POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

FraudBench: A Benchmarking Software for Fraud Detection Systems

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Author: **Luca Maniscalchi**

Student ID: 968628
Advisor: Prof. Michele Carminati
Co-advisors: Tommaso Paladini
Academic Year: 2022-23

Abstract

Banking frauds pose significant risks to customers and financial institutions, making the development of effective Fraud Detection Systems crucial in safeguarding assets and maintaining trust in the banking industry. The advent of Machine Learning has revolutionised this field by providing powerful tools. By leveraging a sample of labelled transactions, classification algorithms can be trained to process a massive amount of transactions in real time and effectively identify fraudulent activities. Despite the extensive research in this domain, a meaningful comparison between different studies remains challenging. This is primarily due to the absence of benchmarking software that standardises experimental procedures and ensures comparable results.

In this thesis, we introduce FraudBench, a flexible framework designed to streamline the development and evaluation of Fraud Detection Systems. This open-source framework provides a guideline for creating detection systems, supporting the development process in all its phases, from data collection to performance evaluation. Through its modular architecture, FraudBench facilitates the incorporation of different algorithms, data sources, preprocessing procedures and model selection strategies, thus enabling a more systematic comparison and validation of different fraud detection techniques. We assess our methodology by examining the effectiveness of six commonly used Machine Learning algorithms in the field of fraud detection, as well as two ensemble methods. We simulate three distinct attack scenarios where the attacker injects fraudulent transactions following different policies. Our findings reveal that one of the detection systems we tested based on the Support Vector Machine outperforms the others in two out of three scenarios over the long term. Despite its simplicity, this Fraud Detection System yields an estimated overall loss of 81% lower than the mean loss value on average.

Nonetheless, carefully selecting an optimised ensemble technique can achieve even better results. Specifically, the ensemble model that employs the Multiplicative Weight Update approach is particularly effective. Thanks to its dynamic online learning strategy that consistently picks the most reliable algorithm, this model leads to an estimated loss that is, on average, 83% lower than the mean loss value.

Keywords: fraud detection, machine learning, detection framework, benchmark, online banking

Abstract in lingua italiana

Le frodi bancarie comportano rischi significativi sia per i clienti che per le istituzioni finanziarie, rendendo lo sviluppo di efficaci sistemi di rilevamento cruciale per la salvaguardia dei beni e il mantenimento della fiducia nel settore bancario. L'avvento del Machine Learning ha rivoluzionato questo campo, fornendo strumenti molto potenti. Sfruttando un campione di transazioni etichettate, gli algoritmi di classificazione possono essere addestrati per elaborare un'enorme quantità di transazioni in tempo reale e identificare efficacemente le attività fraudolente. Nonostante l'ampia ricerca condotta in questo campo, il confronto significativo tra i diversi studi rimane difficile. Ciò è dovuto principalmente all'assenza di un software di benchmarking che standardizzi le procedure sperimentali e garantisca risultati comparabili.
In questa tesi presentiamo FraudBench, un framework flessibile progettato per semplificare lo sviluppo e la valutazione dei sistemi di rilevamento delle frodi. Questo framework open-source fornisce una linea guida per la creazione di sistemi di rilevamento, supportando il processo di sviluppo in tutte le sue fasi, dalla raccolta dei dati alla valutazione delle prestazioni. Grazie alla sua architettura modulare, FraudBench facilita l'incorporazione di diversi algoritmi, fonti di dati, procedure di pre-elaborazione e strategie di selezione dei modelli, consentendo così un confronto e una convalida di diverse tecniche di rilevamento più sistematici. Valutiamo il nostro approccio esaminando l'efficacia di sei algoritmi di Machine Learning comunemente utilizzati nel campo del rilevamento delle frodi, nonché di due metodi di ensemble. Abbiamo simulato tre distinti scenari di attacco in cui l'aggressore inietta transazioni fraudolente seguendo diverse politiche. I nostri risultati rivelano che uno tra sistemi di rilevamento che abbiamo testato basato su Support Vector Machine, nel lungo termine, supera gli altri in due scenari su tre. Nonostante la sua semplicità, questo modello produce una perdita complessiva stimata che è, in media, dell'81% inferiore al valore medio delle perdite. Tuttavia, è possibile ottenere risultati ancora migliori utilizzando una tecnica di ensemble. In particolare, il modello che impiega l'approccio di ensemble denominato Multiplicative Weight Update è particolarmente efficace. Grazie alla sua strategia di apprendimento dinamico che, nel corso del tempo, seleziona l'algoritmo più affidabile, questo modello porta ad una perdita complessiva sti-

mata che è, in media, dell'83% inferiore al valore medio delle perdite.

Parole chiave: rilevamento frodi, sistema di rilevamento, transazioni bancarie online, machine learing, benchmark

Contents

1 | Introduction

In today's rapidly advancing digital age, online banking has become an integral part of our lives. The convenience and accessibility it offers have transformed the way we manage our finances. With a few clicks, individuals can effortlessly transfer funds and conduct transactions from the comfort of their homes or on the go. However, as the online banking landscape continues to evolve, there is a parallel increase in potential threats that pose risks to users and their sensitive financial information [14]. The consequences of fraudulent activities can be severe, affecting both individuals and financial institutions. Besides the financial loss on the customers' side, online banking frauds also impact financial institutions, which bear the burden of regulatory penalties [4], damaged reputation [3], and the loss of customer confidence. Therefore, it is necessary to develop robust and effective detection systems that allow financial institutions to safeguard their customers, protect their assets, and foster a secure and trustworthy digital banking ecosystem.

Every day, millions of transactions are performed [58]. A manual inspection by human experts may require an unreasonable amount of time and resources; therefore, to address this challenge effectively, cybersecurity researchers and experts have developed automated detection systems that possess the capability to process a huge number of transactions in real time. By leveraging Machine Learning [1], these Fraud Detection Systems (FDSs) can detect and prevent fraudulent transactions more efficiently and at a greater scale compared to human review.

Previous works have explored the application of Machine Learning techniques to the fraud detection domain. To mention a few, Dhankhad et al. [27] apply ten different supervised Machine Learning models to detect credit card fraudulent transactions using a publicly available dataset. Furthermore, they combine these algorithms through ensemble learning to create a stacking classifier. The authors address the challenge of working on an unbalanced dataset by under-sampling the majority class. Their evaluation reveals that, according to the F1-score, the top-performing models are Random Forest (RF), the stacking classifier, and the Extreme Gradient Boosting (XGB) classifier. Similarly, D. Varmedja et al. [64] discuss the performance of four Machine Learning classifiers based on Logistic Regression, Naive Bayes, Random Forest and Multilayer Perceptron. Their

main goal is to show if simple Machine Learning algorithms with appropriate preprocessing can achieve the same performances of more complex models. They evaluate models in terms of precision, recall, and accuracy on a credit card fraud detection dataset available on the Kaggle platform. Differently from the previous work, the authors employ an oversampling technique to address the high unbalance of the dataset. Additionally, they reduced the feature set, selecting only the most relevant ones. Their comparison shows that the algorithm that scored the highest values in each evaluation metric used is Random Forest. Roy et al. [56], on the other hand, evaluate the efficacy of a subsection of Deep Learning topologies based on Artificial Neural Networks (ANNs), on a dataset of nearly 80 million credit card transactions. They have the opportunity to work on a real-world dataset provided by a financial institution according to a confidential disclosure agreement. In their work, in addition to data cleaning and other data preparation steps, they overcome class imbalance and scalability problems by using undersampling. The experimental evaluation shows that the size of the network is the primary driver of model performance, as larger networks tend to perform better than smaller networks. Despite the numerous challenges and ongoing developments in the fraud detection domain, the results proposed by these works are hard to compare. In particular, the experiments proposed by the authors achieve different goals, models are tested on different data sets, and the experimental source code is often not shared. A common experimental ground can be provided by benchmarking tools, which guide the construction, evaluation and comparison of experimental approaches. Related solutions have been proposed in the area of Adversarial Machine Learning (AML) [49, 53], but, to the best of our knowledge, none have been proposed in this search area.

