as it struggles with non-linearly separable data unless feature engineering or kernel extensions are applied. Still, due to its efficiency, scalability, and statistical rigor, logistic regression remains a cornerstone model in domains such as credit scoring, medical diagnosis, and natural language processing, often serving as a benchmark against which more complex models are compared.

2.7 Decision Tree

The Decision Tree algorithm represents one of the most interpretable and intuitive models in supervised learning, capable of handling both classification and regression tasks. Its operation is based on recursively partitioning the input space into regions that are as homogeneous as possible with respect to the target variable. In the work by J. Ross Quinlan [9], the ID3 algorithm was introduced, laying the foundation for subsequent methods. A decision tree can be viewed as a hierarchical structure where each internal node corresponds to a feature test, each branch represents an outcome of that test, and each leaf node denotes a predicted class or value. The core idea is to select, at each step, the feature that best splits the data according to a given impurity measure, such as information gain, Gini index, or entropy reduction. The entropy of a dataset S is defined as

$$H(S) = -\sum_{c \in \mathcal{C}} p(c) \log_2 p(c), \tag{2.7}$$

where $p(c)$ denotes the proportion of samples belonging to class c. The information gain achieved by splitting on a feature A is then given by

$$\text{Gain}(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v), \tag{2.8}$$

which quantifies the expected reduction in entropy. The recursive process continues until a stopping criterion is met, such as a maximum depth or a minimum number of samples per node. Decision trees naturally handle both categorical and numerical attributes and require minimal data preprocessing. However, they are prone to overfitting, as they can grow excessively complex and adapt too closely to the training data. To mitigate this issue, pruning techniques or ensemble approaches such as Random Forests are commonly employed. Despite their limitations, decision trees remain a cornerstone of interpretable machine learning, valued for their clarity, low computational cost and ability to capture non-linear decision boundaries.

2.8 Random Forest

The Random Forest algorithm, introduced by Leo Breiman in 2001 [10], is an ensemble learning method designed to improve the predictive performance and stability of decision trees while mitigating their tendency to overfit. The core idea is to build a large collection of decorrelated decision trees and aggregate their predictions through majority voting (for classification) or averaging (for regression). Each tree in the ensemble is trained on a different sample of the training dataset and, at each split, a random subset of features is considered, introducing further diversity among the trees. Formally, given B trees, the final prediction for an input $\mathbf{x}$ is computed as

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} f_b(\mathbf{x}), \tag{2.9}$$

where $f_b(\mathbf{x})$ denotes the prediction of the b-th decision tree. This ensemble mechanism substantially reduces the model variance without increasing the bias, resulting in robust generalization across a wide range of datasets. The randomness introduced during training ensures that individual trees capture different structures within the data, while their aggregation smooths out noise and irregularities. Furthermore, Random Forests provide an estimate of feature importance, typically computed as the average decrease in impurity or as the reduction in prediction accuracy when the values of a feature are permuted.

Despite their efficiency and scalability, Random Forests are less interpretable than single decision trees, and their performance can degrade when applied to very high-dimensional or sparse datasets. Due to their balance between accuracy, robustness, and ease of use, Random Forests have become one of the most widely adopted ensemble methods in modern machine learning, serving as a strong baseline for both academic research and industrial applications.

2.9 Support Vector Machine

The Support Vector Machine (SVM) is a powerful supervised learning algorithm introduced by Corinna Cortes and Vladimir Vapnik in 1995 [11]. It is founded on the principle of finding the optimal hyperplane that separates the data points of different classes in the feature space. The optimal hyperplane is defined as the one that maximizes the margin, i.e., the distance between the hyperplane and the nearest data points from each class, known as support vectors. Given a set of labeled samples $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$, where $y_i \in \{-1, +1\}$, the optimization problem can be formulated as

$$\min_{\mathbf{w},b} \frac{1}{2}|\mathbf{w}|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, \dots, N, \tag{2.10}$$

where $\mathbf{w}$ and b define the separating hyperplane. To handle non-linearly separable data, the soft-margin SVM introduces slack variables ξ_i and a regularization parameter C that balances margin maximization and classification error. Furthermore, through the use of kernel functions, the SVM can implicitly map input vectors into a higher-dimensional feature space, where linear separation becomes possible. The most common kernel is the Radial Basis Function (RBF), defined as

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{(-\gamma|\mathbf{x}_i - \mathbf{x}_j|^2)}, \tag{2.11}$$

where γ controls the influence of individual training samples. This kernel trick allows SVMs to construct highly flexible, non-linear decision boundaries without explicitly computing the high-dimensional mapping. Although SVMs provide strong theoretical guarantees and often achieve high accuracy, their computational cost can be prohibitive for large datasets due to the quadratic optimization process. Nevertheless, they remain one of the most influential algorithms in machine learning, particularly valued for their robustness, mathematical elegance, and effectiveness in high-dimensional spaces such as text classification, bioinformatics, and image recognition.

Chapter 3

Proposed methodology

This chapter presents the workflow followed during the development of this thesis. The process began with a thorough data exploration phase: the bank provided several datasets along with their guidance and expertise to explain the structure of each table and field. The exploration focused on assessing the data distributions, identifying missing values and, most importantly, determining which variables were most relevant. A detailed discussion of this step is provided in Section 3.1.

The objective of the data exploration was to lay the foundation for the subsequent feature extraction phase (Section 3.2). In close collaboration with the bank's experts, we defined the set of features required to model the transactional behavior of each account. This step was particularly critical, as these features serve as input to the machine learning models to enable the detection of anomalies and, consequently, potential criminal activities.

To ensure a fair evaluation of the models, a set of performance metrics was also established, as described in Section 3.3. Finally, all these steps were consolidated into a unified data processing pipeline (Section 3.6), designed for adoption by the bank in its production environment.

In order to comply with current privacy regulations and to safeguard the confidentiality of the institution's customers, all datasets underwent an anonymization process before being made available for this work. This procedure was entirely managed by the institution, which provided datasets where sensitive attributes (such as customer names or account reference numbers) had been fully anonymized. For the same reason, some of the data reported in this thesis are presented in an altered form: certain field names have been modified, and some distributions have been scaled by standard deviation (e.g., divided by the standard deviation, which is not reported) to preserve the confidentiality of the aggregated data.

3.1 Datasets description

3.1.1 Transactions

The central dataset used in this thesis is the transactions dataset, which records the complete set of financial operations performed by customers during the year 2024 (from 01/01/2024 to 31/12/2024). In total, the dataset contains 3 722 744 322 transactions carried out by 13 351 999 distinct accounts (accounts are formally defined in Section 1.2).

This dataset is of primary importance since it captures the core of the customers' financial activity. It provides both the temporal dimension of the behavior (through timestamps) and the economic one (through transaction amounts). In addition, categorical information such as the reason for the transaction enables a richer characterization of account activity.

The structure of the dataset is summarized in Table 3.1:

Column name	**Data type**	**Description**
ACCOUNT_ID	String	Unique identifier of the account that performed the transaction.
TIMESTAMP	DateTime	Exact date and time when the transaction was executed.
REASON_ID	String	Encoded reason of the transaction (as explained in Sub-section 3.1.2).
AMOUNT	Numeric	Monetary value of the transaction (positive or negative, normalized as explained in Section 3).

Table 3.1: Transactions dataset structure.

A number of preliminary observations can be made on this dataset. First, the scale is particularly relevant, with the transactions being evenly distributed throughout the months, as shown in Table 3.2.

Month	Transactions count
2024-01	311 249 514
2024-02	277 769 084
2024-03	282 881 084
2024-04	313 559 128
2024-05	307 876 069
2024-06	289 525 314
2024-07	353 409 685
2024-08	294 295 522
2024-09	306 488 090
2024-10	333 623 526
2024-11	297 315 472
2024-12	354 751 834

Table 3.2: Number of transactions per month in 2024.

Second, the distribution of transactions across accounts is highly heterogeneous (Figure 3.1): while the vast majority of accounts are associated with only one or a few transactions over the year, a small subset exhibits extremely intense activity. This behavior becomes even clearer when analyzed through the Empirical Cumulative Distribution Function (ECDF), shown in Figure 3.2, which represents the proportion of accounts that perform up to a given number of transactions.

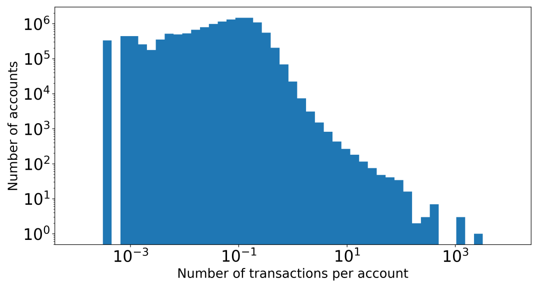


Figure 3.1: Distribution of the number of accounts per number of transactions.

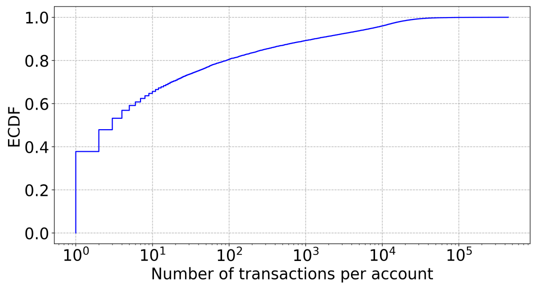


Figure 3.2: Empirical Cumulative Distribution Function of the number of transactions per number of accounts.

Finally, the range of transaction amounts spans several orders of magnitude. Due to privacy considerations, only the order of magnitude of the transaction values is shown in Figure 3.3. The distribution is approximately bell-shaped and resembles a normal distribution, with the vast majority of transactions having relatively low values, while a small fraction reaches extremely high amounts. This is confirmed by the Empirical Cumulative Distribution Function (ECDF) of the absolute value of the order of magnitude of the transactions shown in Figure 3.4.

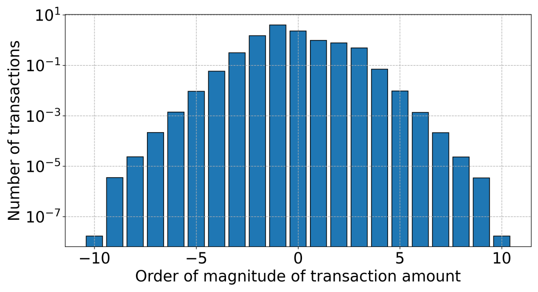


Figure 3.3: Distribution of transaction values by order of magnitude.

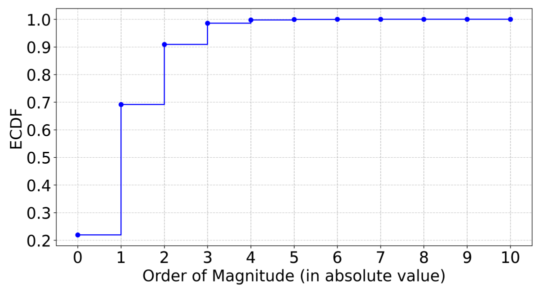


Figure 3.4: Empirical Cumulative Distribution Function of the amount of transactions per absolute value of order of magnitude.

3.1.2 Reasons

In addition to the transaction data set, the reason data set provides the semantic and categorical context of the financial operations. Each transaction recorded in the main dataset is associated with exactly one REASON_ID, which acts as a foreign key referencing this dataset. In total, the dataset contains 1 483 distinct reasons, enabling the interpretation of transaction purposes and thus supporting a better understanding of financial behavior.

The structure of the dataset is summarized in Table 3.3:

Column name	**Data type**	**Description**
REASON_ID	String	Unique identifier of the transaction reason.
REASON_DESCRIPTION	String	Human-readable description of the reason (e.g., “Salary payment”, “Tax payment”).
CATEGORY	String	High-level category grouping similar transaction reasons (e.g.,“Administrative Transactions”, “Service Fees”).
IS_WIRE	Boolean	Indicates whether the transaction corresponds to a wire transfer.
IS_CASH	Boolean	Indicates whether the transaction corresponds to a cash operation.

Table 3.3: Reasons dataset structure.

The categorization of reasons allows grouping transactions into meaningful classes. Table 3.4 reports the categories and their frequency in the reasons and transactions datasets.

Additionally, among all reasons, 85 are explicitly associated with wire transfers

Category	Reasons [%]	Transactions [%]
Non-attributable transactions	22	72.06
Contract-Related Payments	23.4	10.11
Service Fees	21.6	9.49
Ad-hoc Transaction	15.7	4.08
Recurring Transaction	0.9	2.14
Administrative Transactions	16.4	2.11

Table 3.4: Distribution of reasons and transactions across categories.

(IS_WIRE = true), and 57 correspond to cash operations (IS_CASH = true). It is important to note that the dataset design ensures mutual exclusivity: no reason can simultaneously be classified as both wire and cash.

3.1.3 Balances

The balances dataset provides information about the monthly amount of funds available for each account. For every account, the dataset records the balance at the end of each month, thus complementing the transactions dataset with an aggregated financial perspective over time. The total cardinality of the dataset is 184 624 194 records. The structure is summarized in Table 3.5.

Column name	Data type	Description
ACCOUNT_ID	String	Unique identifier of the account.
YEAR_MONTH	Numeric	Reference month of the record, expressed as YYYYMM.
BALANCE	Numeric	Balance of the account at the end of the reference month.

Table 3.5: Balances dataset structure.

Before employing this dataset in the analysis, a data quality process was carried out to handle missing values. Two distinct strategies were applied:

- If an account had no valid (non-null) balance values across all months, the balance at the end of December 2023 was assumed to be zero; from this baseline, the balance for January 2024 was reconstructed by applying the net effect of transactions occurring in January, and subsequent monthly balances were obtained by cumulatively adding the amounts of the corresponding transactions.
- If at least one valid balance value was available for an account, this was used as a starting point to reconstruct the balances of the other months by

propagating the effect of transactions forward and backward in time.

This preprocessing step ensured a consistent and complete representation of monthly balances across all accounts, enabling reliable use of this dataset in later modeling phases.

3.1.4 Relationships

The relationships dataset is fundamental because it enables the mapping of each ACCOUNT_ID to a CUSTOMER_ID, thus establishing the association between accounts and their owners. In the case of shared accounts, an account is associated with a JOINT_ACCOUNT_ID, as will be explained in Section 3.1.6. The dataset contains a total of 123 190 721 records. Its structure is reported in Table 3.6.

Column name	**Data type**	**Description**
CUSTOMER_ID	String	Unique identifier of the customer owning the account or of a joint account.
ACCOUNT_ID	String	Unique identifier of the account.
ACCOUNT_TYPE	String	Code that identifies the type of bank account, such as savings account, current account, prepaid card, and similar categories.
VALIDITY_START	DateTime	Date and time when the association between the account and the customer becomes valid.
VALIDITY_END	DateTime	Date and time when the association between the account and the customer ceases to be valid.

Table 3.6: Relationships dataset structure.

It is important to highlight that account ownership can change over time. For this reason, the same account can be associated with multiple customers at different periods. Whenever this occurs, a condition of mutual exclusivity is enforced: the validity end of one association must always precede the validity start of the next one, ensuring that two ownership intervals for the same account never overlap. While a customer can hold multiple accounts simultaneously, an account can only be associated with a single customer identifier or with a joint account identifier in the case of shared ownership.

3.1.5 Registry

The registry dataset contains specific information about customers. It includes a total of 43 870 876 records and its structure is reported in Table 3.7.

Column name	**Data type**	**Description**
CUSTOMER_ID	String	Unique identifier of the customer owning the account or of a joint account.
CUSTOMER_TYPE	String	Type of customer: physical person, legal entity or joint account.
BIRTH_YEAR	Numeric	Year of birth of a physical person or year of foundation of a legal entity.
D23	Numeric	Code representing the economic segmentation of the customer.
D23_DESCRIPTION	String	Textual description corresponding to the D23 code.

Table 3.7: Registry dataset structure.

CUSTOMER_TYPE

The field CUSTOMER_TYPE distinguishes among three categories of customers.

- The first category is “physical person”, which identifies natural persons acting as individual account holders. A total of 30 759 168 records, representing approximately 70% of all entries in the dataset, fall into this category.
- The second is “legal entity”, which refers to juridical persons such as companies, institutions or other organizations. Approximately 11.5% of the records (5 008 084 entries) belong to this category.
- The third category is “joint account”, which indicates that the identifier refers to multiple account holders; in this case, the mapping between a joint account identifier and the corresponding individuals is managed in the links dataset (Section 3.1.6). This category accounts for 8 103 624 records, corresponding to 18.5% of the dataset.

This field is particularly relevant for behavioral analysis, as the type of customer provides implicit information on the expected patterns of activity.

BIRTH_YEAR

The field BIRTH_YEAR is relevant because it enables the derivation of two features: the age of the individual in the case of physical persons, and the active time in the case of legal entities. These features will be formally introduced in Section 3.2.

During the initial data quality assessment, several inconsistencies regarding this field were identified. Some values are implausible, with the earliest recorded birth year being 1051, which is clearly unacceptable. In total, approximately 130 records report a birth year prior to 1800. Moreover, about 843,300 records (roughly 2% of the dataset) correspond to a birth year earlier than 1925, which would imply customers aged 90 years or more. The distribution and the ECDF of BIRTH_YEAR, restricted to values greater than 1900 for visualization purposes, are illustrated in Figure 3.5 and 3.6.

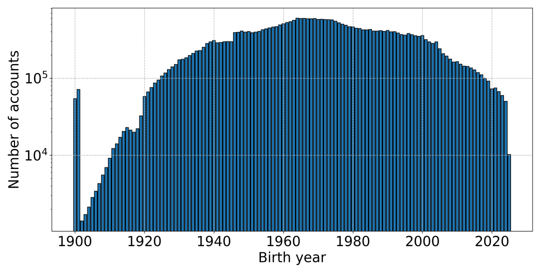


Figure 3.5: Distribution of the birth year.