In this thesis, we introduce FraudBench, a versatile and open-source framework tailored for designing and benchmarking FDSs. Our tool supports the development process of FDSs at all stages. With FraudBench, researchers can preprocess raw data to make them suitable for Machine Learning algorithms through tasks like data extraction, data augmentation, transaction aggregation, and attribute scaling. The framework also enables feature engineering and models' hyperparameter tuning. Within FraudBench, we already provide the implementation six algorithms that are commonly used in the fraud detection literature: Neural Network (NN) [15], Extreme Gradient Boosting (XGB) [22], Logistic Regression (LR) [15], Random Forest (RF) [17], Support Vector Machine (SVM) [15], and a variant of the Active Learning systems [28], which we refer to as Active Learning (AL). Additionally, we also include two ensemble models based on two different approaches: Majority Voting [57] (MV) and Multiplicative Weight Update (MWU) [8]. FraudBench allows researchers to evaluate the performance of the implemented detection systems against different kinds of synthetically generated fraudulent transactions. In particular, we simulate

two distinct real-world fraud schemes [19]: Information Stealing and Transaction Hijacking. Information Stealing refers to the unauthorised access and acquisition of confidential information, such as usernames and passwords for the online banking personal area, to perform transactions on behalf of the victims. On the other hand, with Transaction Hijacking, we refer to an attacker who intercepts or alters a user's transaction details in real time without the user's knowledge. These two fraud schemes define the technical means adopted by the attacker. We also categorise the attacker's behaviour employing three "fraud's profiles": Low, Medium, and High. This way, we can identify attackers implementing more or less aggressive strategies depending on the monetary value of the fraudulent transaction submitted on behalf of the user.

In our experimental setup, we use a real-world dataset provided by an Italian banking group, spanning 18 weeks from October 2014 to February 2015. We reserve the initial six weeks of transactions for the training and the subsequent twelve weeks as a test set. Starting from the training dataset, we create seven unique training sets, one for each combination of fraud schemes (Information Stealing, Transaction Hijacking) and profiles (Low, Medium, and High), plus an additional one that includes all types of fraud. These sets are then aggregated and scaled to make them compatible with Machine Learning algorithms. For each combination of base FDS (LR, NN, XGB, RF, SVM, and AL) and the aforementioned training sets, we perform feature selection and hyperparameters tuning, creating 42 different FDSs in total. Finally, upon these FDSs, we build two FDSs based on MV and MWU ensemble learning techniques. In our experimental validation, we simulate three attack scenarios where an adaptive attacker targets a financial institution employing such FDSs, all built using FraudBench, by injecting different types of fraudulent transactions (i.e., combinations of fraud schemes and profile) on a weekly basis according to different policies. We estimate economic losses using a custom loss function that places a different monetary value on false negatives and false positives. Specifically, each false positive has a cost to the institution, which has to manually analyse the transaction. At the same time, we estimate the financial impact of false negatives equivalent to the entire transaction amount. In this second case, we assume that the institution has to refund the victim. The first scenario models a situation in which the attacker follows a random policy, so fraudulent transactions are submitted on behalf of the victims by choosing schemes and profiles randomly. In the second scenario, instead, we recreate a context in which the attacker consistently targets the best-performing FDS of the financial institution and submits frauds that compromise its performance as much as possible. In the third and final scenario, we simulate an attacker that cyclically introduces each type of fraudulent transaction.

Interestingly, in the second attack scenario, where systems faced a higher stress level, the

detection model based on an SVM and trained on a dataset comprising all fraud types outperformed its counterparts. Remarkably, this SVM-based system incurred fewer monetary losses compared to systems built on more sophisticated algorithms like RF, AL, and NN, reducing losses by 28%, 60%, and 24%, respectively. Additionally, among the detection systems with the lowest losses, we find the one based on LR and trained on the set comprising all the combinations of fraud schemes and profiles. These findings suggest that developing more complex models may not yield better performance. Our results show that their efficacy varies significantly depending on the context. As expected, the fraudulent transactions causing the most significant increase in loss for most models are those categorised as High-profile, given their very high amounts. Although frauds with lower amounts are more difficult to detect because they blend better, they do not have a significant economic impact when compared to those belonging to the High profile. Our results also show that relying on MV to determine transaction labels is an ineffective solution compared to a more informed strategy like the one employed by MWU and can potentially lead to significant financial losses. In the most adverse scenario, illustrated by the second attack, the financial loss incurred through the MV ensemble approach (about 20.5 million Euros) exceeds the average losses of the individual detection systems and is around four times higher than the one achieved by the MWU ensemble (approximately 5.1 million Euros).

The main contribution of this thesis is the implementation of FraudBench, a modular and highly configurable framework specifically designed for benchmarking Machine Learning based Fraud Detection Systems.

The thesis is organised as follows: in Chapter 2, we provide the fundamental definitions and concepts essential for comprehending this research. We present the problem statement we aim to address along with the challenges encountered in our pursuit; in Chapter 3, we focus on the formalisation of our threat model, we provide a detailed explanation of the attackers' objectives and their capabilities; in Chapter 4 we analyse the relevant characteristics of the dataset used in this work. Furthermore, we present how the contained information is enhanced by working on the features; in Chapter 5, we delve into a theoretical exploration of the adopted approach, showcasing the chosen models and the methodologies applied. This includes a range of techniques, from dataset augmentation to hyperparameter tuning; in Chapter 6 we provide an in-depth exploration of the framework implementation details; in Chapter 7 we describe the goals and details of the experiments that we have performed to validate our analysis, we interpret the final results according to specific metrics and considerations; in Chapter 8 we discuss the limitations of our work, outlining potential areas of focus for future research endeavors; in Chapter 9 we summarize the findings of our work and present the conclusions; Appendix A is dedicated

to the experimental evaluation of the approach against a neural network equipped with an adversarial defense mechanism; in Appendix B we provide tables containing detailed results of some discussed procedures to support the conclusions and findings presented in the main body of the document; in Appendix C we provide the formal definitions of the standard metrics we use in this work.

2 | Background and Motivation

2.1. Background

Online banking has revolutionised the way we manage our finances. With digital technology's advent, carrying out transactions, accessing account information, and performing financial activities have attained unprecedented levels of convenience and accessibility. However, this convenience also brings forth new challenges and risks that financial institutions and users must navigate [14]. Cybercriminals continuously devise sophisticated techniques to gain unauthorised access to sensitive data, perpetrating fraud and financial crimes. Financial institutions have responded to these challenges by implementing robust security measures, among which Fraud Detection Systems (FDSs). Advanced authentication protocols, encryption mechanisms, and real-time monitoring systems are deployed to safeguard user accounts and transactions [6]. Despite these proactive measures, the impact of fraud on the global economy remains significant. According to SEON, in 2022, the total value of bank fraud in the United States reached approximately $1.67 billion [31]. The consequences extend beyond financial losses, encompassing reputational damage [30], legal ramifications [4], and erosion of trust in the banking sector. As a result, continuous research and development efforts are mandatory to enhance fraud detection and prevention strategies. By gaining insights into the dynamics of online banking fraud, financial institutions can fortify their defences, develop effective risk mitigation strategies, and ensure the security and trustworthiness of online banking services for users worldwide.

2.1.1. Fraud Detection Challenges

Fraud detection in the context of online banking poses several challenges that we need to overcome. These challenges include:

1. **Large volume of transactions to process**. Online banking transactions occur at a massive scale, with millions of transactions happening daily [2]. Analysing and processing this enormous volume of data in real time becomes a daunting task for banks. Efficient algorithms and scalable infrastructure are crucial to handle this

high transaction volume effectively.

2. **Asymmetrical miss-classification cost**. As in many other classification problems where the targets to detect are rare, in fraud detection, the cost of a false negative (misclassifying a fraudulent transaction as legitimate) can be significantly higher than the cost of a false positive (incorrectly flagging a legitimate transaction as fraudulent) [44]. Financial institutions need to strike a balance between minimising false negatives to prevent financial losses and managing false positives to avoid unnecessary disruptions to legitimate transactions.

3. **Imbalanced dataset**. Fraudulent transactions are rare compared to legitimate transactions, leading to imbalanced datasets where the majority class dominates. Typically, fraud cases represent a very small proportion of the entire dataset, ranging from as low as 0.1% to 1% [19]. This imbalance can hinder the learning process of Machine Learning models, as they may become biased towards the majority class and struggle to detect instances of fraud accurately [39]. Techniques such as oversampling, undersampling, or using specialised algorithms designed for imbalanced data are necessary to address this challenge. In short, those techniques help mitigate the impact of imbalanced data by either artificially balancing the class distribution or modifying the learning process to account for the imbalance.

4. **Class overlapping**. Cybercriminals constantly adapt their techniques to evade detection. Frauds are harder to spot as attackers are becoming more proficient in blending their fraudulent activities within the vast pool of legitimate user transactions. The study conducted by Prati et al. [54] demonstrated that the impact of class overlapping on model performance is even more significant compared to class imbalance. This trend highlights the need for advanced detection techniques to uncover subtle anomalies effectively.

5. **Lack of data**. Obtaining labelled fraud data for training and evaluation purposes can be challenging. Banks are unwilling to share their dataset for privacy and confidentiality reasons, limiting access to real-world fraud examples [9]. Addressing this challenge requires creative approaches, such as synthetic data generation, collaboration with industry partners, or leveraging external data sources to supplement the limited fraud data.

Overcoming these challenges necessitates continuous research and innovation in fraud detection methodologies.

2.2. Machine Learning for Fraud Detection

At the state of the art, Fraud Detection Systems use Machine Learning to tackle the challenges mentioned earlier effectively. Machine Learning is a branch of artificial intelligence that focuses on creating algorithms and models that can learn from data and improve over time without being explicitly programmed. The primary advantage of incorporating these algorithms into FDSs is the automation of the detection process. Depending on the specific problem settings, Machine Learning algorithms are usually classified into three macro categories: supervised, unsupervised, and reinforcement learning. Supervised learning involves training the algorithm using labelled data, where the input and output pairs are provided. On the other hand, unsupervised learning deals with unlabeled data and aims to find patterns or structures within the data. Reinforcement learning focuses on training algorithms to make decisions based on feedback from their environment, seeking to maximise a reward signal over time. In addition to these three broad categories, we can also find active learning, a combination of supervised and unsupervised learning where algorithms are trained using labelled and unlabelled data. Lastly, it is important to note that models belonging to different categories can cooperate and complement one another through Ensemble models. The latter involve combining predictions or knowledge from multiple models to improve overall performance, robustness, and generalisation. In this thesis, we focus on supervised learning, active learning, and ensemble learning because models belonging to these categories are the most frequently employed in fraud detection research.

2.2.1. Supervised Systems

Supervised learning involves training models on a labelled dataset, where each training example has an associated label representing the desired output. The model learns to make predictions or classifications on new data based on what it has learned from training examples. Popular supervised learning algorithms include Logistic Regression, Support Vector Machines, and Neural Networks:

- **Logistic Regression** (LR) [24]: is a simple Machine Learning technique for binary classification. By applying a logistic function (see Equation (2.1)) to the linear combination of input features, it estimates the probability of an observed sample belonging to a particular class. Logistic Regression, in essence, provides probabilities ranging from 0 to 1 rather than assigning definite class labels. A threshold or cut-off value (typically set to 0.5) is employed to obtain the actual class labels. Logistic regression is easily interpretable and computationally efficient, making it suitable

for small and large datasets. However, its simplicity hinders its ability to learn complex relationships within data.

$$p(c|x) = \frac{1}{1 + \exp^{-(wx)}} \tag{2.1}$$

where w is the weights vector and x the observed data sample

- **Support Vector Machines** (SVM) [23]: SVMs are non-probabilistic models used for binary classification. They visualise training samples as points in space and aim to find a hyperplane that separates the space into two regions (representing the two classes) with the largest margin (i.e. the distance between the hyperplane and the nearest points from either group is maximised).
  Mathematically, any hyperplane can be represented as $wx - b = 0$, where w is a weights vector and b is an arbitrary constant. The points lying on the margin boundaries (illustrated as dashed lines in Figure 2.1) take the name of support vectors. They are the most informative and influential in defining the decision boundary position.

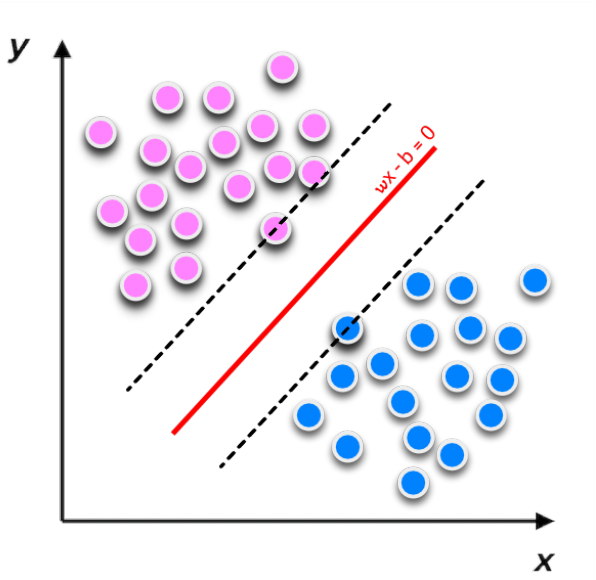
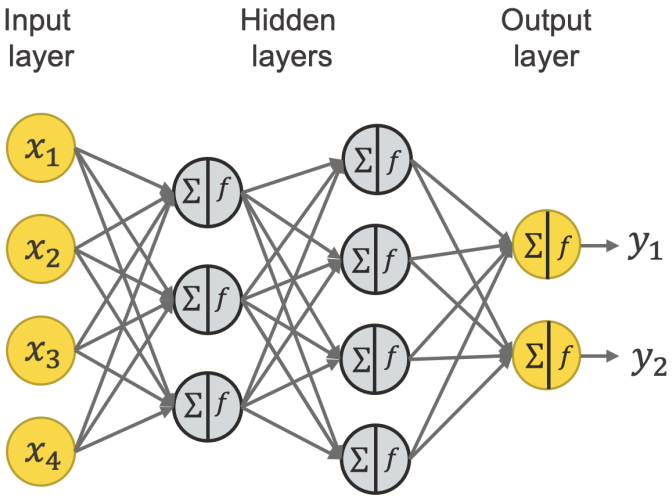


Figure 2.1: Example of decision boundary chosen by a trained SVM model

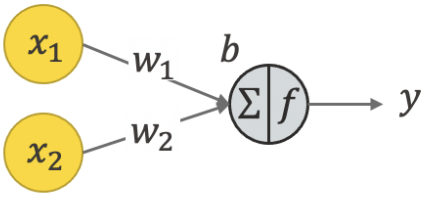
The assumption that two classes are linearly separable in the input space does not always hold. However, by mapping input data to high-dimensional feature spaces, SVMs can also handle complex decision boundaries and nonlinear relationships.
Nevertheless, SVM has a few disadvantages. Firstly, SVM models lack explainability, making it challenging to interpret their decision-making process. Secondly,

finding the best parameters for SVM, in terms of time and memory, can be computationally expensive.

- **Neural Networks** (NN) [16]: Neural networks, often referred to as artificial neural networks, are versatile and flexible models that consist of interconnected nodes, or "neurons", organised in layers. Figure 2.2 illustrates a basic Neural Network configuration together with the structure of a single neuron. Layers can have varying numbers of neurons, and each neuron can establish connections, either complete or partial, with neurons from the previous and subsequent layers in the network's topology.



(a) Example of Neural Network topology (source: [45])



(b) A Neural Network neuron (source: [45])

Figure 2.2

Neural networks learn by adjusting the weights w and biases b of connections between neurons through a process called backpropagation. At each iteration of the learning process, the neural network compares its prediction with the expected output and back-propagates the error to update its weights using specific learning rules

and optimisation techniques. Through successive adjustments, the network gradually generates predictions that closely align with the expected outputs.
NN models can learn complex patterns and relationships, making them well-suited for various tasks, including classification, regression, and even more advanced tasks like image and text processing. These advantages come with the drawback of a complex and difficult-to-interpret model that requires a substantial amount of resources and necessitates a considerable time investment during the training phase.

2.2.2. Active Learning

Active Learning (AL) [28, 41, 65] is an algorithm organised in an "analyst-in-the-loop" framework. It is a special case of Machine Learning in which the learning algorithm can interactively query an analyst to label new data points with the desired outputs. The system exploits the unsupervised model to perform anomaly detection, assigning an anomaly score to each unlabeled data sample and proactively selects the subset of most significant instances to be presented to a human analyst for manual investigation. As the analyst reviews the presented instances, their feedback labels are collected and stored, creating a growing labelled dataset that is subsequently used to train the supervised model. Upon reaching full operational capacity, characterised by a sufficient volume of data to train the supervised model, the final anomaly score for each new data sample is the combination of the scores from both the unsupervised and supervised modules.
The active learning model's power lies in its ability to operate in scenarios where historical labeled data is unavailable, which is a challenging and common scenario. Additionally, it can leverage human analysis's valuable insights, continuously refining and adapting to new patterns over time.