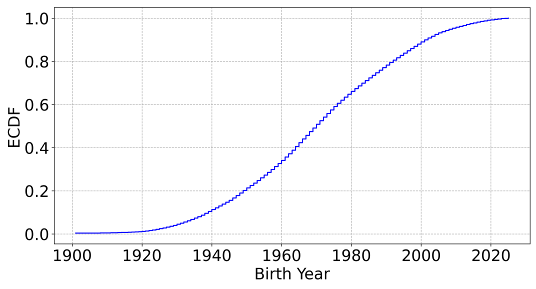


Figure 3.6: Empirical Cumulative Distribution Function of the number of the birth year.

A further critical issue concerns missing values. A total of 9 234 546 records contain a null value for this field, representing 21% of the dataset. The strategies adopted to address these data quality problems, and their implications for features extraction, are discussed in Section 3.2.3.

D23

The fields D23 and D23_DESCRIPTION provide an economic segmentation of the customers. This classification was introduced by the Italian Financial Intelligence Unit (*Unità di Informazione Finanziaria per l'Italia*, UIF) in collaboration with the Bank of Italy (*Banca d'Italia*). Its most recent update [12] defines a segmentation scheme that condenses the essential characteristics of a customer's economic activity into approximately thirty categories. Specifically, the D23 code integrates two complementary dimensions:

1. the **ATECO** classification, a hierarchical system adopted by the Italian government to identify and encode the economic activity of a customer;

2. the **SAE** classification, a banking standard used to categorize subjects according to their economic sector.

The resulting segmentation enables a compact but meaningful representation of economic roles, which is fundamental for anomaly detection tasks.

Table 3.8 reports an exhaustive list of possible values of the D23 field together with their corresponding descriptions.

D23 Code	Description
101	Central and other public administrations.
102	Local administrations.
103	Public health services.
104	Public welfare, recreational and cultural services.
200	Insurance companies and pension funds.
310	Banking system.
311	Financial intermediaries.
312	Other financial intermediaries.
410	Agriculture.
411	Mining, energy, petrochemical and steel industries.
412	Construction industry.
413	Machinery and equipment manufacturing.
414	Food industry.
415	Textile industry.
416	Other industrial products.
510	Wholesale trade.
511	Retail trade.
512	Hospitality and catering.
513	Transport services.
514	Real estate rental and financial auxiliary services.
515	Waste management services.
516	Healthcare services.
517	Other services for sale.
600	Household consumers.
601	Household producers.
711	Rest of the world – non-financial corporations, households, and public administrations.
712	Rest of the world – banking corporations.
713	Rest of the world – financial corporations.
811	Others.
812	Non-profit sector.

Table 3.8: Examples of D23 codes and corresponding descriptions.

As indicated by the field CUSTOMER_TYPE, the majority of the population belongs to the category of physical persons. A similar trend emerges when analyzing the distribution of customers across the different D23 codes, as illustrated in Figure 3.7. The strong correlation between the two columns is shown in Figure 3.8.

It is important to note, due to the logarithmic scale on the y-axis, that approximately 85.5% of the records are concentrated in the D23 code “600”, corresponding to the description “Household consumers”, generally referring to private individuals,

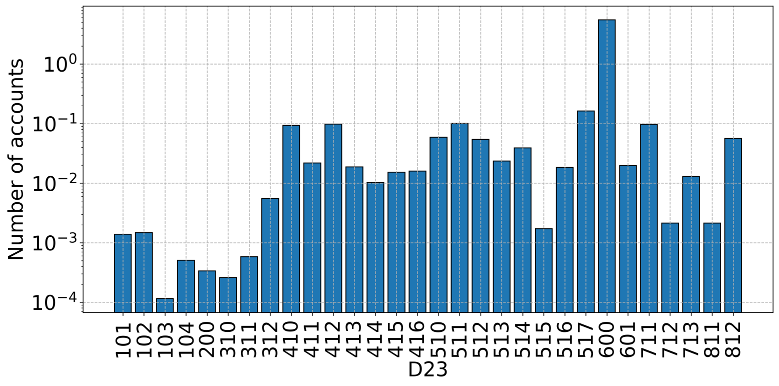


Figure 3.7: Distribution of D23.

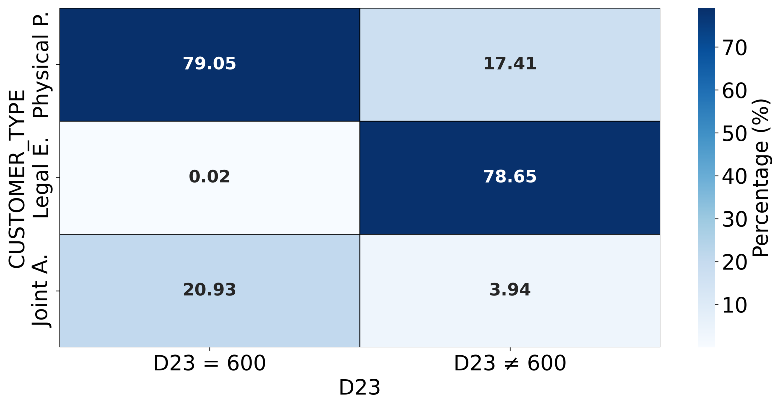


Figure 3.8: Distribution of category of D23 across CUSTOMER_TYPE.

while the remaining 14.5% corresponds to corporate and institutional entities. This strong predominance highlights a significant imbalance in the dataset: training a machine learning model directly on this data could lead to the unintended consequence of systematically classifying all non-household consumers (such as companies, institutions or organizations) as anomalous solely because of their category. Intuitively, this observation is well-founded, as the transactional behavior of a private individual is considerably different from that of a company or a national institution, both in terms of transaction volumes and frequency. Addressing this imbalance is a crucial challenge, and the strategy adopted in this thesis to mitigate it will be described in Chapter 4.

3.1.6 Links

As introduced in Section 3.1.4, the links dataset is employed to record all the holders associated with a joint account. The dataset contains 17 411 555 records and its structure is presented in Table 3.9.

Column name	**Data type**	**Description**
JOINT_ACCOUNT_ID	String	CUSTOMER_ID of the joint account.
JOINT_ACCOUNT_HOLDER_ID	String	CUSTOMER_ID of the joint account holder.
HOLDER_NUMBER	Numeric	Ordinal number of the account holder.

Table 3.9: Links dataset structure.

The field JOINT_ACCOUNT_ID corresponds to a CUSTOMER_ID whose CUSTOMER_TYPE is set to “joint account”. In this case, the identifier does not represent a single physical or legal entity, but rather a group of customers. All the holders associated with that account are identified by means of their JOINT_ACCOUNT_HOLDER_ID, which must always refer to individual customers and never to another joint account. For legal reasons, one of the holders must be designated as the primary account holder: this role is determined through the field HOLDER_NUMBER, where the lowest value indicates the main holder. This association is fundamental, as it enables the extraction of specific features that would otherwise be impossible to derive, as will be detailed in Section 3.2.

3.1.7 Ground Truth

The Ground Truth dataset is employed for the performance evaluation of the trained models. Its structure, summarized in Table 3.10, is straightforward: for each month, every reported account (as introduced in Section 1.1) is assigned a label that specifies whether the detection corresponds to a True Positive (TP) or a False Positive (FP).

Column name	**Data type**	**Description**
ACCOUNT_ID	String	Unique identifier of the account.
YEAR_MONTH	Numeric	Reference month of the record, expressed as YYYYMM.
STATUS	STRING	Current status for the account (either “TP” or “FP”).

Table 3.10: Ground truth dataset structure.

It is worth noting that the dataset does not specify whether the decision originates from competence centres L1, L2, or from the FIU. Consequently, the assigned status may be provisional, as the evaluation of each case can evolve over time with the progression of the analysis performed at different levels of competence.

The dataset consists of 78 330 entries distributed across the months of 2024. Table 3.11 provides the monthly distribution of records, highlighting the relative proportion of TPs and FPs. The analysis shows that False Positives dominate across all months while True Positives maintain a minority share.

Month	**Records** [%]	**TPs** [%]	**FPs** [%]
2024-01	8	21.92	78.08
2024-02	7.7	23.07	76.93
2024-03	8.3	21.24	78.76
2024-04	7.4	23.63	76.37
2024-05	8.5	20.97	79.03
2024-06	8.2	18.82	81.18
2024-07	9.6	19.44	80.56
2024-08	7	16.85	83.15
2024-09	7.1	19.21	80.79
2024-10	8.6	17.58	82.42
2024-11	8.5	15.98	84.02
2024-12	11	12.73	87.27

Table 3.11: Distribution of TPs and FPs across months in the Ground Truth dataset.

3.2 Features extraction

For each account, uniquely identified by the anonymized hash ACCOUNT_ID, a set of 98 human-readable features is computed to characterize the monthly behavior of the account.

The calculation relies on both the incoming operations (credits) and the outgoing operations (debits) registered within the month under analysis. These two categories of movements represent the fundamental basis from which the majority of behavioral indicators are derived, allowing the activity of the account to be quantified in terms of both volume and frequency.

However, for the computation of certain features, it was necessary to extend the analysis beyond the transactions of the current month. In these cases, additional information was incorporated concerning the historical activity of the account, ensuring that temporal dependencies were adequately captured. The specific requirements and methodological details for these cases are discussed within the individual feature descriptions.

3.2.1 Accounts and transactions excluded from the analysis

According to the guidelines provided by the bank's experts, certain accounts and specific categories of transactions were excluded from the feature extraction process.

Excluded accounts

The excluded accounts are:

- A category of accounts, namely "savings accounts", as they were deemed not relevant in terms of transaction activity;
- Accounts with minimal activity, defined as those with both credit and debit volumes below a specific threshold (not disclosed here for confidentiality reasons). Additionally, only accounts performing at least three transactions, including at least one cash operation or wire transfer, were retained for analysis.

The suspiciousness of the excluded accounts may be considered in future investigations.

Excluded transactions

Regarding transactions, exclusions were applied to those associated with a specific set of REASON_IDs. In particular, these correspond to pairs of transactions and their reversals. A reversal occurs when a transaction is canceled, which may happen for various reasons; in this case, the original transaction and its corresponding reversal form a pair. All such pairs were removed from the dataset, as they are equivalent to transactions that never actually took place. The percentage of users for whom features were successfully extracted in each month of 2024 is reported in Table 3.12.

3.2.2 Movements description features

These features are designed to capture the fundamental characteristics of account activity. They are computed considering both the number and the amount of transactions performed during a month, distinguishing between credits and debits, as well as between operations performed in cash and by wire transfer. A transaction is defined as "credit" if its corresponding amount is positive, whereas it is a "debit" if its amount is negative. Each transaction can be a cash operation (flagged by the variable IS_CASH), a wire transfer (flagged by the variable IS_WIRE) or neither. For clarity, the unit of measure of each feature is always reported in square brackets.

Month	Percentage of users
2024-01	69.62%
2024-02	72.61%
2024-03	69.43%
2024-04	70.38%
2024-05	73.46%
2024-06	74.00%
2024-07	72.87%
2024-08	72.65%
2024-09	72.83%
2024-10	69.93%
2024-11	74.29%
2024-12	75.21%

Table 3.12: Monthly percentage of users for whom feature extraction was successfully performed.

Features based on credits

1. **NUM_CREDITS** Number of credit transactions (i.e., operations with positive amount) [count].
2. **CREDITS_VOLUME** Total sum of the amounts of all credit transactions [€].
3. **CREDITS_MIN** Minimum amount of credit transactions [€].
4. **CREDITS_MAX** Maximum amount of credit transactions [€].
5. **CREDITS_MEAN** Average amount of credit transactions [€].
6. **CREDITS_MEDIAN** Median amount of credit transactions [€].
7. **CREDITS_STD** Standard deviation of the amount of credit transactions [€].

Features based on debits

8. **NUM_DEBITS** Number of debit transactions (i.e., operations with negative amount) [count].
9. **DEBITS_VOLUME** Total sum of the amounts of all debit transactions (absolute value) [€].
10. **DEBITS_MIN** Minimum amount of debit transactions (absolute value) [€].

11. **DEBITS_MAX** Maximum amount of debit transactions (absolute value) [€].
12. **DEBITS_MEAN** Average amount of debit transactions (absolute value) [€].
13. **DEBITS_MEDIAN** Median amount of debit transactions (absolute value) [€].
14. **DEBITS_STD** Standard deviation of the amount of debit transactions (absolute value) [€].

Features based on total transactions

15. **NUM_TRANSACTIONS** Total number of transactions (credits + debits) [count].
16. **TOTAL_MOVEMENT** Total sum of the amounts of all transactions (absolute value) [€].
17. **AVG_TX_VALUE** Average amount of all transactions (absolute value) [€].
18. **MEDIAN_TX_VALUE** Median amount of all transactions (absolute value) [€].
19. **STD_TX_VALUE** Standard deviation of transaction amounts (absolute value) [€].

Features based on wire transfers only

20. **NUM_CREDITS_W** Number of credit transactions identified as wire transfers [count].
21. **CREDITS_W_VOLUME** Total sum of amounts of credit wire transfers [€].
22. **CREDITS_W_MIN** Minimum transaction amount of credit wire transfers [€].
23. **CREDITS_W_MAX** Maximum transaction amount of credit wire transfers [€].
24. **CREDITS_W_MEAN** Average transaction amount of credit wire transfers [€].

25. **CREDITS_W_MEDIAN** Median transaction amount of wire credit transfers [€].

26. **CREDITS_W_STD** Standard deviation of the amount of credit wire transfers [€].

27. **NUM_DEBITS_W** Number of debit transactions identified as wire transfers [count].

28. **DEBITS_W_VOLUME** Total sum of amounts of debit wire transfers (absolute value) [€].

29. **DEBITS_W_MIN** Minimum transaction amount of debit wire transfers (absolute value) [€].

30. **DEBITS_W_MAX** Maximum transaction amount of debit wire transfers (absolute value) [€].

31. **DEBITS_W_MEAN** Average transaction amount of debit wire transfers (absolute value) [€].

32. **DEBITS_W_MEDIAN** Median transaction amount of debit wire transfers (absolute value) [€].

33. **DEBITS_W_STD** Standard deviation of the amounts of debit wire transfers (absolute value) [€].

34. **NUM_TXS_W** Total number of transactions identified as wire transfers (credits + debits) [count].

35. **TOTAL_MOVEMENT_W** Total sum of all amounts of wire transfers (absolute value) [€].

36. **TOTAL_W_MEAN** Average amount of wire transfers (absolute value) [€].

37. **TOTAL_W_MEDIAN** Median amount of wire transfers (absolute value) [€].

38. **TOTAL_W_STD** Standard deviation of the amounts of wire transfers (absolute value) [€].

Features based on cash operations only

39. **NUM_CREDITS_C** Number of credit transactions identified as cash operations [count].
40. **CREDITS_C_VOLUME** Total sum of amounts of cash credit operations [€].
41. **CREDITS_C_MIN** Minimum amount of credit cash operations [€].
42. **CREDITS_C_MAX** Maximum amount of credit cash operations [€].
43. **CREDITS_C_MEAN** Average amount of credit cash operations [€].
44. **CREDITS_C_MEDIAN** Median amount of credit cash operations [€].
45. **CREDITS_C_STD** Standard deviation of the amounts of credit cash operations [€].
46. **NUM_DEBITS_C** Number of debit transactions identified as cash operations [count].
47. **DEBITS_C_VOLUME** Total sum of the amounts of cash debit operations (absolute value) [€].
48. **DEBITS_C_MIN** Minimum amount of debit cash operations (absolute value) [€].
49. **DEBITS_C_MAX** Maximum amount of debit cash operations (absolute value) [€].
50. **DEBITS_C_MEAN** Average amount of debit cash operations (absolute value) [€].
51. **DEBITS_C_MEDIAN** Median amount of debit cash operations (absolute value) [€].
52. **DEBITS_C_STD** Standard deviation of the amounts of debit cash operations (absolute value) [€].

Features based on total transactions

53. **NUM_TXS_C** Total number of transactions identified as cash operations (credits + debits) [count].
54. **TOTAL_MOVEMENT_C** Total sum of the amounts of all cash operations (absolute value) [€].

55. **TOTAL_C_MEAN** Average amount of cash operations (absolute value) [€].
56. **TOTAL_C_MEDIAN** Median amount of cash operations (absolute value) [€].
57. **TOTAL_C_STD** Standard deviation of the amounts of cash operations (absolute value) [€].

Additional movements-based features

Further features related to the transactions performed within the month under analysis are considered:

58. **AVERAGE_DAILY_ACCOUNT_BALANCE** Average daily balance of the account [€].
59. **NUM_DAYS_CASH** Number of days with at least one transaction identified as a cash operation [days].
60. **AVG_DAYS_BETWEEN_CASH_CREDITS** Average number of days between two cash credit transactions [days].
61. **AVG_DAYS_BETWEEN_CASH_DEBITS** Average number of days between two cash debit transactions [days].
62. **CASH_STRUCTURING** Boolean value indicating whether the total volume of all cash transactions (in absolute value) exceeds 10 000€ and, simultaneously, the total volume of cash transactions (in absolute value) with individual amounts below 1 000€ is lower than 10 000€ [dimensionless].
63. **WIRE_STRUCTURING_DEBIT** Boolean value indicating whether the total volume of debit wire transactions (in absolute value) with individual amounts below 5 000€ exceeds 15 000€ [dimensionless].
64. **WIRE_STRUCTURING_CREDIT** Boolean value indicating whether the total volume of incoming wire transactions with individual amounts below 5 000€ exceeds 15 000€ [dimensionless].

3.2.3 Subjective and demographic features

These features describe static and categorical characteristics of the account holder, which are then referred to the account and are useful for understanding the economic profile of the user.

In the case of joint accounts, a hierarchy among co-holders is defined with the support of domain experts, as described in Section 3.1.6: the primary account holder's CUSTOMER_ID is used to compute these features.

In the case where the BIRTH_YEAR value is unavailable, the missing value is propagated to the derived feature.

65. **ACTIVE_MONTHS** Number of months of past account activity; a maximum value of 12 is assigned if the user has been a client for more than one year [months].
66. **NUMBER_OF_PARTICIPANTS** Number of account holders in the case of a joint account, otherwise equal to one [users].
67. **D23** Economic segmentation code, as described in Section 3.1.5.
68. **CURRENT_BALANCE** Account balance at the end of the current month [€].
69. **CUSTOMER_TYPE** Account holder's customer type, as described in Section 3.1.5, with only two possible values: "physical person" or "legal entity" [category].
70. **AGE** Account holder's age, calculated as the difference between the current year and the customer's BIRTH_YEAR if the customer is a "physical person", left empty otherwise; a maximum value of 100 is assigned if the user has a reported age higher than that [count].
71. **RECENT_ACTIVATION** Boolean value indicating whether the period of activity of the account is smaller than two years if the customer is a "legal entity", left empty otherwise. The period of activity is calculated as the difference between the current year and the customer's BIRTH_YEAR (considered as the year of foundation of a legal entity) [dimensionless].

3.2.4 Features for trend evaluation

These features describe the behavior of the account during the previous month and the differences between the previous and current, allowing for the analysis of its temporal evolution and the detection of significant changes in its usage.

72. **PREVIOUS_BALANCE** Account balance at the end of the previous month [€].
73. **PREVIOUS_AVERAGE_DAILY_ACCOUNT_BALANCE** Average daily account balance during the previous month [€].

74. **DELTA_CREDITS_VOLUME** Difference between CREDITS_VOLUME and the total amount of credits during the previous month [€].

75. **DELTA_DEBITS_VOLUME** Difference between DEBITS_VOLUME and the total amount of debits (in absolute value) during the previous month [€].

76. **DELTA_CREDITS_W_VOLUME** Difference between CREDITS_W_ VOLUME and the total amount of wire transfer credits during the previous month [€].

77. **DELTA_DEBITS_W_VOLUME** Difference between DEBITS_W_ VOLUME and the total amount of wire transfer debits (in absolute value) during the previous month [€].

78. **DELTA_CREDITS_C_VOLUME** Difference between CREDITS_C_ VOLUME and the total amount of cash credit operations during the previous month [€].

79. **DELTA_DEBITS_C_VOLUME** Difference between DEBITS_C_ VOLUME and the total amount of cash debit operations (in absolute value) during the previous month [€].

3.2.5 Features on weekly movements

These features capture the weekly behavioral variations of the account holder and are defined over a period of six weeks, counted backwards from the last full week (Monday to Sunday) entirely contained within the month under analysis. For example, if November 2024 ends on a Saturday, the last full week within the month corresponds to the period from Monday, November 18th to Sunday, November 24th.

For each of the six weeks, the following quantities are computed:

Cash transactions

- WEEK_C_DEBIT_VOLUME: Total amount of debit cash operations during the week [€].

- WEEK_C_CREDIT_VOLUME: Total amount of debit cash operations during the week [€].

Wire transfers

- WEEK_W_DEBIT_VOLUME: Total amount of debit wire transfers during the week [€].
- WEEK_W_CREDIT_VOLUME: Total amount of credit wire transfers during the week [€].

Based on these values, the following features are defined:

Weekly cash debits

80. **WEEK_C_DEBIT_VOLUME_MEAN** Mean value of WEEK_C_DEBIT_VOLUME across the six weeks [€].
81. **WEEK_C_DEBIT_VOLUME_STD** Standard deviation of WEEK_C _DEBIT_VOLUME across the six weeks [€].

Weekly cash credits

82. **WEEK_C_CREDIT_VOLUME_MEAN** Mean value of WEEK_C_CREDIT_ VOLUME across the six weeks [€].
83. **WEEK_C_CREDIT_VOLUME_STD** Standard deviation of WEEK_C_CREDIT_VOLUME across the six weeks [€].

Weekly wire debits

84. **WEEK_W_DEBIT_VOLUME_MEAN** Mean value of WEEK_W_DEBIT_VOLUME across the six weeks [€].
85. **WEEK_W_DEBIT_VOLUME_STD** Standard deviation of WEEK _W_DEBIT_VOLUME across the six weeks [€].

Weekly wire credits

86. **WEEK_W_CREDIT_VOLUME_MEAN** Mean value of WEEK_W_CREDIT_VOLUME across the six weeks [€].
87. **WEEK_W_CREDIT_VOLUME_STD** Standard deviation of WEEK_W_CREDIT_VOLUME across the six weeks [€].

3.2.6 Savers feature

This feature is designed to capture the long-term saving behavior associated with the analyzed account over a six-month period, starting from the month currently under analysis. The computation is based on the application of a linear regression model to the sequence of monthly account balances observed during the last six months.

The regression line, estimated by minimizing the residual error with respect to the six data points, provides a quantitative representation of the temporal evolution of the account balance. The slope of this line constitutes the extracted feature, as it reflects the general trend of the account. A positive and steeper slope is indicative of an increasing balance, which may be interpreted as a stronger propensity toward saving. Meanwhile, a negative slope suggests a progressive reduction in the account balance. In instances where the account has been active for less than six months, the missing observations are completed by propagating the balance value of the earliest available month. This ensures consistency in the temporal dimension and preserves the comparability of the resulting feature across accounts. Formally, the extracted feature is defined as:

88. **SAVERS_COEFFICIENT** Slope coefficient of the regression line estimated on the sequence of monthly balances over the last six months [dimensionless].

3.2.7 High-Rotation features

These features are designed to capture financial behaviors characterized by rapid inflows and outflows of similar monetary amounts. For their computation, both the transactions recorded in the current month and those from the last D days of the previous month are considered, where D is defined as: $D = 45 - N$, with N being the number of days in the current month. This formulation ensures a fixed observation window of 45 consecutive days.

The analysis is restricted to transactions classified either as wire transfers or as cash operations. Within this transaction set, a rolling window of 5 days is defined, starting from the first of the considered days. The window is advanced by one day at a time, resulting in a total of 41 overlapping five-day windows.

For each window, a set of indicators is computed in order to quantify the degree of high rotation of the account:

- TOT_WINDOW: total transaction volume (absolute value) [€].
- RVCD: ratio between credit volume and debit volume (“Ratio of Volume between Credits and Debits”)[dimensionless].
- NUM_CREDITS: number of transactions with positive amounts [count].

- NUM_DEBITS: number of transactions with negative amounts [count].

Then the windows are filtered: only the windows where at least one credit and one debit transaction were performed and with an almost even volume of credits and debits (i.e., RVCD value close to 1 are) preserved. Formally, only the windows satisfying the following conditions are retained:

- NUM_CREDITS ≥ 1;
- NUM_DEBITS ≥ 1;
- RVCD $\in [0.8, 1.2]$

All other windows are discarded. If at least one window passes the filter, the account is classified as "High Rotational". On the filtered set of windows, the following features are subsequently computed:

89. **FILT_TOT_WINDOW_MIN** Minimum value of TOT_WINDOW across filtered windows [€].
90. **FILT_TOT_WINDOW_MAX** Maximum value of TOT_WINDOW across filtered windows [€].
91. **FILT_TOT_WINDOW_MEAN** Average value of TOT_WINDOW across filtered windows [€].
92. **FILT_TOT_WINDOW_MEDIAN** Median of TOT_WINDOW across filtered windows [€].
93. **FILT_TOT_WINDOW_STD** Standard deviation of TOT_WINDOW across filtered windows [€].
94. **FILT_TOT_WINDOW_COUNT** Number of filtered windows [count].

3.2.8 Exfiltration Features

Money muling[13] is a criminal practice in which individuals are exploited, knowingly or unknowingly, to transfer illicitly obtained funds on behalf of organized groups. This mechanism plays a critical role in the process of money laundering, as it allows criminals to obscure the origin of illegal profits and move them across different accounts, institutions, or even countries.

The "exfiltration" features are specifically designed to capture this type of criminal behavior, typically characterized by an inflow of funds followed shortly by an outflow of a similar amount, reflecting the mule's role in transferring and obscuring the origin of the proceeds.

For the computation of these features, only cash and wire operations are considered. For each credit transaction, the proportion of its amount that is offset within the subsequent five days by one or more debit transactions is evaluated. Such matching is allowed across categories, meaning that a credit executed via a wire transfer can be offset by a debit executed in cash, and vice versa. Once a debit (or part of it) has been used to offset a credit, the corresponding amount cannot be reused to offset other credits. This methodology enables the measurement of sequential fund movements within each account, providing an ordered analysis of inflows and outflows.

For analytical purposes, only transactions with an absolute value greater than or equal to 30€ are considered. Moreover, a debit is regarded as a relevant outgoing transaction ("exfiltration", abbreviated as "exfil") only if its amount corresponds to at least 20% of the credit being evaluated or if it is greater than or equal to 1 000€, irrespective of the credit size.

The resulting features are computed within a sliding five-days window over the last 45 days of account activity (as described in Section 3.2.7) and are defined as follows:

95. **EXFIL_CASH_TO_CASH** Total exfiltrated amount from cash credits to cash debits [€].

96. **EXFIL_CASH_TO_WIRE** Total exfiltrated amount from cash credits to wire debits [€].

97. **EXFIL_WIRE_TO_CASH** Total exfiltrated amount from wire credits to cash debits [€].

98. **EXFIL_WIRE_TO_WIRE** Total exfiltrated amount from wire credits to wire debits [€].

3.3 Evaluation metrics

In the context of this thesis and, generally, of financial crime detection, since fraudulent or criminal activities are typically rare compared to legitimate ones, the datasets are highly imbalanced, making traditional accuracy metrics insufficient to evaluate model performances. Furthermore, a reliable evaluation framework is essential to ensure that the models not only identify true anomalies but also minimize false alerts that could generate a high amount of detections, with the consequence of compromising the operational efficiency of the competence center.

For these reasons, evaluation metrics such as Precision, Recall, F1-Score and Normalized Discounted Cumulative Gain (NDCG) are used to assess the effectiveness of the models. Precision, Recall, and F1-Score are employed to quantify a model's ability to correctly identify suspicious accounts while avoiding misclassification of legitimate ones. Meanwhile, NDCG offers an additional perspective by evaluating the quality of the ranking produced by the model, particularly relevant when anomalies are ranked by their likelihood score.

3.3.1 Precision

Precision measures the proportion of correctly predicted positive instances out of all instances predicted as positive. It quantifies how many of the model's positive predictions are actually correct. High precision indicates a low rate of false positives. It is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.1}$$

where TP (True Positives) is the number of correctly predicted positive cases and FP (False Positives) is the number of negative cases incorrectly classified as positive.

3.3.2 Recall

Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances out of all actual positive instances. It assesses the model's ability to identify all relevant cases. A high recall value means that the model successfully captures most of the positive examples. The formula is:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.2}$$

where FN (False Negatives) are positive cases incorrectly classified as negative.

3.3.3 F1-Score

The F1-Score is the harmonic mean of Precision and Recall, providing a single metric that balances both. It is particularly useful when the dataset is imbalanced, as it penalizes models that achieve high precision but low recall, or vice versa. It is defined as:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})} \tag{3.3}$$

An F1-Score close to 1 indicates strong performance across both metrics, whereas lower values reveal an imbalance or poor classification capability.

3.3.4 Normalized Discounted Cumulative Gain (NDCG)

Normalized Discounted Cumulative Gain is a metric primarily used to evaluate ranking models by measuring how well the predicted order of items of a ranked list matches their true relevance. The NDCG metric is based on two components: the Discounted Cumulative Gain (DCG) and the Ideal DCG (IDCG). The DCG measures the quality of the predicted ranking by assigning higher importance to relevant items appearing at the top of the list, while using a logarithmic discounting factor to progressively penalize the contribution of items in the lower positions. The IDCG, instead, represents the maximum possible DCG obtained when all relevant items are ideally placed at the top of the ranking. The ratio between the two provides a normalized score between 0 and 1, making the metric comparable across different datasets or models.

In this thesis, the NDCG is computed according to the following implementation. Let $G = \{g_1, g_2, \dots, g_m\}$ be the set of ground truth anomalous items (i.e., accounts with the status of true positives), and let $R = [r_1, r_2, \dots, r_n]$ be the list of items ranked by the model according to their anomaly scores. The evaluation is performed considering the top-k elements of R. Each ranked element r_i is assigned a relevance score defined as:

$$\mathrm{rel}_i = \begin{cases} 1, & \text{if } r_i \in G \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

The Discounted Cumulative Gain (DCG) at position k is computed as:

$$\mathrm{DCG\ @\ k} = \sum_{i=1}^{k} \frac{\mathrm{rel}_i}{\log_2(i+1)} \tag{3.5}$$

The logarithmic denominator introduces the discount factor that reduces the contribution of relevant items appearing in lower ranks, thus rewarding models that correctly place relevant items (in this case, suspicious accounts) near the top of the list.

The Ideal Discounted Cumulative Gain (IDCG) represents the maximum possible DCG achievable if all relevant items were ranked at the top positions. It is defined as:

$$\mathrm{IDCG\ @\ k} = \sum_{i=1}^{\min(|G|,k)} \frac{1}{\log_2(i+1)} \tag{3.6}$$

Finally, the Normalized Discounted Cumulative Gain (NDCG) is obtained as the ratio between the actual and ideal gains:

$$\text{NDCG @ k} = \begin{cases} \dfrac{\text{DCG @ k}}{\text{IDCG @ k}}, & \text{if IDCG @ k} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

The resulting value of NDCG @ k lies in the range $[0, 1]$, where 1 indicates a perfect ranking (i.e., all truly anomalous accounts appear at the top of the model's ranking) and a value closer to 1 indicates that the predicted ranking closely matches the ideal one.

Together, these metrics provide a comprehensive evaluation framework. Precision, Recall, and F1-Score are focused on the classification nature of the problem, where the distinction between positive and negative classes matters, while NDCG is particularly suited for the ranking task, where the order of predictions affects utility. This combination of metrics was deemed appropriate to accurately assess model performance and ensure that it aligns with the intended goals.

3.4 Features pre-processing

Before training the models, a comprehensive pre-processing step is applied to ensure data consistency, handle missing values, and appropriately transform numerical and categorical features. The pre-processing strategy is designed to standardize heterogeneous features, mitigate skewness, and encode categorical information for compatibility with the machine learning algorithms. The overall pre-processing procedure can be summarized as follows.

3.4.1 Data cleaning

Missing values handling

Initially, the missing numeric values are imputed with the default value of -1 when the feature represents a standard deviation and with 0 otherwise. When missing, boolean values are always imputed with the default value of 0. This approach aims at preserving potentially meaningful distinctions between missing statistical features and missing count-based or transactional variables.

D23 clustering

Furthermore, domain-specific corrections are applied to the categorical D23 feature. Missing values are replaced with the default value "811", and a dimensionality reduction is performed through clustering of values, based on domain experts knowledge.

This procedure aims at grouping semantically similar and underrepresented categories, thereby simplifying the analysis and improving the model's generalization. The resulting semantic clusters of D23 codes are the following:

- **Banks cluster**: includes categories 310 (Banking system) and 712 (Rest of the World – Banking companies), both associated with banking activities.
- **Financial intermediaries cluster**: aggregates codes 311 (Financial intermediaries), 312 (Other financial intermediaries), 713 (Rest of the World – Financial companies), and 200 (Insurance companies and pension funds), representing entities operating in financial intermediation and insurance.
- **Healthcare cluster**: groups codes 103 (Public healthcare services) and 516 (Healthcare services), both belonging to the healthcare sector.
- **Administrations cluster**: includes codes 101 (Central administrations) and 102 (Local administrations), representing the institutional public sector.
- **"Others" category**: includes code 811 and missing values, used as a residual class for cases not attributable to structured groups.

3.4.2 Logarithmic transformation of skewed features

Highly skewed numerical variables are transformed using a signed logarithmic function, defined as: $x' = \mathrm{sign}(x) \cdot \log(1 + |x|)$. This operation is applied selectively to the features identified as non-normally distributed. The objective of this transformation is to reduce the influence of extreme values, improve the symmetry of the feature distributions and facilitate the convergence of distance-based and gradient-based algorithms: by compressing the magnitude of large observations, the models become less sensitive to outliers while preserving the relative ordering of data points.

3.4.3 Features scaling

After the logarithmic transformation, numerical features are standardized using the z-score normalization, defined as: $x'' = \frac{x'-\mu}{\sigma}$, where μ and σ represent the mean and standard deviation of the feature, respectively. This standardization ensures that all numerical variables contribute equally to the learning process, preventing features with large numerical ranges from dominating the optimization dynamics. Standard scaling is particularly relevant for algorithms such as One-Class SVM and Autoencoder which rely on distance-based computations in their latent space.

3.4.4 Categorical encoding

Categorical variables such as D23 and CUSTOMER_TYPE are encoded using one-hot encoding. The encoding process ignores unseen categories at inference time, in order to prevent errors due to data drift or the appearance of unexpected values in production environments.

The resulting encoded features form a sparse representation in which each category is mapped to an independent binary vector to enable compatibility with the input requirements of machine learning algorithms. This representation ensures that no artificial ordinal relationships are introduced among categorical features.