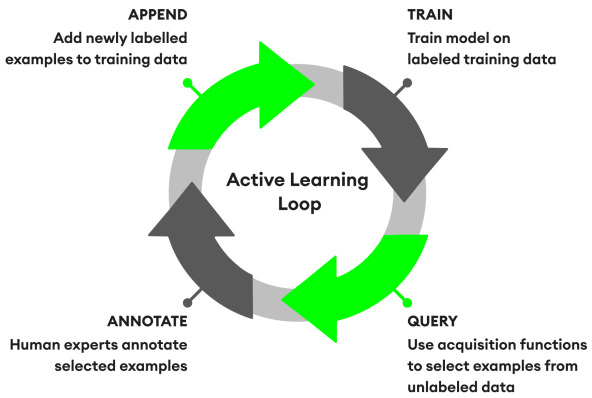


Figure 2.3: Active Learning Training Loop (source: [63])

2.2.3. Ensemble Models

Ensemble learning is an approach that aims to improve prediction accuracy by combining multiple Machine Learning models in the prediction process, leveraging their individual strengths. It exploits techniques such as majority voting, bagging, boosting, and stacking to create ensembles of diverse models that collectively make more reliable predictions. Some techniques may also include a dynamic approach where the model learns and adapts in real time as new data becomes available (online learning).

- **Majority Voting** (MV) [57]: it involves aggregating the predictions of individual models, every learner makes individual predictions, and the candidate for final prediction is the one which gets more than half of the total votes. Given a set of binary values $x_1, x_2, ..., x_n$, where each x_i represents a binary vote (0 or 1), the majority voting function returns the majority value in the set. We can mathematically represent it as:

$$\text{MajorityVote}(x_1, x_2, \ldots, x_n) = \begin{cases} 0, & \text{if} \sum_{i=1}^{n} x_i < \frac{n}{2}, \\ 1, & \text{if} \sum_{i=1}^{n} x_i \geq \frac{n}{2}, \end{cases}$$

  In statistical terms, the predicted target label of the ensemble is the mode of the distribution of individually predicted labels. This approach can be effective in scenarios where each model has equal importance and contributes equally to the final decision.

- **Multiplicative Weight Update** (MWU) [8]: is an online learning ensemble model technique that dynamically adjusts the weights assigned to each underlying model based on their performance. At the beginning of the algorithm, all models are assigned equal weights, which can be represented as the vector w given by:

$$w = \left[\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right]$$

  where n is the total number of models.
  For each incoming data sample, the MWU outcome is determined by taking the weighted sum of predictions from all models, denoted as $[\text{prediction}_1, \ldots, \text{prediction}_n]$:

$$\text{MWU\_prediction} = \sum_{i=1}^{n} w_i \cdot \text{predictions}_i$$

  The key aspect of this ensemble method lies in the weights update step. After each

prediction, the MWU model adjusts the weights vector w to give more importance to models that yield a lower loss. This is achieved through the following formula:

$$w_i^{(t+1)} = w_i^{(t)} \cdot \exp(-\eta \cdot m_i^{(t)})$$

where η represents the learning rate and $m_i^{(t)}$ represents a cost vector computed using the gradient of the loss.
Finally, to ensure that the weights sum up to 1 after the update, they are normalised as follows:

$$w = \frac{w}{\sum_i w_i}$$

This adaptive approach enables the ensemble to consistently enhance its performance by placing greater emphasis on the model that has successfully minimised losses up to that point.

- **Random Forest** (RF) [17]: is an ensemble learning algorithm that combines multiple Decision Trees [55]. The latter construct models in the form of trees using labeled data. Initially, all training samples reside in the root node of the tree. In each iteration, the node is split into two child nodes based on a criterion, often chosen to separate instances from different classes effectively. As the algorithm progresses, nodes continue to be split until reaching a stopping condition.

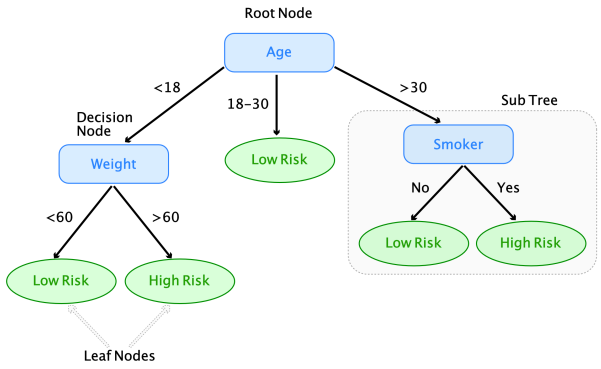


Figure 2.4: Example of a Decision Tree model that classifies the risk of having a heart attack (for illustrative purposes only; the shown data are not reliable)

When a node contains only one sample, it becomes a leaf node, marking the end of further splitting. Figure 2.4 visually represents a Decision Tree. One notable benefit of this approach is the possibility to examine and interpret each decision made by the tree, branch by branch. This provides a more comprehensive understanding of the reasoning behind the classification of data points into specific classes. However,

Decision Trees tend to overfit on the training data.
Random Forests addresses this overfitting by combining multiple trees through bootstrapping, where each tree is trained on a random subset of the data with replacement. This introduces variability and reduces the risk of overfitting to specific patterns. Additionally, feature selection is randomised, with each tree considering only a subset of features at each split, preventing over-reliance on any single feature. The final prediction in Random Forests is the result of combining predictions from all trees, either through voting (for classification) or averaging (for regression). This ensemble approach provides a regularisation effect, enhancing generalisation to unseen data, but sacrifices some interpretability compared to individual Decision Trees.

- **Extreme Gradient Boosting** (XGB) [22]: before explaining how an XGB model works in detail, it is essential to introduce the concepts of *boosting* and *gradient boosting*.
  Boosting is a Machine Learning technique that combines weak learners sequentially to create a robust predictive model. It works by training a series of models where each new model focuses on correcting the errors of the previous ones. During each iteration, misclassified data points are given more weight, making subsequent models focus on them, while correctly predicted data points receive less weight.
  Gradient boosting is an advancement of traditional boosting. It not only combines weak learners but also minimises a differentiable loss function using gradient descent as the optimisation method.
  XGB is a powerful gradient boosting algorithm that employs an ensemble of decision trees for predictive modelling. It optimises an objective function composed of two parts: a loss term ($L(\theta)$) and a regularisation term ($\Omega(\theta)$). The loss term quantifies how well the model predicts the training data, often using mean squared error as a common choice. The regularisation term controls model complexity and prevents overfitting. Figure 2.5 visually shows how the objective function is optimised.

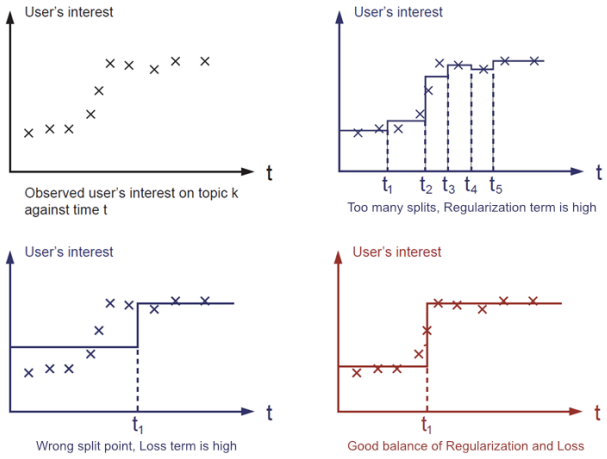


Figure 2.5: Visualization of a step function fitting process (toy example)

XGB offers several advantages, including effective regularisation techniques, flexibility in objective functions and evaluation metrics, feature importance insights, handling of missing values, and support for parallel processing. However, it can be computationally intensive, particularly when dealing with large datasets. Furthermore, it has several hyperparameters that need to be tuned to achieve optimal performance, and when compared with simpler models, it may result in less interpretability.

An essential difference between Majority Voting and Multiplicative Weight Update, on the one hand, and Random Forests and XGBoost, on the other, lies in the process of creating basic models within their respective ensembles. MV and MWU have no separate training phase; they combine predictions from pre-existing, individually trained models to make their final forecast or classification. On the other hand, Random Forests and XGBoost incorporate a distinct training phase to construct and refine their basic models. These algorithms iteratively create and optimise models as an integral part of the ensemble-building process rather than relying on pre-trained models.

In the upcoming chapters, we use the term "ensemble" exclusively for MV and MWU. Though technically ensembles, we discuss RF and XGB as individual models for simplicity.

2.3. Related Works

In the current literature, several research papers about fraud detection offer ad hoc benchmarks of Machine Learning algorithms. For instance, Cheng et al. [32] propose a fraud detection framework based on Convolutional Neural Networks (CNNs) to recognise inher-

ent behavioural patterns of fraud from annotated data. Their fraud detection framework consists of a training part and a detection part. The training part mainly includes four modules that deal with feature engineering, sampling methods, feature transformation, and a CNN-based training procedure. The detection part consists of a feature extraction module, a feature transformation module, and a classification module. Their experiments on a dataset containing over 260 million credit card transactions from a major commercial bank show that the proposed approach outperforms some state-of-the-art methods.

N. E-Arefin [29], on the other hand, explores the performance of several types of classification algorithms in the context of credit card fraud detection. These include Bayesian classifiers, function-based classifiers, lazy algorithms, meta-algorithms, rule-based classifiers, and tree-based algorithms. Bayesian classifiers are probabilistic models that employ Bayes' theorem to assess the likelihood of a sample belonging to a specific class. Function-based classifiers, e.g. Logistic Regression and Support Vector Machines, aim to establish a mathematical decision function to map input features to a target class. Lazy algorithms, e.g. k-nearest neighbours (KNN), store training instances and make predictions based on the similarity between new samples and stored instances rather than building an explicit model. Meta-algorithms enhance overall performance by aggregating the predictions of multiple base classifiers through techniques like bagging, boosting, and stacking [18]. Rule-based classifiers rely on a set of static rules, each comprising a condition on input features, to make predictions. Lastly, tree-based algorithms, e.g. decision trees, construct a tree-like model where each internal node represents a decision based on one or more input features, and each leaf node represents a class label. The experimental findings indicate that meta and tree classifiers perform better than other types of classifiers.

Similarly, G. K. Kulatilleke [38] conducts a study on credit card fraud detection and proposes a data-driven approach to dynamically select an appropriate model based on evaluation metric scores and balancing strategies. The study focuses on two PCA-encoded real-world credit card transaction datasets, and all 15 considered models employ supervised learning techniques. He provides a comprehensive explanation of various methods to address the issue of unbalanced classification, investigating two undersampling methods (random majority under-sampling with replacement and Instance Hardness Threshold) and three oversampling methods (SMOTE, random minority over-sampling with replacement, and ADASYN). His framework assumes an unknown, massively unbalanced dataset as input and creates a collection of 530 classifiers by combining different models, sampling strategies, and feature selection techniques. He then evaluates these classifiers using eight performance metrics (accuracy, precision, recall, F1, G-mean, AUC-ROC, Cohen's kappa, and Matthew correlation coefficient), and the top-performing three classifiers (in terms of

F1-score) are selected to create two ensemble models.

These papers offer valuable reference points; however, their methods cannot be directly compared due to the absence of a benchmark. This limitation makes the cross-comparison of methods challenging and hinders the cumulative progress in fraud detection research. In the literature, we can also find publicly available frameworks for evaluating Machine Learning algorithms, but these frameworks pursue different objectives with respect to what we want to achieve. Their primary aim is to test the resilience of models against adversarial attacks. Furthermore, these frameworks mainly focus on handling image data, which may limit or even make it impossible to directly employ them in fraud detection scenarios without applying substantial modifications.
To mention a few works that exploit these libraries in their process of model evaluation:

Cartella et al. [21] illustrate a novel approach to modify and adapt state-of-the-art adversarial algorithms to imbalanced tabular data in the context of fraud detection. This study proposes a model-agnostic approach applicable to any architecture. The starting point for their adversarial examples creation is the Adversarial Robustness Toolbox [49], which is a Python library for Machine Learning Security that provides tools to enable developers and researchers to evaluate Machine Learning models against adversarial attacks. While certain ART algorithms can be applied to tabular data, most of these algorithms are primarily designed for handling image data. Therefore, they had to make modifications to generate adversarial examples effectively.

Wang et al. [67] develop a method to increase Deep Neural Network classifiers' resistance to adversarial perturbations by adding to the classification process a pseudo-random matrix key generated by Logistic Chaos. To generate adversarial examples and test them against their classification procedure, they exploit CleverHans [53], which is a Python library to benchmark Machine Learning systems' vulnerability to adversarial examples. Differently from the previous one, this library is specifically focused on the image context, so it cannot be directly translated into the fraud detection context.

In summary, while existing research provides valuable insights and methodologies, the lack of standardised frameworks makes it challenging to build upon these findings in a cohesive manner.

2.4. Motivation

In this section, we introduce the problem we tackle, and we outline our objectives.

2.4.1. Problem Statement

Machine Learning algorithms and data analytics techniques represent a significant advancement in the fraud detection field. However, the evaluation and comparison of these algorithms vary from one research work to another, primarily due to the absence of benchmarking frameworks. To the best of our knowledge, there is no publicly available framework designed to benchmark Machine Learning models in the fraud detection field. In light of this limitation, we introduce FraudBench, a dedicated framework explicitly tailored for fraud detection tasks. This novel framework aims to provide researchers with a valuable tool for evaluating and analysing the performance and capabilities of diverse Machine Learning models. By offering a more thorough understanding of their effectiveness, it enables researchers to make informed decisions regarding their applicability in real-world fraud detection scenarios.

2.4.2. Goals

In this work, we aim to achieve a set of specific goals:

- Design a versatile, modular and easily extendible framework to facilitate the development and testing of Fraud Detection Systems. The framework must be flexible and highly configurable, allowing researchers to tailor the execution process to their unique needs and preferences. Furthermore, it has to be modular so that different framework components can be modified or replaced without affecting the entire system, facilitating the integration of new detection techniques or updates in the future.
- Use the framework to study the behaviour of Fraud Detection Systems when subjected to different types of attacks.

3 | Threat Model

In this chapter, we define the threat model we consider, which centers on the concept of online banking fraud. When we use the term "online banking frauds", we refer to all the malicious activities and deceptive practices that criminals use to gain unauthorised access to individuals' online banking accounts to carry out fraudulent transactions. We examine the attacker's capabilities and the main fraud schemes that exist nowadays.

3.1. Attacker Capabilities

Our work is based on the fundamental assumption that attackers possess the capability to carry out transactions on behalf of unsuspecting victims. This assumption acknowledges the existence of various techniques that enable such unauthorised actions [7]:

- Phishing and Social Engineering: This technique involves hackers disguising themselves as legitimate entities, such as banks, online services, or trusted individuals. They typically send deceptive emails or text messages that appear authentic, often mimicking the design and language of a genuine organisation. These messages may contain malicious links or attachments. If the victims click on such links or open the attachment, they may download malwares onto their device, allowing the attacker to gain unauthorised access to sensitive information or take control of the system.
- Malware: Attackers can use various methods to distribute malware. For instance, they may employ malicious advertisements (malvertising) that appear on legitimate websites. Clicking on these ads can trigger the download and installation of malware onto the victim's device. Furthermore, attackers can compromise websites, embedding malicious code into web pages. When unsuspecting users visit these compromised websites, their devices can be infected through a technique known as a drive-by-download, without any interaction or awareness from the user.
- Man-in-the-Middle (MITM) Attacks: In a man-in-the-middle attack, an attacker intercepts communication between two parties who believe they are directly communicating with each other. In the context of executing transactions on behalf of

a victim, an attacker could eavesdrop on the victim's Wi-Fi connection in a public setting. Using specialised software or devices, the attacker can monitor the victim's network traffic and capture sensitive information, such as passwords or personal details, as the victim enters them into legitimate websites or online services. This allows the attacker to gain unauthorised access to the victim's accounts and carry out transactions on their behalf.

3.2. Fraud Schemes

We examine two types of fraudulent activities, already discussed in [20].

1. **Information Stealing**. With this term, we refer to an attacker who has already obtained a user's credentials. This can occur through various methods, including phishing attacks, data breaches, or malware-infected devices. With these stolen credentials, the attacker gains unauthorised access to the user's bank account and, once there, can perform transactions and redirect funds to the preferred location. Within this context, the attacker can manipulate various aspects of the transaction. This includes the possibility to manipulate the timing of when the transaction is executed, the designated International Bank Account Number (IBAN) to which funds are transferred, and the specific amount of money involved in the transaction. In this particular scenario, the attacker executes the transaction from his or her device, resulting in the IP address and Autonomous System Country Code (ASN CC) being distinct from those associated with the user.