3.4.5 Pipeline integration

The pre-processing pipeline distinguishes three main groups of features:

- **Numerical features**, which undergo conditional imputation, logarithmic transformation and standardization. They represent a total of 92 features.
- **Binary features**, which are only imputed, as scaling or transformation could distort their semantic meaning. Only four features belong to this category.
- **Categorical features**, including only D23, which undergoes dimensionality reduction, and CUSTOMER_TYPE, which are encoded. After this process, 26 features represent this group.

This structured pre-processing design guarantees that all input data follow a uniform format and scale, mitigating the impact of outliers and categorical inconsistencies. Such standardization is crucial for unsupervised anomaly detection models, where subtle variations in feature distributions can significantly influence the learned representations and the overall detection performance.

3.5 Model configuration

3.5.1 Semi-supervised autoencoder

For the task addressed in this thesis, a semi-supervised autoencoder architecture was adopted. This model extends a traditional autoencoder by attaching a supervised head to its latent space, along with specific modifications to the training and prediction procedures, which will be detailed in this section. A schematic representation of the semi-supervised autoencoder is shown in Figure 3.9.

The reason behind this choice is to leverage the available ground truth information on accounts that have been reported and assessed as suspicious, in order to enhance the model's ability to rank the most anomalous accounts at the top positions. This approach combines supervised and unsupervised learning within a unified framework, making it particularly suitable for scenarios in which labeled data is only partially available or updated intermittently over time.

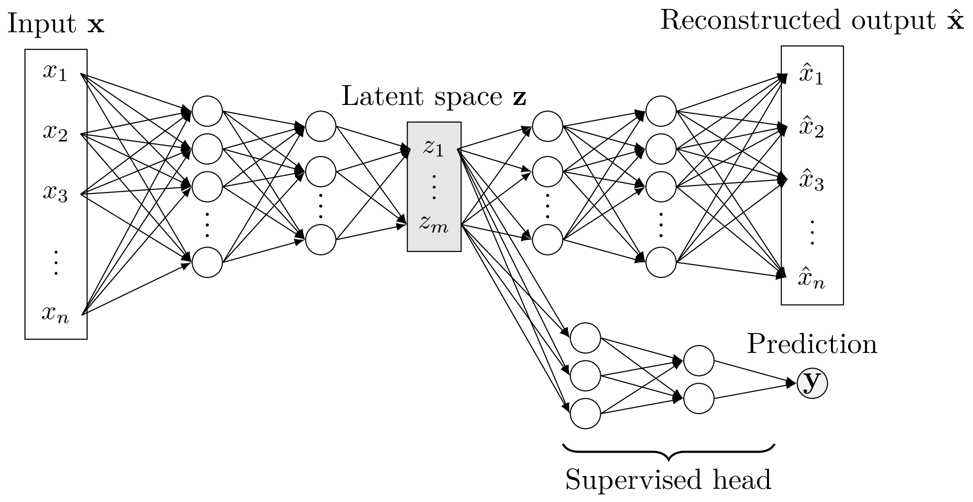


Figure 3.9: Semi-supervised autoencoder with a supervised head connected to the latent space.

3.5.2 Model architecture

The autoencoder consists of three main components: the encoder, the decoder and the supervised head. The encoder projects the input vector $\mathbf{x} \in \mathbb{R}^n$ into a lower-dimensional latent representation $\mathbf{z} \in \mathbb{R}^m$ through a sequence of n_{layers} fully connected linear layers. Each linear layer is followed by a ReLU (Rectified Linear Unit) activation function, defined as

$$\text{ReLU}(x) = \max(0, x) \tag{3.8}$$

which was chosen to introduce non-linearity while keeping the computational cost low.

The decoder architecture is symmetric with respect to the encoder. It reconstructs the input vector from the latent representation by reversing the sequence of transformations applied by the encoder. The last layer of the decoder does not include any activation function, allowing the reconstruction to remain unconstrained by non-linear transformations. The dimensions of the hidden layers follow a logarithmic progression: in the encoder, each layer's size decreases approximately as the base-2 logarithm of the previous one, from n to m. The decoder mirrors this structure, with the size of each hidden layer increasing logarithmically until it matches the original input dimension.

The dimensionality m of the latent space, also known as hidden size, the number n_{layers} of hidden layers, and the learning rate are all determined during the hyperparameter tuning phase, described later in Section 4.2.

3.5.3 Loss functions

The overall loss function combines multiple error components, each corresponding to a specific feature type:

- **Continuous loss**, computed as the Mean Squared Error (MSE) between the original and reconstructed values:

  $$\mathcal{L}_{\text{cont}} = \frac{1}{N_{\text{cont}}} \sum_{i=1}^{N_{\text{cont}}} (x_i - \hat{x}_i)^2 \tag{3.9}$$

  where N_{cont} is the number of continuous features.

- **Binary loss**, computed using the Binary Cross Entropy with logits:

  $$\mathcal{L}_{\text{bin}} = -\frac{1}{N_{\text{bin}}} \sum_{i=1}^{N_{\text{bin}}} [x_i \log(\sigma(\hat{x}_i)) + (1 - x_i) \log(1 - \sigma(\hat{x}_i))] \tag{3.10}$$

  where where N_{bin} is the number of binary features and $\sigma(\cdot)$ denotes the sigmoid function:

  $$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.11}$$

- **Categorical loss**, computed using the Cross-Entropy Loss over the one-hot encoded categorical features. Let C be the number of classes for a given feature. For each account i, the model outputs a logits vector $\hat{\mathbf{x}}_i = [\hat{x}_{i,1}, \hat{x}_{i,2}, \dots, \hat{x}_{i,C}]$, where each logit $\hat{x}_{i,c}$ represents the unnormalized confidence score that account i belongs to class c. The logits are converted into probabilities via the softmax function:

  $$\text{softmax}(\hat{x}_{i,c}) = \frac{e^{\hat{x}_{i,c}}}{\sum_{k=1}^{C} e^{\hat{x}_{i,k}}} \tag{3.12}$$

  Given the true label $\mathbf{x}_i$, represented as a one-hot encoded vector, and the predicted probability distribution $\mathbf{p}_i = \text{softmax}(\hat{\mathbf{x}}_i)$, the categorical loss is defined as:

  $$\mathcal{L}_{\text{cat}} = -\frac{1}{N_{\text{cat}}} \sum_{i=1}^{N_{\text{cat}}} \sum_{c=1}^{C} x_{i,c} \log(\mathbf{p}_{i,c}) \tag{3.13}$$

where N_{cat} is the number of categorical features.

The total unsupervised loss is defined as a weighted sum of the individual components:

$$\mathcal{L}_{\text{unsup}} = \alpha_{\text{bin}} \cdot \mathcal{L}_{\text{bin}} + \alpha_{\text{cat}} \cdot \mathcal{L}_{\text{cat}} + \mathcal{L}_{\text{cont}} \tag{3.14}$$

where the coefficients α_{bin} and α_{cat} control the relative contribution of the corresponding feature types. The values of these hyperparameters are determined during the model tuning phase, as detailed in section 4.3.

3.5.4 Supervised head

In parallel with the reconstruction pathway, the autoencoder includes a supervised head composed of one or more linear layers with ReLU activation and a final scalar output. This component learns a binary classification function that estimates the probability of an account being suspicious, using as input the latent representation produced by the encoder.

The supervised training phase can operate in two distinct modes:

- **Encoder-updating mode**, in which the gradient of the supervised loss is propagated through the encoder, allowing the latent representation to be influenced by the classification process;
- **Frozen-encoder mode**, in which the supervised head is trained independently, keeping the encoder weights fixed.

This flexibility is particularly valuable since the supervised head is updated only when new ground truth labels become available, typically at different time intervals compared to the unsupervised training of the autoencoder. Assessing which of the two training approaches is the best is one of the objectives of this thesis and it will be discussed in Section 4.5.

Eventually, the overall training loss integrates both the unsupervised and supervised components into a single loss function:

$$\mathcal{L}_{\text{total}} = \alpha_{\text{unsup}} \cdot \mathcal{L}_{\text{unsup}} + \alpha_{\text{sup}} \cdot \mathcal{L}_{\text{sup}} \tag{3.15}$$

where $\mathcal{L}_{\text{sup}}$ denotes the supervised binary classification loss, typically computed as Binary Cross Entropy, and α_{unsup} and α_{sup} are scalar weights controlling their relative influence which will be determined during the model tuning phase.

3.5.5 Incremental training

The architecture supports incremental training, allowing the model to reuse previously learned weights as initialization for subsequent training phases.

In production, the autoencoder is periodically retrained, typically at the end of each month, updating its latent representation based on newly available data while preserving previously acquired knowledge.

This design is particularly advantageous in operational environments where the data distribution may evolve gradually over time. By updating the model incrementally, it is possible to maintain the effectiveness without retraining from scratch at each iteration, thereby improving efficiency and reducing training costs.

3.6 Data processing pipeline

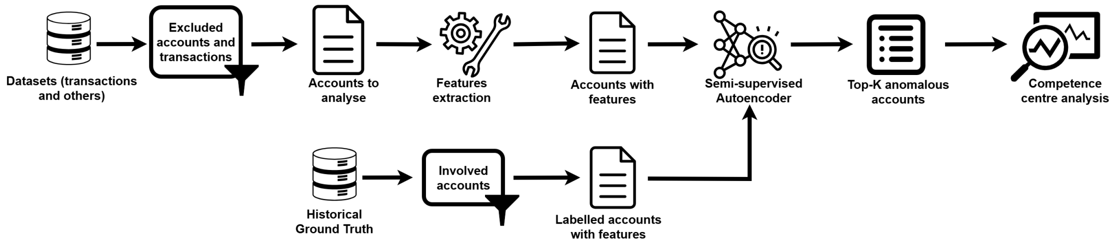

Datasets (transactions and others)
Excluded accounts and transactions
Accounts to analyse
Features extraction
Accounts with features
Semi-supervised Autoencoder
Top-K anomalous accounts
Competence centre analysis
Historical Ground Truth
Involved accounts
Labelled accounts with features

Figure 3.10: The complete data processing pipeline.

The aim of this section is to summarize all the previously described components and illustrate how their sequential execution constitutes the complete pipeline through which the data flow from their initial availability in the datasets to the model's final predictions, which ultimately generate reports on potentially suspicious customers.

As this project is being implemented for the first time, the current starting point (at the end of the first month of operation in the production environment) consists of the data becoming available in the datasets described in Section 3.1. These data are filtered and passed to the feature extraction algorithm, which models each customer's behavior through the 98 features detailed in Section 3.2. Subsequently, the features undergo the pre-processing steps outlined in Section 3.4, making them ready for model ingestion.

The model then processes the accounts and produces a ranking: each user is assigned an anomaly score derived from both the reconstruction error and the supervised anomaly score. Users are sorted in descending order according to this score, and the top K most anomalous cases are reported to the competence center, where they are analyzed by human experts with access to non-anonymized data and additional contextual information (for instance, all the accounts associated with a customer flagged as suspicious).

After the competence center's review, feedback is provided and incorporated into the ground truth table described in Section 3.1.7, where suspicious accounts are labeled as either true positives or false positives. Each update to the ground truth dataset triggers a retraining of the supervised head which can, optionally, influence the encoder.

The pipeline then continues in a cyclical fashion: at the end of each month, following the update of the transactions table, the semi-supervised model is retrained to update its unsupervised component, while the supervised head is updated whenever new ground truth labels are received.

Chapter 4

Experimental results

Given the large volume of data and the heterogeneity observed both in the population distribution and in the transactional patterns of the users, the training strategy for the autoencoder was designed by distinguishing between two cases: physical persons and legal entities. As previously discussed in Section 3.1.5, this issue motivated the decision to partition the users according to the value of the D23 feature, resulting in two separate and well-defined groups:

- **Physical persons**, defined as all users for whom D23 takes the value '600'. This code encompasses accounts whose transactional behavior is comparable to that of individual human users;
- **Legal entities**, comprising all remaining D23 codes. This group includes, in broader terms, organizations of various types, each one potentially exhibiting distinct behavioral patterns.

Training a single autoencoder to model and reconstruct such a heterogeneous set of accounts would have been detrimental to performance. For this reason, the decision was made to train two separate autoencoder models, one dedicated to each of the mentioned categories. Throughout this chapter, a detailed description of all experiments is presented. Every experiment was conducted for each model independently, which we refer to as 'Physical Person' and 'Legal Entity' from this point onwards. On a monthly basis, the dataset includes approximately 6 million accounts labeled as physical persons and around 1.5 million accounts associated with legal entities, reflecting a significant imbalance in population size between the two categories.

All analyses presented in this chapter were carried out on data from October 2024. This month was chosen as it represents a particularly stable and representative period: unlike summer months, it is not affected by seasonal holidays or atypical spending patterns, and unlike December, it is not influenced by Christmas-related peaks in transactions. October also typically lacks major national holidays or exceptional events that might distort users' transactional behavior, making it a suitable reference point for model evaluation.

The goal of these experiments is to improve both the learning and classification capabilities of the autoencoder models. Performance is assessed using the evaluation metrics introduced in Section 3.3. In particular, the evaluation focuses on the top–K positions of the ranking, with $K \in \{1000, 2500, 5000, 10000\}$. Although multiple values of K are examined, the threshold of $K = 5000$ is especially relevant, as it reflects the stated benchmark from the bank. Ultimately, the objective is to maximize each model's ability to position the most suspicious cases at the top of the ranking. These flagged accounts are subsequently forwarded to the competence center, where specialized analysts, with access to non-anonymized data and additional contextual information, conduct an in-depth assessment to determine whether the users are involved in financial crime.

4.1 Incremental versus single-shot training

This experiment investigates two alternative training strategies for the autoencoder: a single-shot approach and an incremental approach. In the single-shot configuration, the model is trained independently for each month, reinitializing all weights at random before every new training cycle. In contrast, the incremental strategy initializes the model for month t using the weights learned at month $t - 1$, thereby enabling the model to accumulate information across time.

The evaluation was conducted separately on the top-5000 accounts in the ranking for three account groups: Legal Entities, Physical Persons, and the entire population. This design makes it possible to assess not only the effect of the training strategy but also the impact of population heterogeneity on model performance.

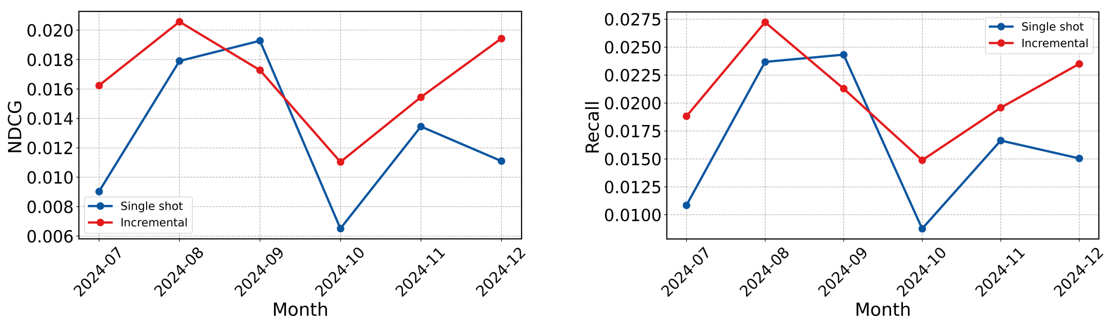


Figure 4.1: NDCG and Recall at K = 5000 for the entire population.

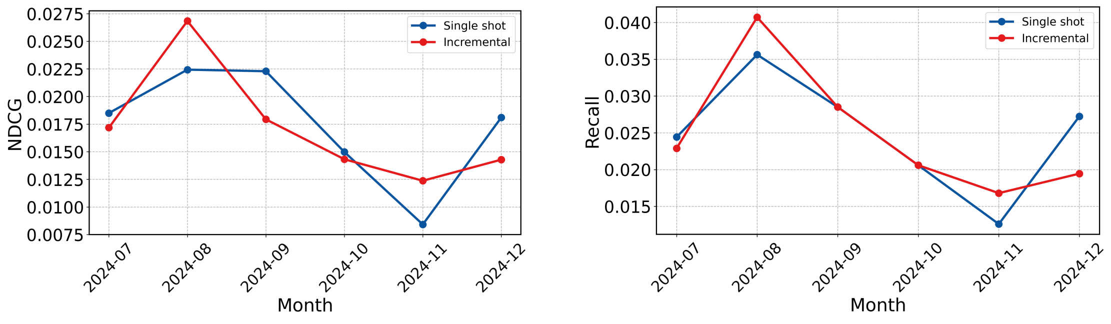


Figure 4.2: NDCG and Recall at K = 5000 for Legal Entities.

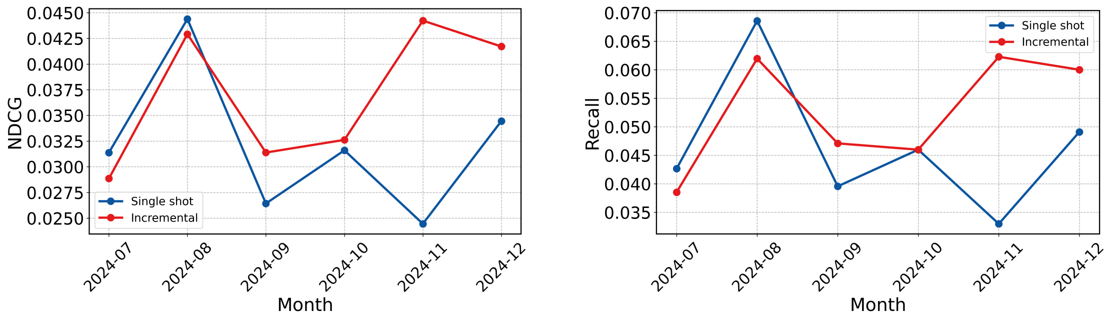


Figure 4.3: NDCG and Recall at K = 5000 for Physical Persons.