2. **Transaction Hijacking**. With this term, we refer to a situation where an attacker has compromised a user's device. The latter gains control over the user's browser or mobile banking application (typically through the installation of malware), allowing them to manipulate the transaction process. When the user initiates a transaction, the trojan intercepts and hijacks it, making the user believe his legitimate transaction has succeeded. The attackers can manipulate the transaction details and redirect funds to their accounts without the user's knowledge or consent. Within this context, the attacker can manipulate specific elements of the transaction, such as the destination IBAN and the transfer amount. However, he or she cannot arbitrarily determine the timing of the transaction; it is tied to the moment when the user initiates the transaction that the attacker seeks to hijack. As the transaction is executed from the user's device, the IP address and ASN CC associated with the transaction correspond to those of the user. This makes detection even more challenging.

4 | Dataset Analysis

Acquiring datasets from financial institutions presents a significant challenge due to confidentiality considerations. However, we have the privilege of having access to an anonymised real-world labelled dataset provided to our research group by a major Italian bank. In this chapter, we provide a comprehensive description and analysis of this dataset, and we show the preprocessing steps.

4.1. Dataset Extraction

The dataset we have at our disposal covers 125 days, specifically from October 22, 2014, to February 23, 2015. Transactions are divided into `bank transfers` and `reports`. The former contains records of legitimate user transactions, while the latter consists of identified fraudulent activities during that period. The dataset resulting from the combination of the two SQL tables is heavily unbalanced. Out of a total of 471,787 transactions, only 0.124% are classified as fraudulent. As discussed in Subsection 2.1.1, this is a common scenario when dealing with a banking dataset.
Each transaction of the extracted dataset is characterised by 32 features. In Table 4.1, we briefly describe the most relevant.

Feature Name	Description
TransactionID	Unique identifier of the transaction
IP	The IP address of the user's connection
SessionID	Value (assigned by the online banking platform) that identifies the session
Timestamp	Date and time at which the transaction is executed
Amount	Transaction amount in Euros
ErrorMsg	Error message (in case of a transaction that was not correctly executed)
UserID	Unique identifier assigned to the user
IBAN	Beneficiary account number
Confirm_SMS	Flag that indicates whether the transaction required a confirmation code to be completed
IBAN_CC	Country code of the beneficiary IBAN
CC_ASN	Country code of the Autonomous System Number associated with the connection of the device from which the transaction is performed
Fraud	Flag that indicates whether the transaction is fraudulent

Table 4.1: Most relevant dataset features

Data contained in *IP*, *IDSessione*, *UserID*, and *IBAN* are hashed to guarantee users privacy without losing information.

4.2. Dataset Description

The dataset includes 58,481 users and 471,199 legitimate transactions, so on average, each user performs eight transactions. However, as Figure 4.1 shows, when we divide the users into equal-sized bins based on the number of samples, it becomes evident that the distribution is heavily skewed. Nearly half of the users in the dataset have conducted fewer than four transactions.

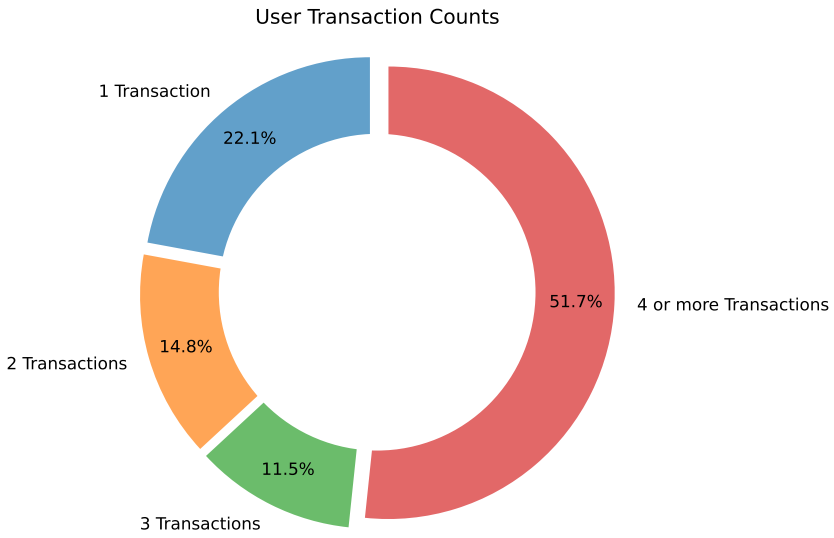


Figure 4.1: Transaction count per user

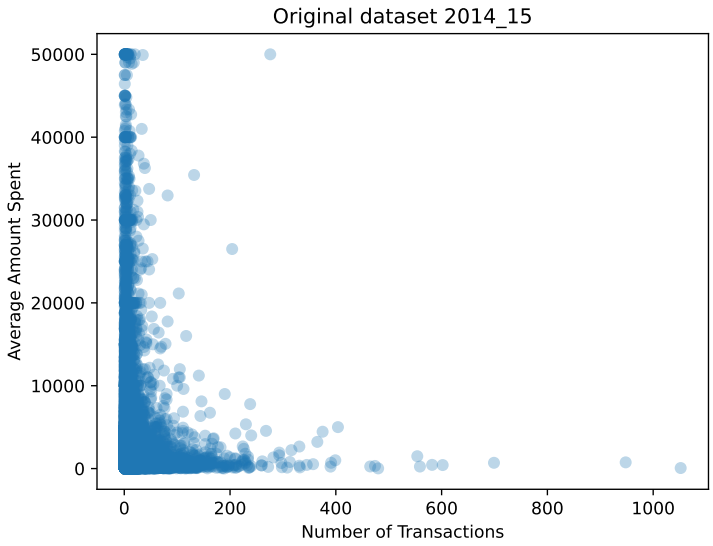


Figure 4.2: Users habits

Upon analysing the spending patterns of users, the dataset lacks clear and distinct clusters. Instead, the majority of users display a uniform distribution in the lower left quadrant of the spending patterns plot (see Figure 4.2). On average, these users engage in 0 to 100 transactions with an amount spanning from 10 € to 10,000 €.

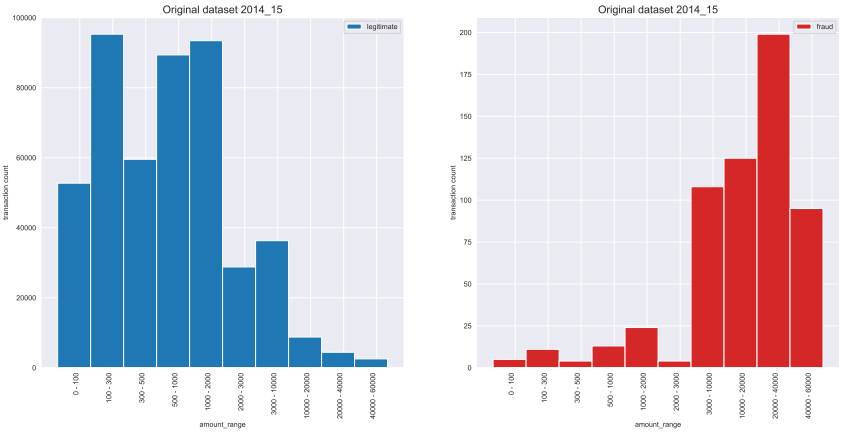


Figure 4.3: Amount distribution using custom bins

We observe an interesting trend by analysing transaction amounts grouped in custom bins. As Figure 4.3 shows, legitimate transactions predominantly fall below the 2000 €, while frauds tend to involve larger amounts, exceeding 20000 €. The average amount of fraudulent transactions is approximately 12 times higher than that of legitimate transactions, highlighting the financial impact of fraud.

Looking at Figure 4.4, we can examine the monthly distribution of transactions and notice that they appear almost uniformly distributed over time. Each month registers around 10^5 transactions, indicating a consistent flow of activity. However, it is worth clarifying that October 2014 stands as an exception, with transactions recorded only after the 22^{nd} of that month. Furthermore, there is no substantial preference for specific months when considering the occurrence of frauds over time. The percentage of frauds remains relatively consistent across different periods.

Turning to the hourly distribution of transactions depicted in Figure 4.5, we observe that the majority of transactions occur during typical working hours, with a notable peak of activity between 9:00 and 12:00. As expected, there is a decline in activity during the late-night hours, from 0:00 to 6:00. Interestingly, the hourly distribution of fraudulent transactions follows a similar trend to that of legitimate transactions. Attackers consciously mimic user transaction patterns to avoid suspicion, as deviating too much from the normal distribution would raise red flags. To enhance visualisation, we display the plot on a logarithmic scale.