The parameters used for model training were inherited from an earlier configuration and are therefore not optimal; this limitation will be addressed in Section 4.2. All models were initially trained using the single-shot strategy on June 2024, and the resulting model served as the starting point for the incremental training performed in the subsequent months.

The results lead to two main conclusions. First, the incremental strategy consistently outperforms the single-shot approach: it achieves higher overall performance and significantly reduces training time, confirming the benefits of leveraging previously learned representations.

Second, training a single model jointly on Legal Entities and Physical Persons results in degraded performance compared with training two specialised models. The strong heterogeneity between the groups, both in terms of behavioural patterns and in the relative number of accounts, negatively affects the global model, reinforcing the importance of maintaining separate models for the two populations.

4.2 Hyper-parameter tuning of the unsupervised Autoencoders

The first stage of the experimental process consisted of tuning some of the hyperparameters only on the unsupervised part of the model, independently for the Physical Persons and Legal Entities autoencoders. For both, a grid search was carried out over the following set of hyperparameter values:

$$\begin{aligned} \text{hidden size} &\in \{4,\, 8,\, 16\}, \\ \text{number of layers} &\in \{2,\, 4\}, \\ \text{learning rate} &\in \{10^{-3},\, 10^{-4},\, 10^{-5}\}. \end{aligned} \tag{4.1}$$

A total of 18 configurations were trained for 200 epochs with a batch size of 1024 accounts, without applying incremental training at this stage. For each experiment, performance was evaluated using the NDCG and recall metrics, with a focus on the values at $k = 5000$ and $k = 10000$, as these thresholds are especially relevant for the operational constraints already discussed. The figures reported in this section display, for each group and metric, the top three configurations achieving the highest values at these cutoffs.

The results of the grid search reveal substantial differences between the two user categories. For the Legal Entities model, the best-performing configuration consisted of a latent size of 16, four layers, and a learning rate of 10^{-3}, suggesting the need for a latent representation with higher dimensionality in order to capture the more diversified behaviors characteristic of this group. Conversely, the Physical persons model achieved its best results with a latent size of 4, four layers, and a learning rate of 10^{-5}, reflecting the more homogeneous structure of transactional patterns within this population. In Figures 4.4 and 4.5 the best configuration is highlited with respect to the other ones.

These findings confirm the necessity to tailor the architecture and the learning dynamics of each autoencoder to the statistical properties of the corresponding accounts group, reinforcing the reason for training two separate models.

Beyond NDCG and recall, additional metrics were considered during the evaluation phase, including precision and their corresponding variants computed on false positives. Although false positives do not directly contribute to the identification of confirmed suspicious accounts, they remain of interest in this context: a high-quality model should also be capable of detecting accounts that previously triggered human suspicion, even when they were not ultimately confirmed as anomalous. For the sake of brevity, and given their lower relevance to the core analysis, these supplementary metrics are reported in Appendix A, Section A.1.

A further set of analyses examined the stability of each hyperparameter configuration. Specifically, for a fixed value of one hyperparameter, all combinations

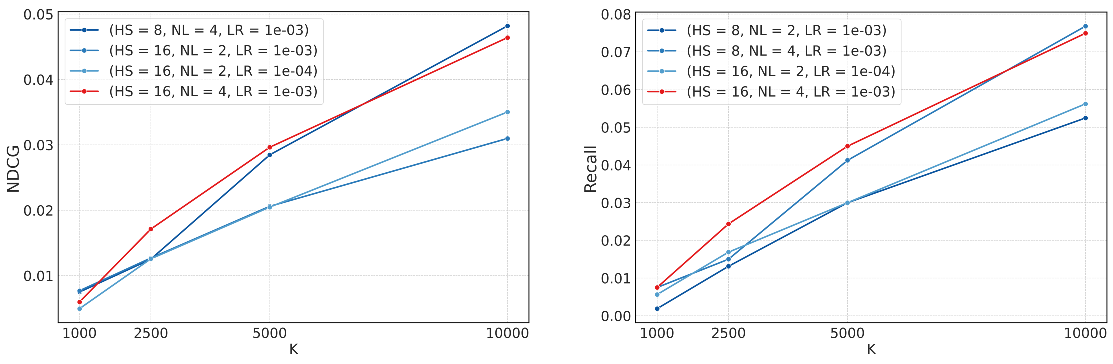


Figure 4.4: NDCG and Recall at K for Legal Entities.

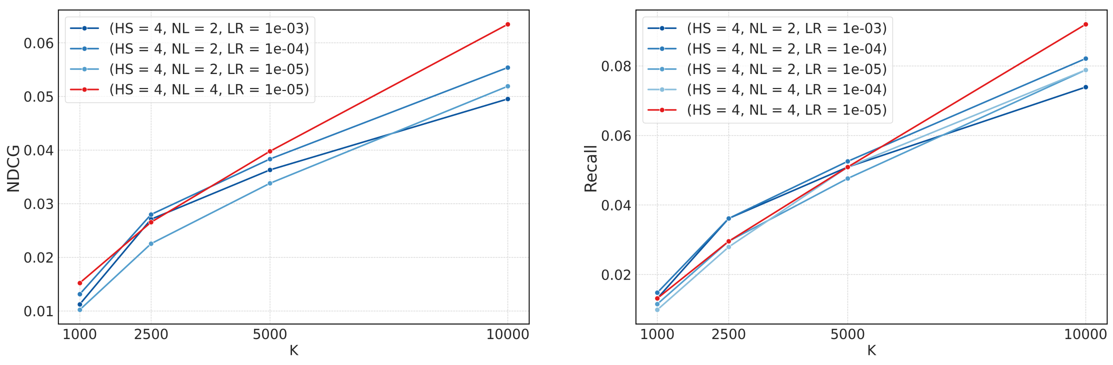


Figure 4.5: NDCG and Recall at K for Physical Persons.

of the remaining two were evaluated to assess whether a given choice consistently yielded strong performance across different settings. A hyperparameter value can be considered reliable only when its effectiveness holds regardless of the configuration. The results of this robustness assessment confirmed that the configurations identified as optimal in the grid search maintain high performance across all tested variations. Consequently, these configurations can be adopted as the baseline for the unsupervised component of the model moving forward. The detailed results of this stability analysis are also provided in Appendix A, Section A.1.

4.3 Loss coefficients tuning

As shown in Equation 3.14, two linear coefficients, α_{bin} and α_{cat}, are associated with the binary and categorical loss components in the unsupervised loss. In the previous experiment on hyperparameter tuning of the unsupervised Autoencoders

(Section 4.2), both coefficients associated with the binary loss terms were fixed to 1.

A subsequent analysis was conducted, focused on determining more suitable values for these coefficients. For optimisation purposes, the fine-tuning phase targeted only the binary-loss coefficient, while keeping $\alpha_{\text{cat}} = 1$. Using the five best-performing configurations identified in Section 4.2, separately for legal entities and physical persons, the Autoencoders were retrained by varying the binary-loss weight α_{bin} across the set $\{0.25, 0.50, 0.75, 1\}$.

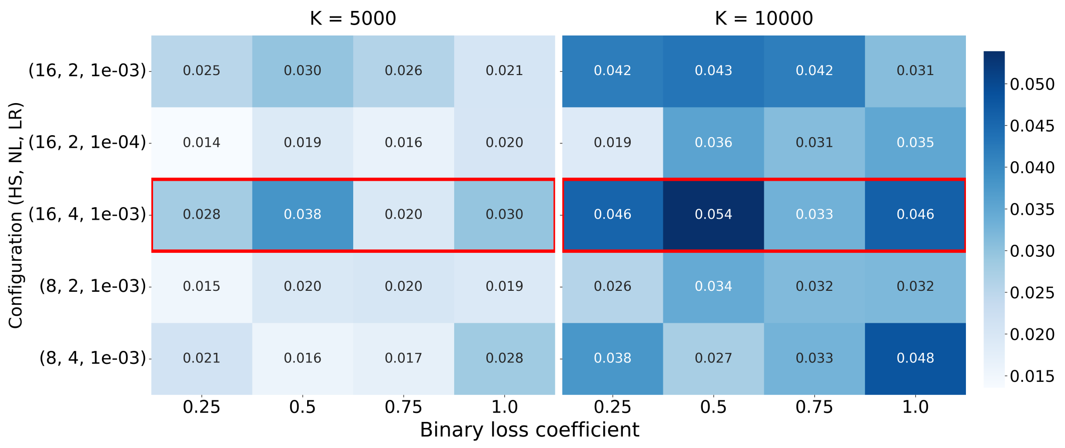


Figure 4.6: NDCG for the best unsupervised configurations at different values of α_{bin} and K for Legal Entities.

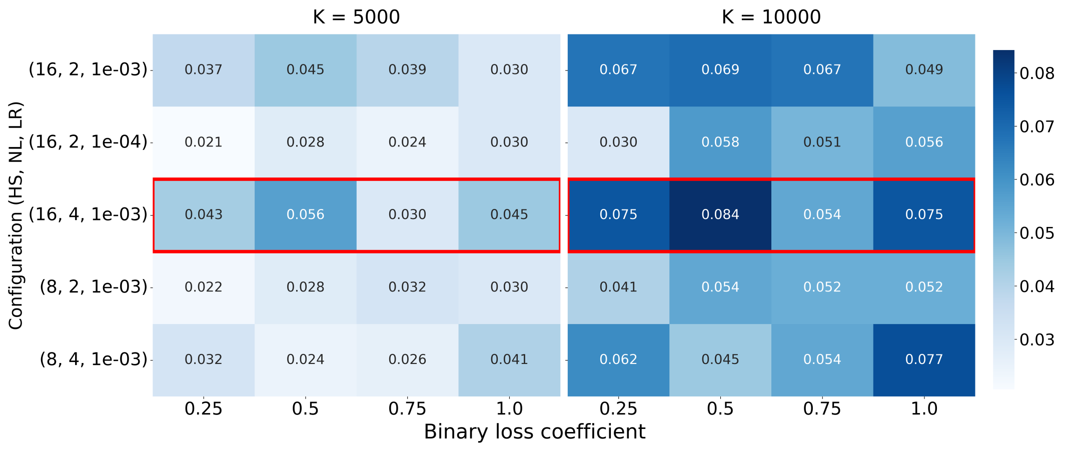


Figure 4.7: Recall for the best unsupervised configurations at different values of α_{bin} and K for Legal Entities.

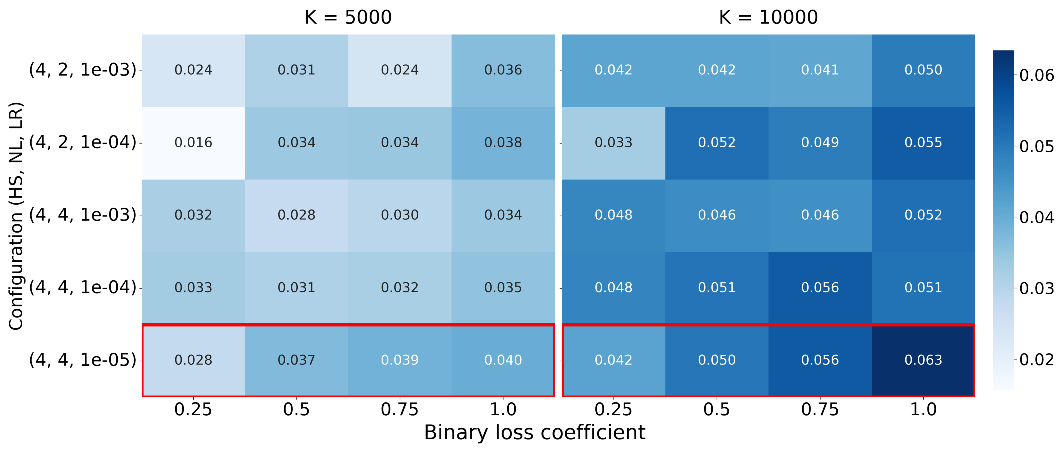


Figure 4.8: NDCG for the best unsupervised configurations at different values of α_{bin} and K for Physical Persons.

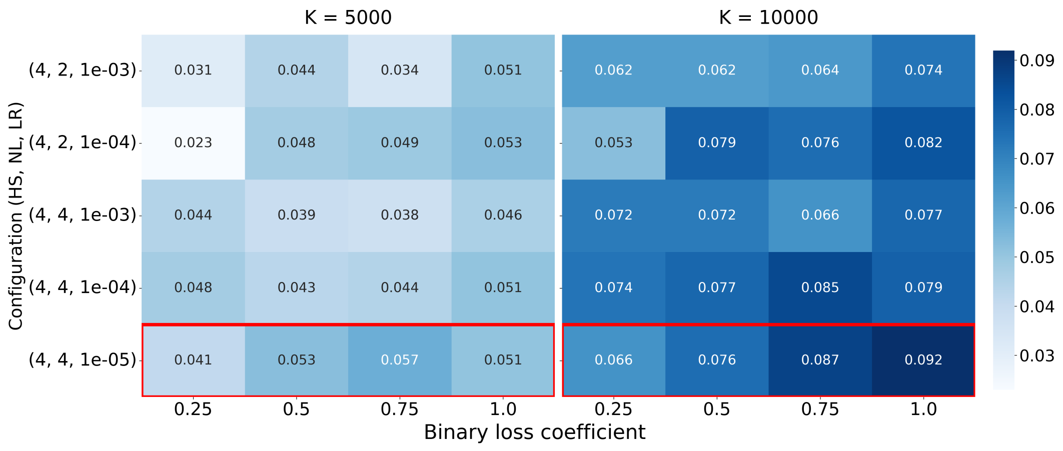


Figure 4.9: Recall for the best unsupervised configurations at different values of α_{bin} and K for Physical Persons.

For both groups, an initial observation is that the best configuration, which are highlighted in the heatmaps, consistently outperforms the others regardless of the value of α_{bin}. This provides further confirmation of the validity of the selected hyperparameters.

As illustrated in the heatmaps, when analysing the behaviour of the metrics as α_{bin} varies for $k = 5000$ and $k = 10000$, the highest performance—both in terms of NDCG and recall—is obtained with $\alpha_{\text{bin}} = 1$ for the Physical Persons and $\alpha_{\text{bin}} = 0.5$ for the Legal Entities.

4.4 Seed variation experiment

This experiment evaluates the sensitivity of the results to variations in the random seed (or random state). All configurations tested so far used a fixed seed value of 42; in this analysis, seeds from 1 to 7 were explored. The experiment was conducted on June data only, in order to avoid repeatedly analysing October and thus reduce the risk of overfitting to a specific month.

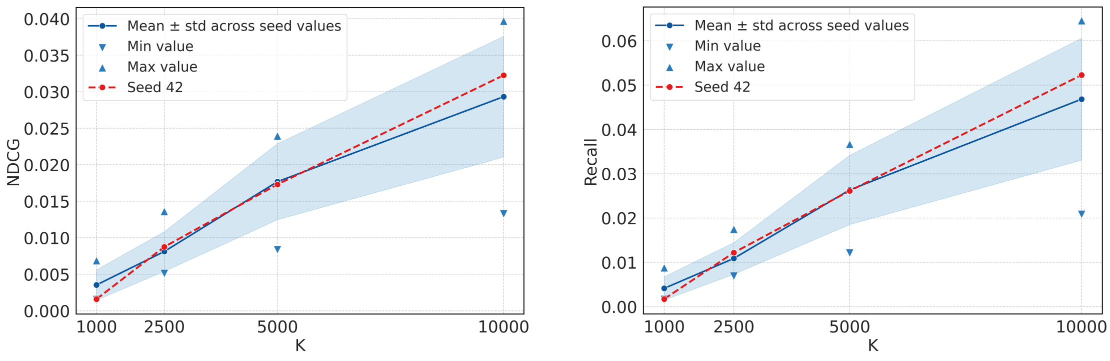


Figure 4.10: NDCG and Recall at K through different random seed values for Legal Entities.

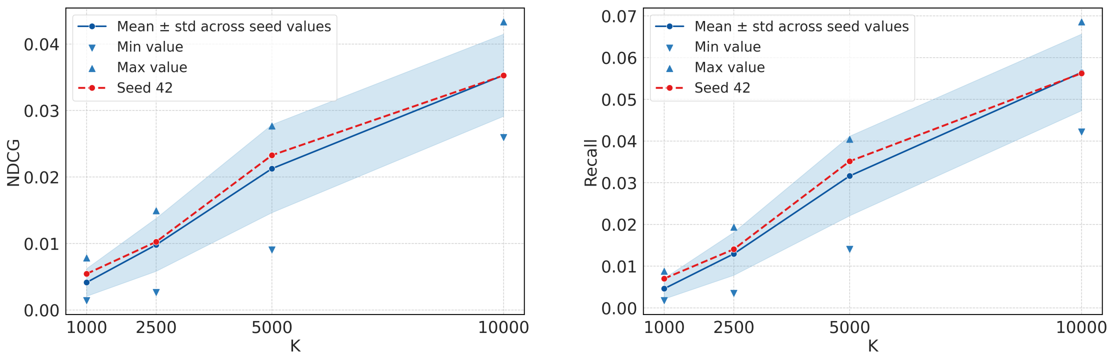


Figure 4.11: NDCG and Recall at K through different random seed values for Physical Persons.

The first step consisted in evaluating the performance of each model configuration. The two resulting plots report the mean behaviour across the seven seeds, together with the corresponding standard deviation and the minimum and maximum value achieved at each K. As shown in the Figures 4.10 and 4.11, performance remains stable for both Legal Entities and Physical Persons: the variations induced by changing the seed are small and do not deviate meaningfully from the baseline obtained with seed = 42, which is highlighted.

Subsequently, for $K = 5000$, the stability of the ranking produced by the seven runs (i.e., seed = $\{1, 2, \ldots, 7\}$) was analysed. Specifically, for each account, the number of times it appears in the top–5000 across the seven runs (a frequency between 1 and 7) was counted and visualised through the Empirical Cumulative Distribution Function (ECDF) of these frequencies, as shown in figures 4.12 and 4.13. The total number of distinct accounts who appear at least once in the top–5000 of any run, i.e., the size of the union of the seven rankings was also computed.

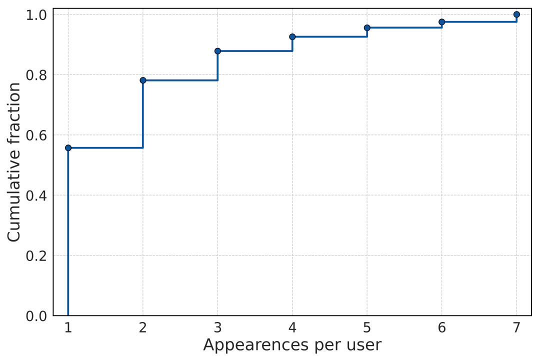


Figure 4.12: ECDF of the frequency of each account appearing in the top-5000 of the ranking for Legal Entities.

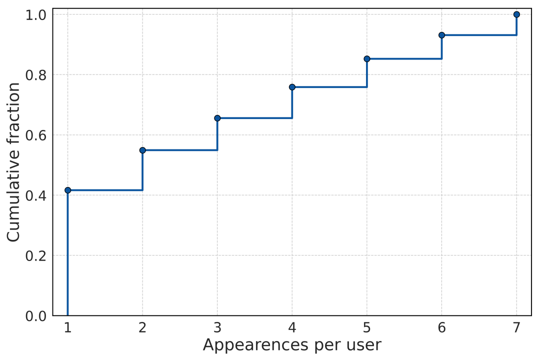


Figure 4.13: ECDF of the frequency of each account appearing in the top-5000 of the ranking for Physical Persons.

The same analysis was then repeated for the users appearing in the ground truth, distinguishing between true positives (TP) and false positives (FP). For clarity and completeness, the corresponding ECDF plots are provided in Appendix A, Section A.2.

For Legal Entities, the union of the seven top–5000 rankings comprises 18,165 distinct accounts, of which 451 (2.5%) appear consistently across all runs. Within the ground truth, 457 accounts are present overall (48 TPs and 409 FPs), and among the persistent users, 50 belong to the ground truth (2 TPs and 48 FPs).

For Physical Persons, the union contains 12,340 distinct accounts, with 849 of them (6.8%) appearing in all seven rankings. Of the 184 ground-truth accounts detected in at least one run (44 TPs and 140 FPs), only 9 are persistent across all seeds (1 TP and 8 FPs).

Overall, these results indicate that seed variation introduces limited instability in the top–5000 rankings. Although a non-negligible number of accounts appear intermittently across runs, only a small fraction of users is consistently present in all top–5000 lists, and most of these persistent users are false positives. The ECDF curves further confirm that the majority of accounts appear only a few times across seeds, while a comparatively small subset exhibits high cross-seed consistency. These findings suggest that, while seed variation affects the exact composition of the top-ranked users, the magnitude of these fluctuations remains moderate. To complement this analysis and to obtain a more quantitative measure of ranking stability across seeds, a correlation-based evaluation was then conducted.

Pearson correlation analysis

To further assess the stability of the ranking with respect to seed variation, Pearson's correlation coefficient was computed between every pair of rankings derived from different seeds. Pearson's coefficient measures the strength of the linear relationship between two numerical vectors; in this context, the vectors correspond to the ranking scores assigned to users by two independent model runs. A value of $r = 1$ indicates perfect alignment between the rankings, while $r = 0$ indicates no linear relationship. Negative values would imply an inverse ordering, which is not expected in this setting. High values of r thus correspond directly to high ranking stability.

For each of the $\binom{7}{2} = 21$ seed pairs, Pearson's r was computed, together with its statistical significance (p-value). Following standard practice, correlations with $p \geq 0.01$ were deemed unreliable and were therefore set to zero; the number of discarded correlations was also recorded. The final stability indicators are the mean and variance of the 21 resulting r values.

For Legal Entities, the mean Pearson correlation is 0.848 with a variance of 0.0075. For Physical Persons, the mean correlation is 0.837 with a variance of 0.0016. Importantly, no pair exceeded the $p \geq 0.01$ threshold, indicating that all computed correlations are statistically meaningful. These results demonstrate a very strong linear consistency between rankings generated with different seeds, confirming that the overall ranking structure is robust to stochastic variation.

A complementary perspective is provided by analysing the intersection ratio between seed pairs, defined as:

$$\text{ratio}_{i,j} = \frac{\text{topK}(i) \cap \text{topK}(j)}{K}.$$

This metric captures the degree of agreement specifically in the top portion of the ranking. For Physical Persons, the mean intersection ratio is 0.545 (variance 0.0068), indicating that two runs typically share about 54% of their top–5000 accounts. For Legal Entities, the mean ratio is 0.331 (variance 0.0055), reflecting a greater variability in the very top-ranked segment: an expected outcome, as intersection ratios are sensitive to small perturbations within the upper tail, whereas Pearson correlations capture global linear alignment.

Taken together, these analyses confirm that while seed variation can influence which specific accounts appear in the highest-ranked subset, the global ranking structure remains highly stable. Consequently, and in the absence of a clearly superior alternative, the original seed value of 42 was retained for the remainder of the study.

4.5 Joint vs Sequential Training of the Semi-Supervised Autoencoder

This experiment represents the first evaluation of the supervised head of the semi-supervised autoencoder. The supervised head is trained on a monthly basis. To simulate an incremental learning scenario, all accounts reported in previous months are collected from the ground-truth dataset, their feature representations are extracted, and these are passed through the encoder and then through the supervised head for training. All ground-truth accounts (both true positives and false positives) are assigned label 1, whereas negative samples are obtained via random sampling from the set of accounts not present in the ground truth. The sampling ratio is controlled by the parameter 'pos_to_neg_ratio', set to 0.5, ensuring a balanced dataset with an equal number of positive and negative samples.

Another key parameter is α, which determines the proportion of supervised samples within each training batch. Here, $\alpha = 0.3$, meaning that in every batch, 30% of the data points are used by the supervised head (that is, they are labeled), while the remaining 70% contribute to the unsupervised reconstruction loss.

Two training strategies were evaluated:

- **Sequential training**: the supervised head is trained after the unsupervised autoencoder has been fully trained. During this phase, the unsupervised encoder–decoder remains frozen, preventing the supervised head from modifying encoder weights. The hyperparameters used for the unsupervised component correspond to those obtained in the previous hyperparameter-tuning stage described in Section 4.2.

- **Joint training**: both the supervised and unsupervised heads are trained simultaneously. In this configuration, gradients from the supervised head are allowed to propagate through the encoder, altering its weights. The unsupervised branch uses the same set of tuned hyperparameters as in the sequential setup.

The resulting plots compare the performance of the two approaches across multiple configurations. Rankings labelled "Unsupervised" correspond to those generated using the reconstruction error from the unsupervised head, while rankings labelled "Supervised" correspond to the output of the supervised classifier.

The results highlight two clear findings. First, the supervised head trained jointly with the unsupervised component exhibits substantially better performance than

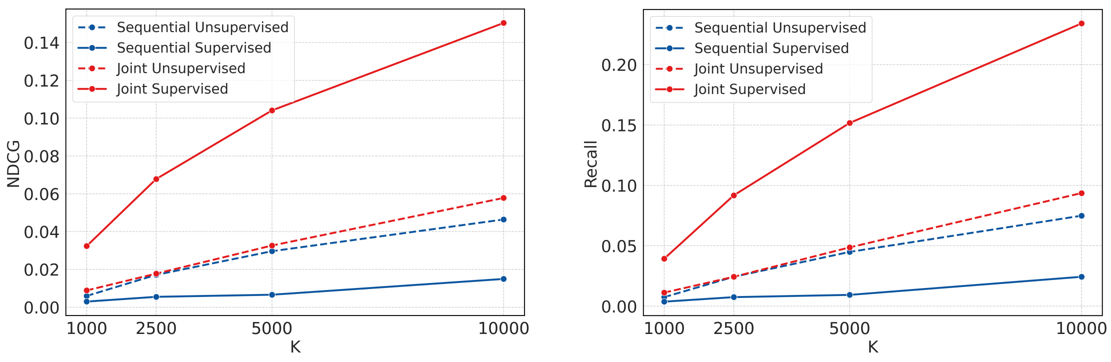


Figure 4.14: NDCG and Recall at K for Legal Entities.

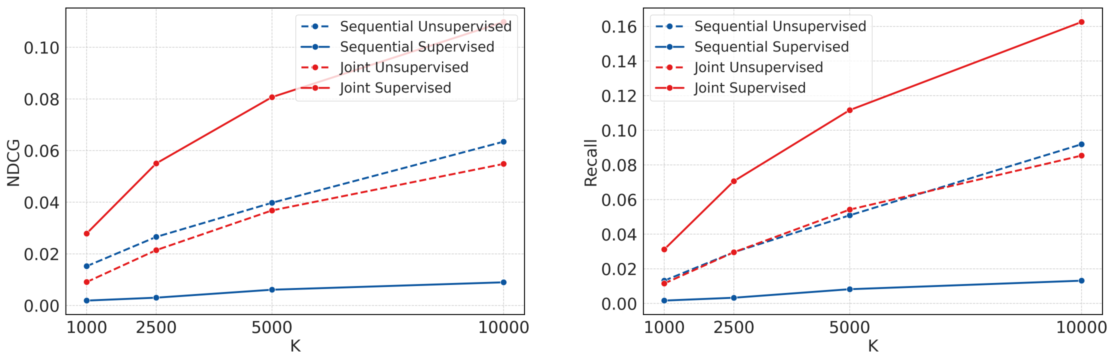


Figure 4.15: NDCG and Recall at K for Physical Persons.

when trained sequentially. Allowing the supervised loss to influence the encoder during training leads to latent representations that are more discriminative for the detection task, thereby enhancing downstream ranking performance. In contrast, the unsupervised head shows only minimal differences between the two settings. This is expected, as the reconstruction objective remains structurally unchanged: even when trained jointly, its contribution dominates 70% of each batch, and the reconstruction loss is largely insensitive to the relatively sparse supervised signal.

Second, when trained jointly, the supervised head significantly outperforms the unsupervised ranking. This confirms that the semi-supervised architecture provides a clear advantage over relying solely on a classical unsupervised autoencoder: the supervised signal effectively guides the encoder towards learning meaningful, task-relevant structures, leading to a considerable improvement in detection capability.

4.6 Benchmarking unsupervised baselines against the autoencoder models

To assess the relative performance of the proposed Autoencoder architectures, two of the classical unsupervised anomaly detection models described in Section 3 (One-Class SVM and Isolation Forest) were selected as baselines. Each baseline model was tuned and evaluated separately on the Legal Entities and Physical Persons datasets. Local Outlier Factor (LOF) was initially considered as a third baseline; however, it was discarded because the algorithm does not scale to the dimensionality and volume of the data, failing to converge within feasible memory and time constraints.

Hyperparameter tuning was performed via an exhaustive grid search. The One-Class SVM was evaluated across 32 configurations, while the Isolation Forest was evaluated across 72 configurations. These models were trained on the month of June 2024 and the following six months were used as test sets. For both models, the preprocessing steps were aligned with those used for the Autoencoder, as described in Section 3.4. Additional steps, such as dimensionality reduction, correlation-based feature selection and PCA, were optionally applied depending on the model configuration. When enabled, PCA was applied to reduce the feature space to the number of components explaining 90% of the variance.

The performance comparison focuses on NDCG and Recall evaluated over the top-5000 ranked accounts, contrasting each baseline model with the unsupervised Autoencoder and with the unsupervised output of the semi-supervised Autoencoder. The resulting trends indicate that the baseline methods achieve performance broadly comparable to the unsupervised Autoencoders, with notable differences depending on the dataset:

- **Legal Entities**: both the unsupervised and semi-supervised Autoencoders clearly outperform the baseline models.
- **Physical Persons:** the baseline models retrieve a larger number of true positives but also a disproportionately high number of false positives, as confirmed by the dedicated metric plots. Consequently, the Autoencoder models exhibit lower values of both metrics in this setting, as they perform more novelty detection.

Among the baselines, Isolation Forest is observed to perform relatively well. This is consistent with its ability to isolate anomalies along individual features; when highly specialised domain-specific features are present, like High-Rotation 3.2.7 or Exfiltration 3.2.8 features, Isolation Forest can effectively exploit them. In contrast, the Autoencoder redistributes representational importance through learned weights in the encoder, rather than treating each feature independently.

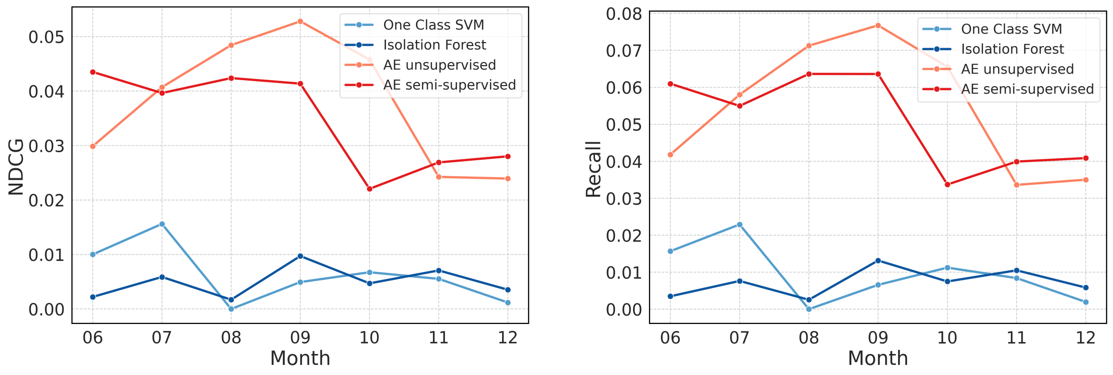


Figure 4.16: NDCG and Recall across months for Legal Entities.

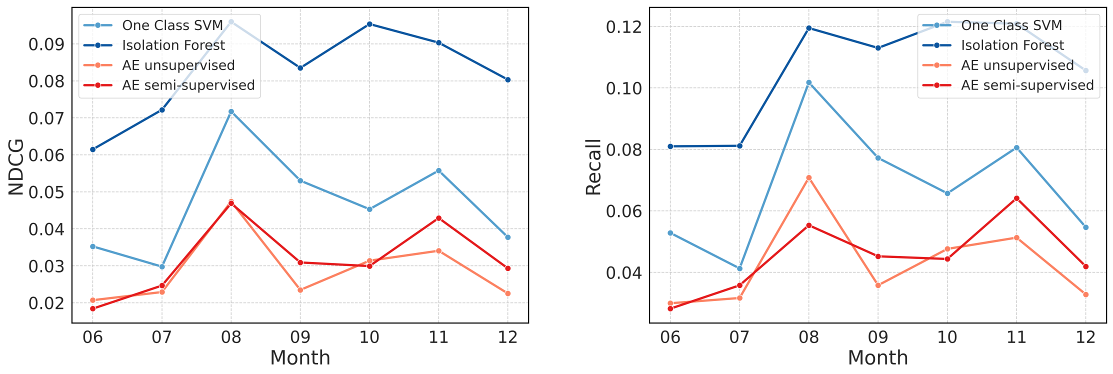
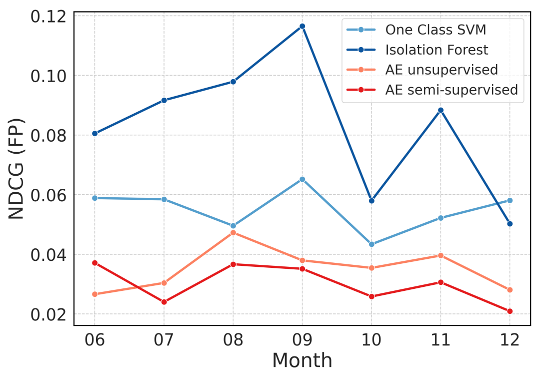


Figure 4.17: NDCG and Recall across months for Physical Persons.

Among the baselines, Isolation Forest is observed to perform relatively well. This is consistent with its ability to isolate anomalies along individual features; when highly specialised domain-specific features are present, such as the High-Rotation features discussed in Section 3.2.7 or the Exfiltration features introduced in Section 3.2.8, Isolation Forest can effectively exploit these dimensions. In contrast, the Autoencoder distributes representational importance across multiple features through learned weights in the encoder, rather than treating each feature independently.

Despite the reasonable performance of One-Class SVM and Isolation Forest, the Autoencoder models remain preferable for two fundamental reasons. First, the baseline models cannot support incremental training and therefore cannot leverage temporal continuity in the data.

Second, they are unable to exploit historical ground-truth data, and thus cannot



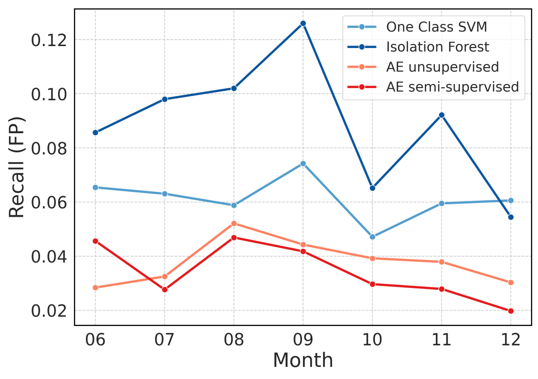


Figure 4.18: NDCG and Recall on False Positives across months for Physical Persons.

incorporate a supervised training signal. These limitations prevent them from reaching the performance levels achieved by the semi-supervised Autoencoder, whose supervised head guides the encoder toward more discriminative representations.