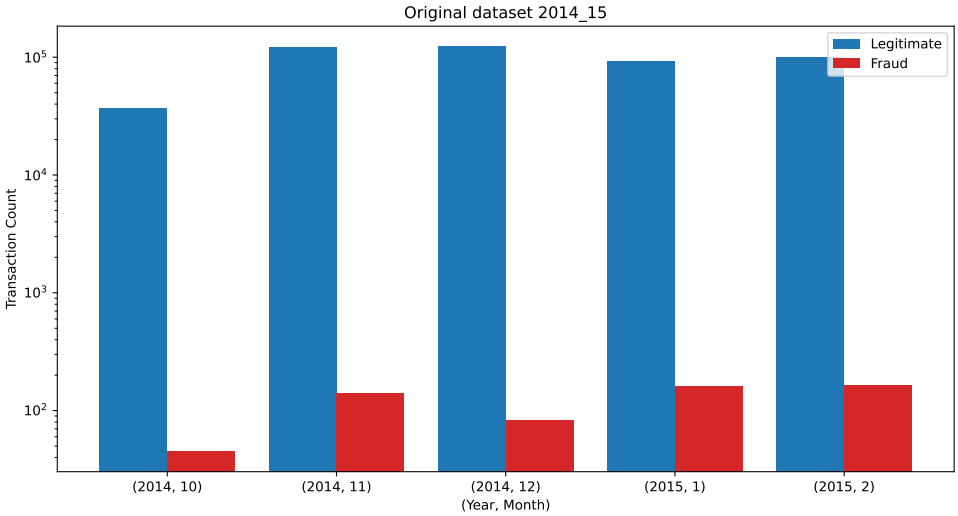


Figure 4.4: Transaction count per month

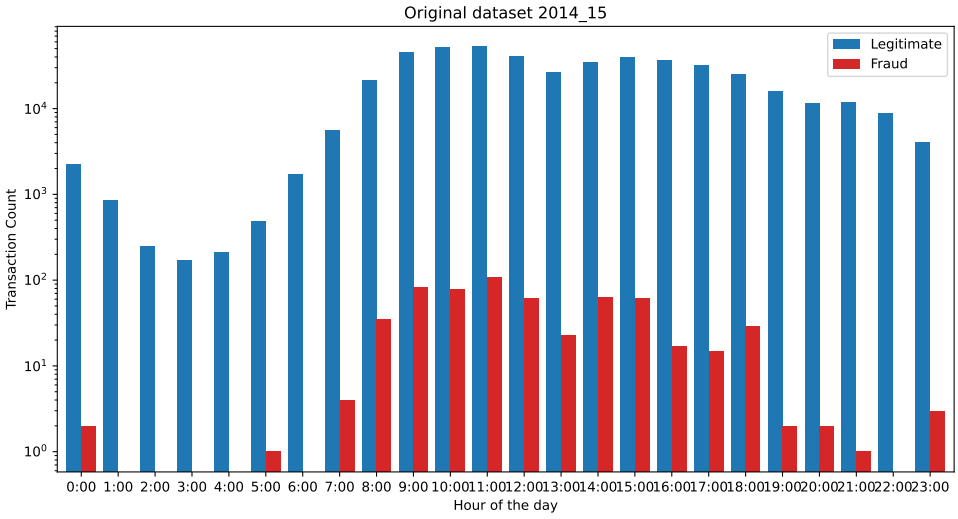


Figure 4.5: Transaction count per hour

In terms of the Country Code associated with the beneficiary IBAN and connection Autonomous System Number (ASN), it is unsurprising to find that the most commonly observed code is IT (Italy), aligning with the nationality of the bank itself. The doughnut charts in Figure 4.7 and Figure 4.6 show the distribution of IBAN country codes for legitimate transactions and fraudulent transactions, respectively. Focusing on the beneficiary IBAN Country Codes (CCs), we find that the most common CC for legitimate transactions is IT (Italy), involving 98.1% of the total number. For what concerns foreign countries, the most common CCs are DE (Germany), FR (France), RO (Romania), ES (Spain), and GB (United Kingdom). On the other hand, the percentage of frauds with an

Italian beneficiary (10.5%) is considerably lower when compared to legitimate transfers. This observation suggests that fraudulent activities often involve international beneficiaries. The foreign countries that emerge as the most common nationalities associated with fraudulent transactions are SK (Slovakia), DE (Germany), GB (United Kingdom), and NL (Netherlands).

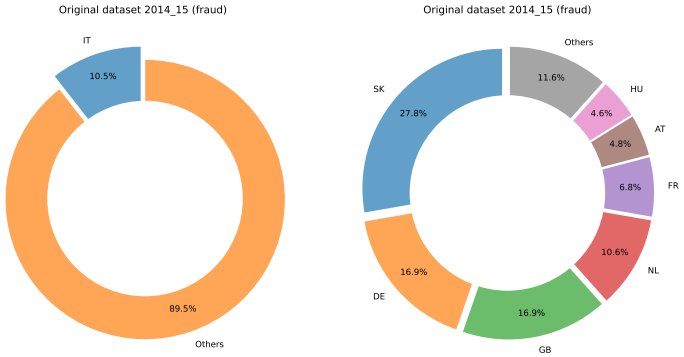


Figure 4.6: Fraudulent transaction count per IBAN Country Code

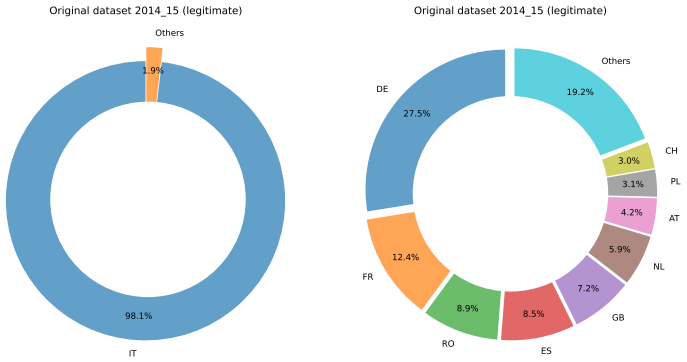


Figure 4.7: Legitimate transaction count per IBAN Country Code

In conclusion, our dataset analysis highlights some key differences between fraudulent and legitimate transactions. Fraudulent transactions tend to involve significantly higher amounts compared to legitimate ones. Furthermore, fraudulent activities often involve international beneficiaries, while legitimate transactions are mainly directed to Italian beneficiaries. These insights provide valuable information for developing effective fraud detection mechanisms and enhancing the security of financial transactions.

5 | Approach

In this chapter, we describe the approach we follow to build the framework, providing a high-level view of the proposed architecture.