4.7 Benchmarking supervised classifiers on latent representations

This experiment evaluates the discriminative quality of the latent spaces produced by the two Autoencoder architectures: the unsupervised Autoencoder, whose latent representation is shaped solely by reconstruction objectives, and the semi-supervised Autoencoder, in which the supervised head actively influences the structure of the latent space. The goal is to assess whether the representations learned in the semi-supervised setting provide a measurable advantage when used as input to downstream classification models.

A suite of shallow learning classifiers was trained on top of the respective latent embeddings: K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machine (SVM), Decision Tree, Random Forest and a feed-forward neural network (FFNN) mirroring the architecture of the supervised head of the semi-supervised Autoencoder (a single linear layer with input dimensionality equal to the latent size and a one-dimensional output). Model training and hyperparameter optimisation were performed on a balanced dataset comprising 50% ground-truth accounts (from January to May, approximately 15 000 accounts) and 50% accounts sampled at random from the June population.

After obtaining the latent representations from the unsupervised Autoencoder trained in single-shot mode on June, an exhaustive hyperparameter search was carried out for each classifier. The configuration maximising the Recall was selected for both Legal Entities and Physical Persons, which were treated independently in all experiments.

Each tuned classifier was then applied to the full June population (approximately 6 million accounts for Physical Persons and 1.2 million for Legal Entities), using the predict_proba() function to generate a continuous anomaly score. This score enables a direct comparation between this ranking and the one produced by the semi-supervised Autoencoder.

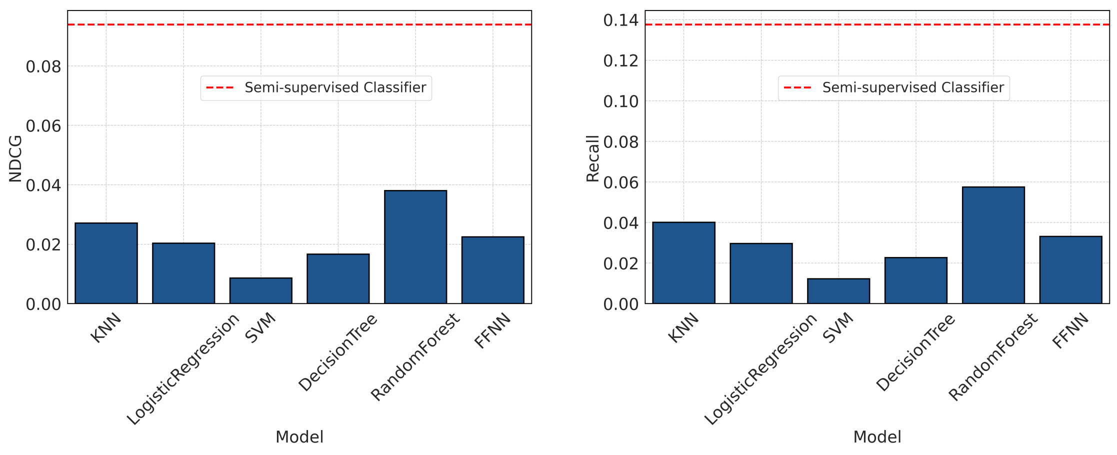


Figure 4.19: NDCG and Recall at K = 5000 for Legal Entities.

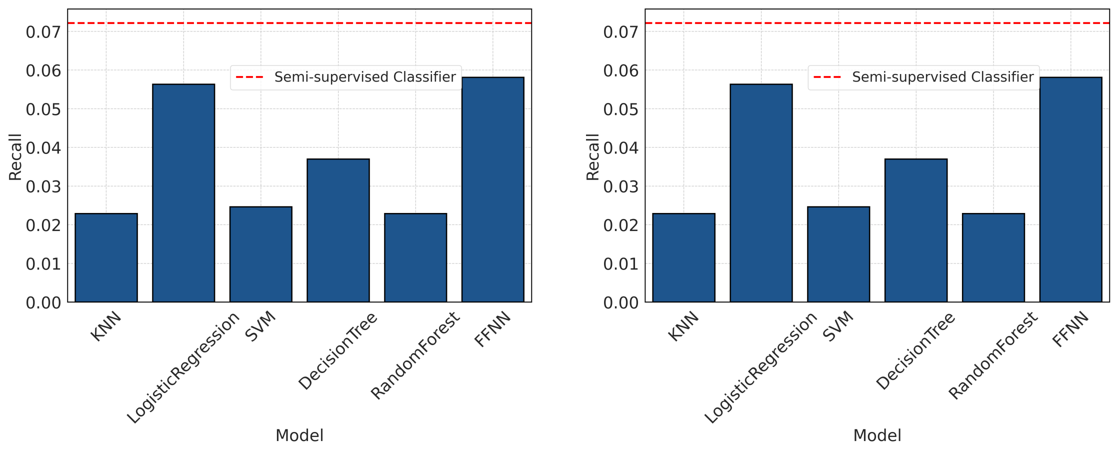


Figure 4.20: NDCG and Recall at K = 5000 for Physical Persons.

The resulting performance metrics show that the semi-supervised Autoencoder consistently outperforms the supervised classifiers trained on the unsupervised latent space representation, as illustrated by the red dashed line in Figure 4.19 and 4.20. These results reinforce the conclusion that the training of the supervised head in the semi-supervised Autoencoder shapes a more informative and discriminative latent space, thereby justifying its adoption in subsequent analyses.

4.8 Evaluation of a fully supervised baseline

In this experiment, the goal is to compare the semi-supervised pipeline of the Autoencoder with a fully supervised alternative. The same data used in the previous experiment were retained; however, instead of relying on the latent representations produced by the Autoencoder, the models were trained directly on the 98 original features.

The preprocessing pipeline was aligned with the one adopted for the Autoencoder. First, the standard preprocessing steps were applied, followed by correlation-based feature reduction, where features with pairwise correlation greater than 0.9 were removed. Principal Component Analysis (PCA) was then applied, retaining the components that explained 90% of the total variance. After these transformations, the same set of classifiers: K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machine (SVM), Decision Tree, Random Forest, and a feed-forward neural network (FFNN) was trained on the same balanced dataset used in the previous experiment (approximately 30 000 accounts selected using the same sampling strategy described in Section 4.7), and subsequently used to perform inference on the June population.

A first relevant observation is that, for both Legal Entities and Physical Persons, the combination of preprocessing, feature reduction and PCA resulted in 26 principal components explaining approximately 91% of the variance.

The overall conclusion mirrors the findings of the previous experiment: the ranking-based metrics consistently indicate that the semi-supervised Autoencoder (represented by the red dashed line in Figure 4.21 and 4.22) outperforms the fully supervised pipeline. This further confirms the superiority of integrating a supervised head within the Autoencoder architecture over relying solely on a conventional supervised learning pipeline.

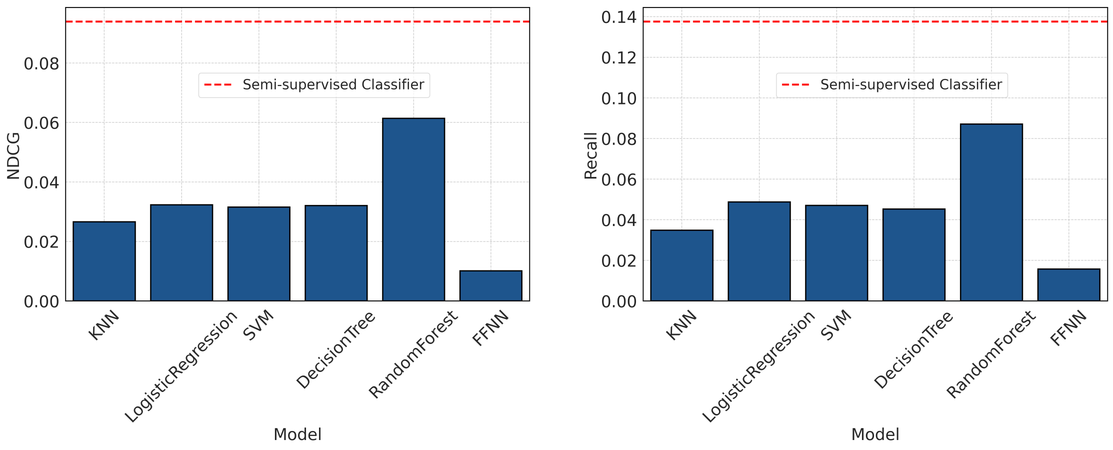


Figure 4.21: NDCG and Recall at K = 5000 for Legal Entities.

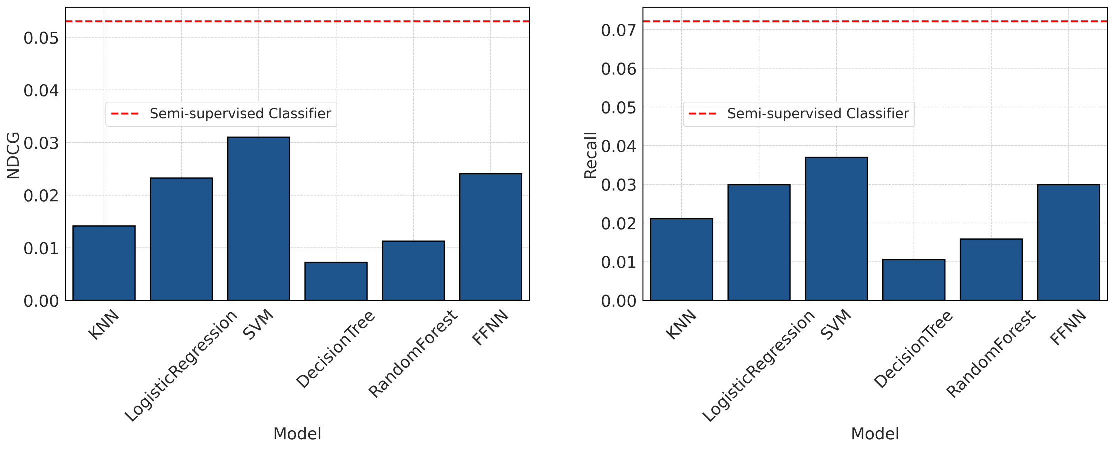


Figure 4.22: NDCG and Recall at K = 5000 for Physical Persons.

Chapter 5

Conclusion

5.1 Final results

The work presented in this thesis explores a semi-supervised approach to anomaly detection in standard banking accounts, with the goal of producing a monthly ranking of accounts based on their likelihood of engaging in financial misconduct. The core model is an Autoencoder, selected both for its strong ability to capture unusual behavioural patterns through reconstruction error and for its flexibility in supporting unsupervised and supervised extensions.

The project unfolded in several stages. The bank's datasets were first examined in depth, leading to the extraction of 98 features characterizing each account's transactional behaviour. After filtering and preprocessing, these features were used to train two separate Autoencoder models: one for Physical Persons and one for Legal Entities.

The architecture combines a classical encoder–decoder with an additional feed-forward neural network attached to the latent space. Each component produces its own ranking: the unsupervised part relies on reconstruction error, while the supervised head outputs a logit score for each account. These scores, together with historical labels of reported accounts (True Positives and False Positives), allow the computation of metrics such as Precision, Recall and Normalized Discounted Cumulative Gain, which serve as performance indicators.

Throughout development, a broad set of experiments was conducted to refine and test the models. These included comparisons between incremental and single-shot training, hyperparameter optimization for the unsupervised setup, sensitivity analyses across random seeds, and an evaluation of supervised classifiers built on latent representations. The Autoencoders were also benchmarked against standard unsupervised baselines and a fully supervised pipeline. Overall, the experiments show that separating Physical Persons and Legal Entities improves results, that incremental and joint training strategies strengthen detection quality, and that the semi-supervised Autoencoder consistently outperforms both pure unsupervised models and fully supervised alternatives.

These investigations enabled a detailed assessment of model behaviour, summarized in the final plots shown in this Section, comparing metrics across the unsupervised AE, the unsupervised component of the semi-supervised AE, and the supervised component of the semi-supervised AE.

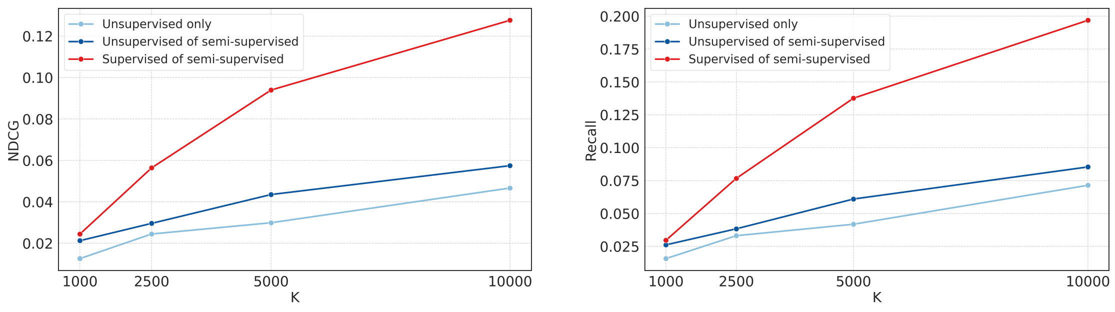


Figure 5.1: NDCG and Recall on June 2024 for Legal Entities.

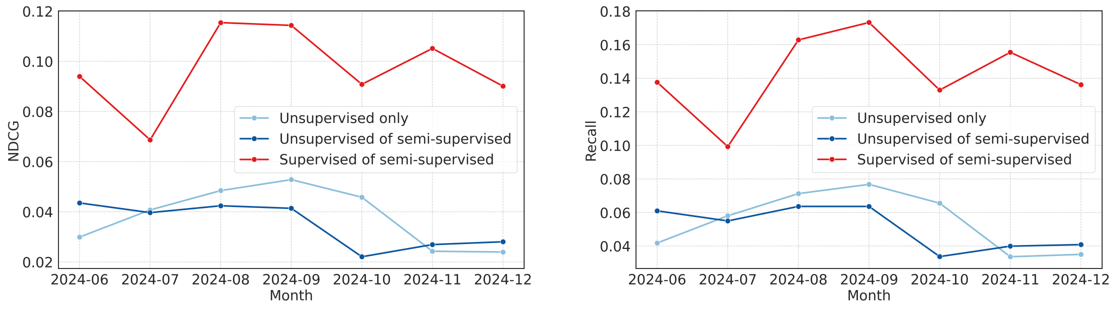


Figure 5.2: NDCG and Recall at K = 5000 for Legal Entities.

The results indicate that the semi-supervised Autoencoder achieves competitive or superior performance compared to alternative methods. Even in cases where the number of identified true positives is similar, the AE tends to produce noticeably fewer false positives: an important practical advantage, as it reduces the workload for analysts in the competence centre. In addition, the ability to inspect feature-level reconstruction errors provides valuable interpretability, helping domain experts understand why an account has been flagged.

In conclusion, this work illustrates that Autoencoder-based anomaly detection effectively captures meaningful behavioural signatures in financial data, aligns well with domain knowledge and offers tangible operational benefits. Experts

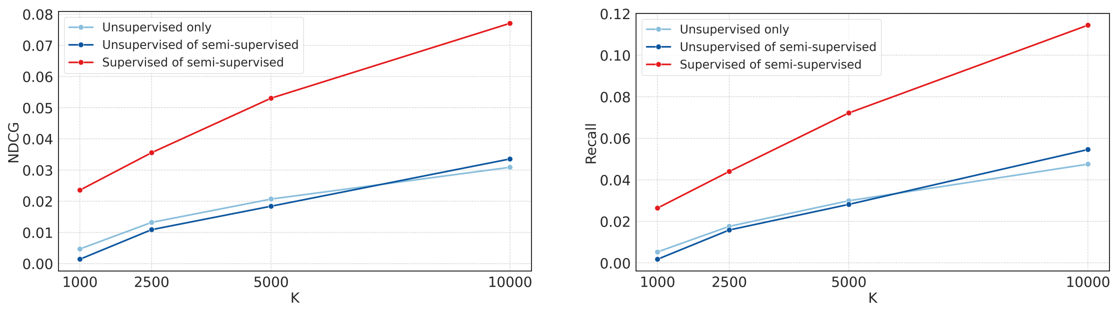


Figure 5.3: NDCG and Recall on June 2024 for Physical Persons.

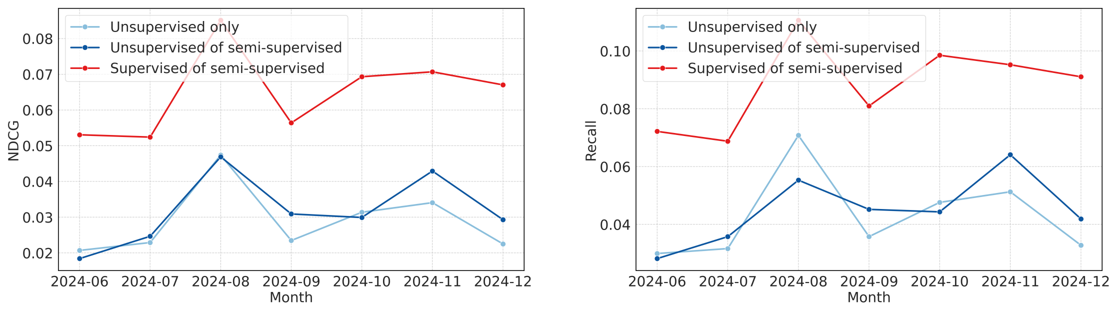


Figure 5.4: NDCG and Recall at K = 5000 for Physical Persons.

feedback has been positive, and there is a concrete possibility for this system to be deployed in production as an enhancement or even a replacement for the existing rule-based Transaction Monitoring framework. The combination of interpretability, reduced false positives and the adaptability of semi-supervised methods highlights the potential of the proposed approach, while ongoing collaboration with domain specialists and the progressive integration of new labelled data will be essential for continuous improvement.

5.2 Future works

Despite the encouraging results and the positive feedback from domain experts, several limitations remain. The most significant challenge is the extreme class imbalance: truly anomalous accounts represent only about 0.25% of all Legal Entities and roughly 0.05% of Physical Persons. This imbalance not only makes the task intrinsically difficult but also limits the maximum achievable performance of any detection system.

Future work could explore several promising directions:

- **Merging supervised and unsupervised rankings**. A unified ranking could be built by jointly fine-tuning the loss coefficients of both components, allowing the model to better balance reconstruction-based and logit-based anomaly signals.
- **Learning feature-specific weights**. Introducing trainable or carefully tuned per-feature weights may help the model focus on the most informative behavioural patterns, improving both reconstruction and supervised discrimination.
- **Applying post-processing techniques**. Additional post-processing strategies, similar to those investigated in MAD 2024[2], could further refine the ranking and correct residual weaknesses of the raw model outputs.

Pursuing these directions may enhance both performance and stability, making the system more robust in a scenario where anomalies are rare and difficult to detect.

Appendix A

Additions to experimental results

A.1 Hyper-parameter tuning of the unsupervised Autoencoders

Additional evaluations metrics

This section reports supplementary evaluation metrics considered during the analysis in Section 4.2, including precision and measures derived from false positives. While not central to the main discussion, these metrics provide additional insight into the model's behaviour.

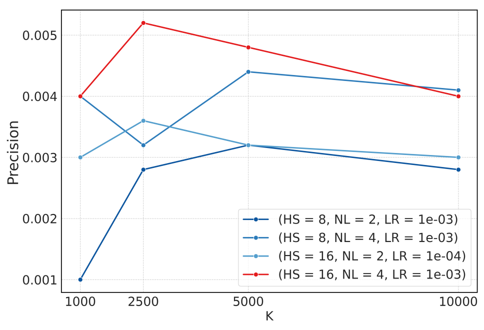


Figure A.1: Precision at K for Legal Entities.

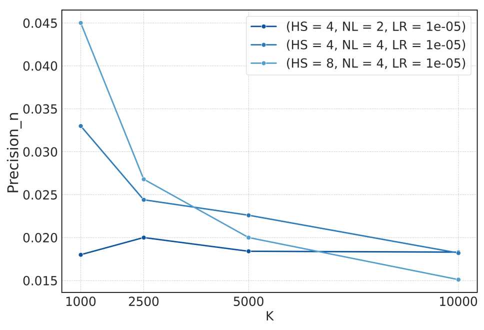


Figure A.2: Precision on FP at K for Legal Entities.

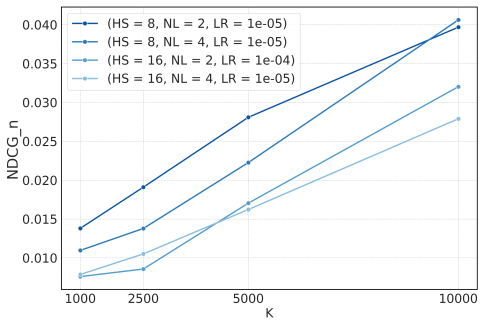


Figure A.3: NDCG on FP at K for Physical Persons.

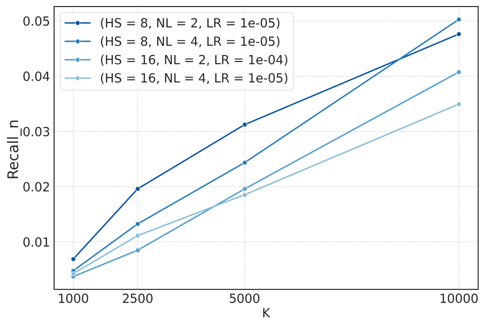


Figure A.4: Recall on FP at K for Physical Persons.

Hyperparameter stability verification

This section presents the results of the stability analysis conducted across different hyperparameter configurations, expanding the analysis of Section 4.2. By fixing one hyperparameter at a time and varying the remaining ones, the evaluation assesses whether the selected values demonstrate consistent performance, thereby supporting their reliability for deployment. The analysis was conducted on NDCG and Recall with a fixed value of $K = 5000$.

Legal Entities

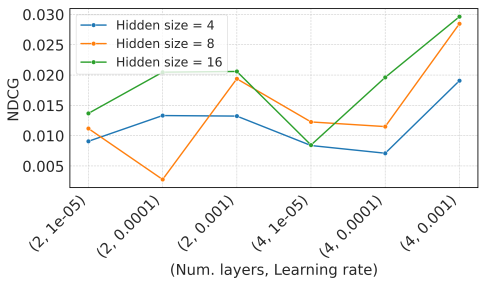


Figure A.5: NDCG at hidden size values for Legal Entities.

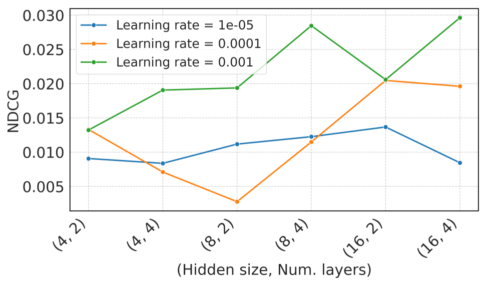


Figure A.6: NDCG at learning rate values for Legal Entities.

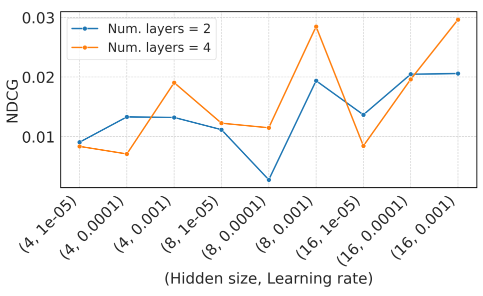


Figure A.7: NDCG at number of layers values for Legal Entities.

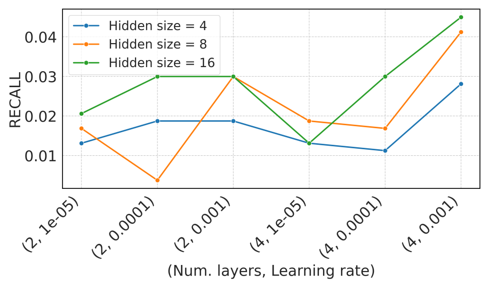


Figure A.8: Recall at hidden size values for Legal Entities.

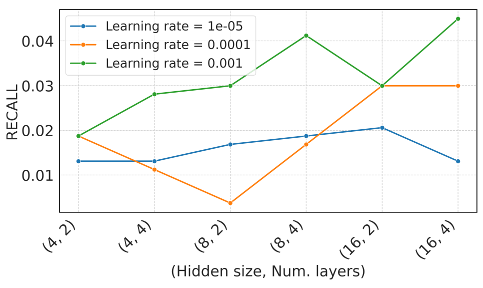


Figure A.9: Recall at learning rate values for Legal Entities.

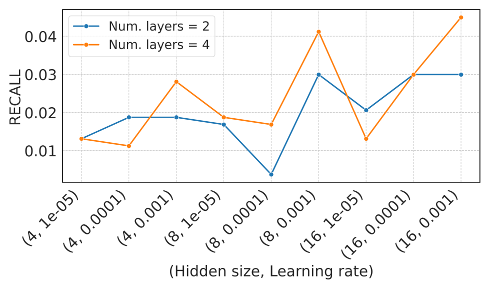


Figure A.10: Recall at number of layers values for Legal Entities.

Physical Persons

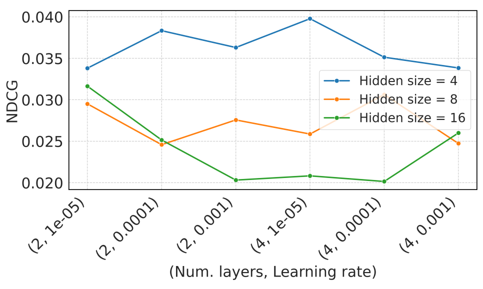


Figure A.11: NDCG at hidden size values for Physical Persons.

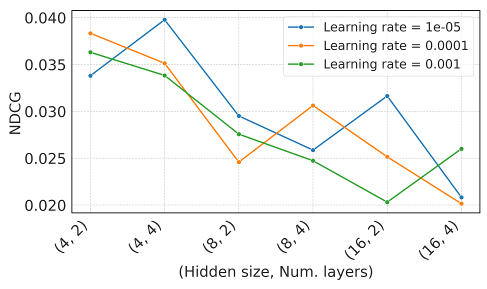


Figure A.12: NDCG at learning rate values for Physical Persons.

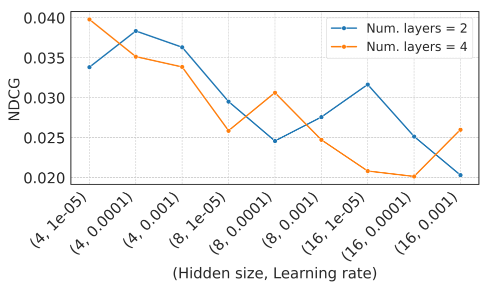


Figure A.13: NDCG at number of layers values for Physical Persons.

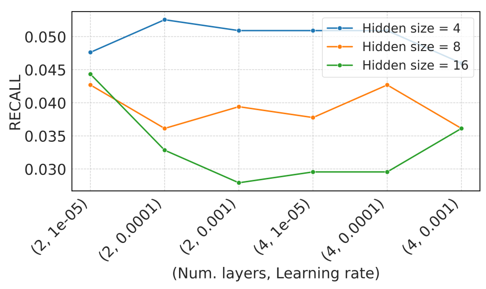


Figure A.14: Recall at hidden size values for Physical Persons.

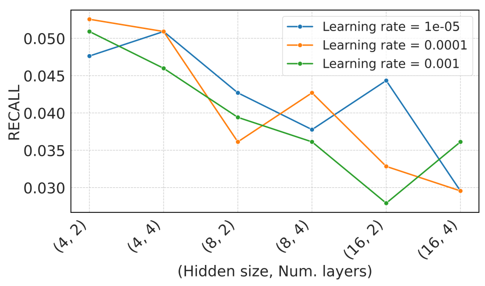


Figure A.15: Recall at learning rate values for Physical Persons.

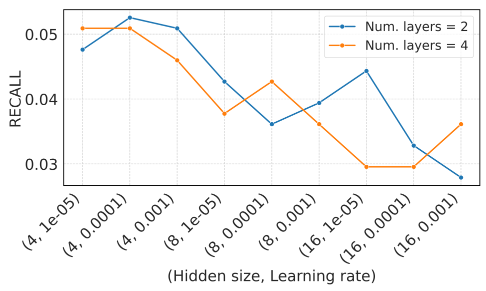


Figure A.16: Recall at number of layers values for Physical Persons.

A.2 Seed variation experiment

ECDF analysis of TP and FP frequency across seed-dependent top–5000 rankings

This subsection reports the Empirical Cumulative Distribution Functions (ECDFs) describing how frequently each ground truth account, both true positives (TP) and false positives (FP), appears in the top–5000 rankings across the seven seed values. These plots complement the aggregate statistics presented in Section 4.4 by providing a more granular view of ranking stability at the level of individual users.

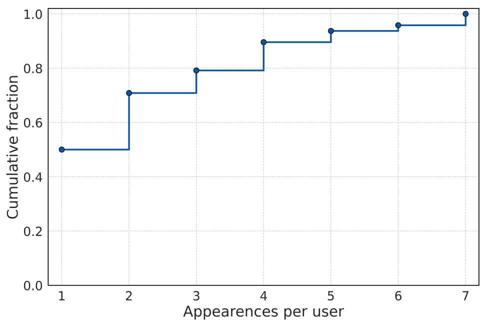


Figure A.17: ECDF of the appearance frequency of true-positive (TP) accounts in the top–5000 rankings across seven seeds (Legal Entities).

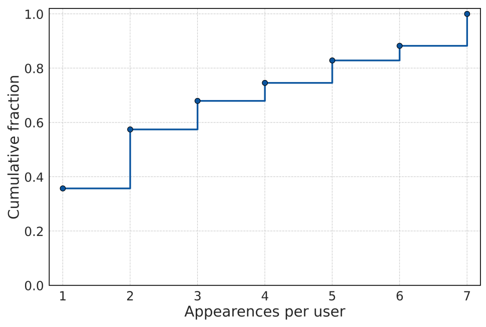


Figure A.18: ECDF of the appearance frequency of false-positive (FP) accounts in the top–5000 rankings across seven seeds (Legal Entities).

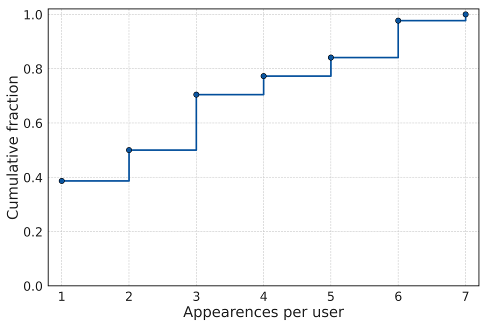


Figure A.19: ECDF of the appearance frequency of true-positive (TP) accounts in the top–5000 rankings across seven seeds (Physical Persons).

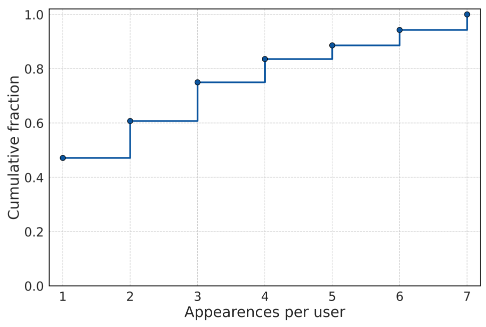


Figure A.20: ECDF of the appearance frequency of false-positive (FP) accounts in the top–5000 rankings across seven seeds (Physical Persons).

Bibliography

[1] Financial Action Task Force (FATF). *International Standards on Combating Money Laundering and the Financing of Terrorism & Proliferation - The FATF Recommendations.* 2012 (amended June 2025). URL: `https://www.fatf-gafi.org/en/publications/Fatfrecommendations/Fatf-recommendations.html` (cit. on p. 1).

[2] Giordano Paoletti, Flavio Giobergia, Danilo Giordano, Luca Cagliero, Silvia Ronchiadin, Dario Moncalvo, Marco Mellia, and Elena Baralis. «MAD: Multicriteria Anomaly Detection of Suspicious Financial Accounts from Billions of Cash Transactions». In: *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* August 3–7. Toronto, Canada: ACM, 2025, pp. 4751–4760. DOI: `10.1145/3711896.3737244` (cit. on pp. 2, 4, 75).

[3] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alexander J Smola, and Robert C Williamson. «Estimating the support of a high-dimensional distribution». In: *Neural computation* 13.7 (2001), pp. 1443–1471. DOI: `10.1162/089976601750264965` (cit. on p. 7).

[4] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. «Isolation forest». In: *2008 Eighth IEEE International Conference on Data Mining.* IEEE. 2008, pp. 413–422. DOI: `10.1109/ICDM.2008.17` (cit. on pp. 8, 9).

[5] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. «LOF: Identifying Density-Based Local Outliers». In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data.* ACM, 2000, pp. 93–104. DOI: `10.1145/342009.335388` (cit. on pp. 9, 11).

[6] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. «Reducing the dimensionality of data with neural networks». In: *Science* 313.5786 (2006), pp. 504–507. DOI: `10.1126/science.1127647` (cit. on p. 11).

[7] Thomas M. Cover and Peter E. Hart. «Nearest neighbor pattern classification». In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: `10.1109/TIT.1967.1053964` (cit. on p. 15).

[8] David R. Cox. «The Regression Analysis of Binary Sequences». In: *Journal of the Royal Statistical Society: Series B (Methodological)* 20.2 (1958), pp. 215–242. DOI: `10.1111/j.2517-6161.1958.tb00292.x` (cit. on p. 16).

[9] J. Ross Quinlan. «Induction of Decision Trees». In: *Machine Learning* 1.1 (1986), pp. 81–106. DOI: `10.1023/A:1022643204877` (cit. on p. 17).

[10] Leo Breiman. «Random Forests». In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: `10.1023/A:1010933404324` (cit. on p. 18).

[11] Corinna Cortes and Vladimir Vapnik. «Support-Vector Networks». In: *Machine Learning* 20.3 (1995), pp. 273–297. DOI: `10.1007/BF00994018` (cit. on p. 18).

[12] Financial Intelligence Unit of Italy, Bank of Italy. *Allegato 2 - Codici sintetici delle attività economiche (dal 01/01/2022 al 31/12/2025) [Annex 2 - Synthetic codes of economic activities (from 01/01/2022 to 31/12/2025)]*. `https://uif.bancaditalia.it/normativa/norm-antiricic/provv-2020-08-25/Allegato_2_codici_sintetici_attivita_economica_dal_01-01-2022_al_31-12-2025.pdf`. UIF Provision, August 25, 2020. 2020 (cit. on p. 27).

[13] Financial Action Task Force (FATF). *Professional Money Laundering*. `https://www.fatf-gafi.org/content/dam/fatf-gafi/reports/Professional-Money-Laundering.pdf`. Definition of money mule, p. 24. 2018 (cit. on p. 42).