5.1. Overview

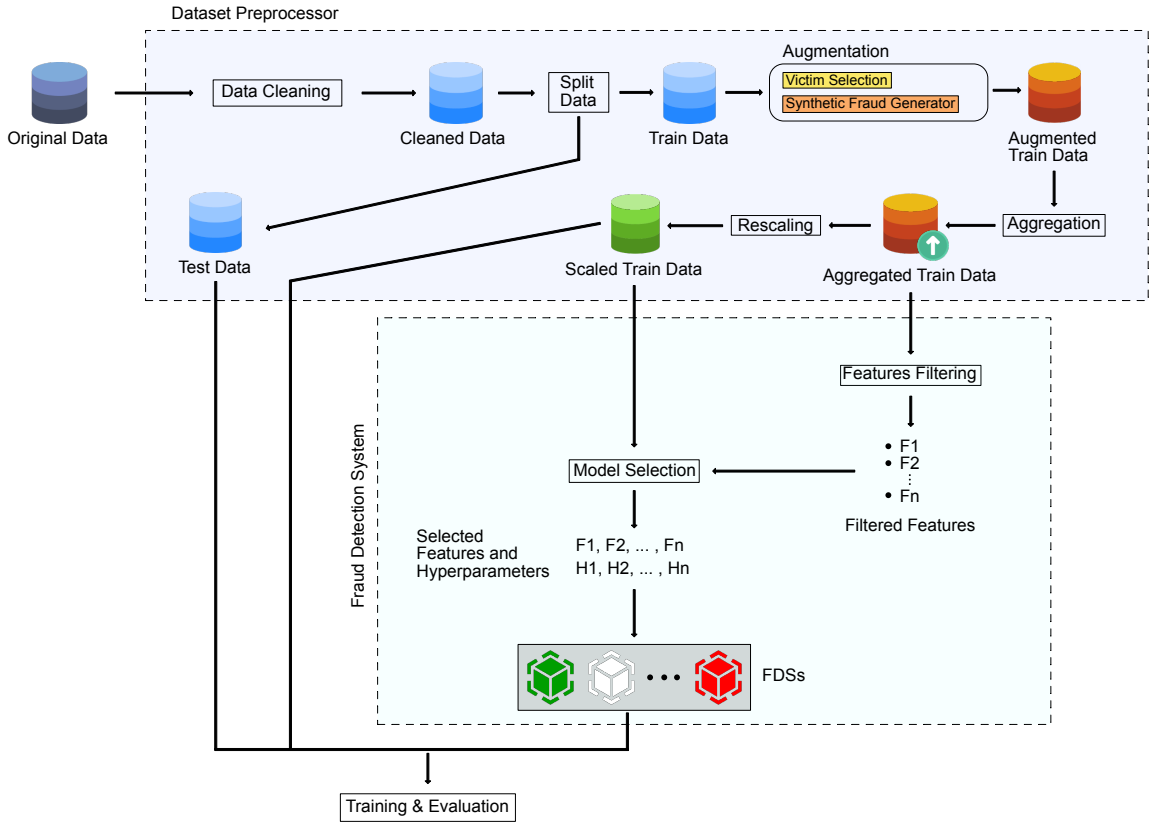


Figure 5.1: Framework logical components

Our framework is composed of two major modules. The first one is the **Dataset Preprocessor** module, which is in charge of performing all preprocessing steps. It houses the sub-components that handle tasks such as eliminating irrelevant features from the raw

data, splitting transactions into train and test sets, aggregating features, scaling them and injecting synthetic fraudulent transactions to perform dataset augmentation and rebalancing. The second main module is the **Fraud Detection System** module, which is the core of our framework. It contains the sub-components that deal with the selection of the most informative features for a given dataset, the hyperparameters optimisation for a specific detection algorithm and the instantiation of the final Fraud Detection Systems. Completing the framework is the module dedicated to the performance evaluation of the resulting detection systems. Figure 5.1 shows the complete architecture with all the sub-components detailed in the following sections.

5.2. Dataset Preprocessor Module

5.2.1. Data Cleaning

As mentioned in Section 4.1, our dataset comprises 32 columns. However, some of them do not provide any meaningful information because of the following reasons:

- **Empty fields**. Due to privacy constraints, some fields containing customers' sensitive information are empty. So we remove columns like NumConto (Account number), Intestatario (Account holder), Indirizzo (Address), etc.
- **Incomplete data**. *DataValuta* and *DataEsecuzione* are two timestamp fields representing the currency date and the execution date. About 93% of transactions contain a non-valid value for these two fields, so we remove them.
- **Non-informational fields**. We remove all columns that contain the same value for all transactions since they do not bring any useful information.

For what concerns the transactions, we filter out duplicates and transactions that were incorrectly processed because of an error by filtering on the feature ErrorMsg.

5.2.2. Dataset Augmentation

As explained in Subsection 2.1.1, one of the main challenges we face in this context is the extremely low percentage of fraudulent transactions within the dataset [19]. To address this problem, we provide an augmentation component that incorporates a synthetic fraud generator. Using this generator, we can replicate realistic fraudulent transactions and inject them inside the dataset, thereby enhancing it and achieving the target proportion of fraudulent transactions in relation to legitimate ones.

5.2.2.1. Victim Selection

A preliminary step to the fraud generation process involves victim selection. Starting from the cleaned dataset, we first discard users with insufficient data, considering only those who have performed more than three transactions as eligible for further processing. Second, to ensure the effectiveness of our Fraud Detection Systems in predicting frauds across users with varying spending patterns, we categorise the users into three distinct profiles based on the average amount and number of performed transactions:

- **High Profile**: This profile includes users who exhibit high average spending amounts or perform a significant number of transactions.
- **Medium Profile**: Users falling under this profile have more moderate spending amounts or carry out a moderate number of transactions.
- **Low Profile**: This profile encompasses users who either have very low spending amounts or execute a minimal number of transactions.

We present the numerical values we assign to the three categories in Table 5.1. By categorising users into these profiles, we account for the diverse spending behaviours that may impact the occurrence and characteristics of fraud.

Victim Profile	Amount Mean (€)	Transaction volume
High	$>$ 3,000	$>$ 35
Medium	1,500 - 2,999	15 - 34
Low	0 - 1,499	5 - 14

Table 5.1: Victims profiles

5.2.2.2. Synthetic Frauds Generator

After the selection of potential victims, we proceed with the synthesis of fraudulent transactions. According to the threat model we defined in Chapter 3, with this generator we synthesise two fraud schemes: Information Stealing and Transaction Hijacking. During the generation process of a synthetic fraud, we assign values to the fields that compose the transaction according to the characteristics of the scheme we simulate.
In our dataset, IP addresses and session IDs are hashed to ensure data privacy and confidentiality. When we craft an Information Stealing fraudulent transaction, we fill those fields by randomly generating strings and then hashing them. In particular, to model attackers who vary their connections, we assume that with low probability (5% in our case)

the IP address of the attacker changes between frauds targeting the same victim. On the other hand, we generate a different `SessionID` for each transaction unless the attacker executes another fraud towards the same victim within a short timeframe. This is because many online banking platforms implement session timeouts as a security measure, so a session expires after a short period of inactivity. Regarding the Transaction Hijacking scheme, we fill those fields using the original `IP` and `sessionID` as the attacker performs the fraudulent transaction from the victim's device and in the same session.
`Confirm_SMS` is a binary value that indicates whether the transaction required an SMS with a confirmation code to be completed. We assign this value replicating the original dataset `Confirm_SMS` frequency distribution because the presence of the two-step authentication process depends on the merchants or banks rather than on the attacker.
Similarly to IP addresses and SessionIDs, the values in the `IBAN` field are also subjected to hashing for privacy reasons. Therefore, following the same approach, we assign a randomly generated hashed string to that field.
The `IBAN_CC` represents a code with which the bank identifies the nationality of the beneficiary IBAN. As stated in the Italian Ministry of Economy's report [26], a significant portion (between 60% and 70%) of frauds in Italy involve foreign IBAN beneficiaries. Germany, France, Romania, Spain and Great Britain are the most commonly encountered European foreign country destinations. To faithfully replicate this behaviour, we assign values to the `IBAN_CC` field as follows: "IT" in 40% of the cases, one of "DE", "FR", "RO", "ES", "GB" in another 40% of the cases, and, for the remaining 20%, we assign a random European `IBAN_CC`. We apply this policy for both Information Stealing and Transaction Hijacking schemes.
The `CC_ASN` represents a code the bank uses to identify the Autonomous System Number associated with the connection of the device from which the transaction is performed. Since competent attackers often obfuscate their connections using VPNs and proxy servers, there is no correlation between the nationality of the beneficiary IBAN and the connection. So, for the Information Stealing scheme, we assign this value replicating the original dataset `CC_ASN` frequency distribution. In the case of Transaction Hijacking, this attribute takes on the same value as the one found in the victim's original transaction, as it is executed from the victim's device.
For what concerns `Amount` and `Timestamp` attributes, we assume that the attacker has full control over these fields.
Regarding the `Amount` of the fraudulent transactions, we provide the generator with a range of values within which it will pick the transaction amount. We define three profiles, namely high, medium and low. We provide the amount range for each profile in Table 5.2. Depending on what we want to achieve, we can use the synthetic fraud generator to inject

either frauds belonging to a single profile only or different profiles. The amount generation process is common to both Information Stealing and Transaction Hijacking schemes.
For what concerns the `Timestamp` instead, when we operate in the Information Stealing context, we randomly select, for each generated fraudulent transaction, a day between the first legitimate transaction of the user and the end of the dataset. Given that real frauds tend to occur more frequently in certain hours [62], we adjust the `Timestamp` accordingly. In the context of Transaction Hijacking, instead, we aim to generate fraudulent transactions that have the same `Timestamp` of the hijacked legitimate transactions or at least are very close in time (we can defer them at most 10 minutes). So, in this case, we cannot freely choose transactions `Timestamp` because it is linked to the original legitimate transactions.

Fraud Profile	Amount Range (€)
High	30,000 - 50,000
Medium	2,000 - 5,000
Low	100 - 1,000

Table 5.2: Fraud profiles

As mentioned in Section 3.2, Information Stealing and Transaction Hijacking schemes mainly differ in the way the attacker performs a fraudulent transaction. In the first case, the attacker steals the victim's sensitive data and performs frauds without modifying the user's existing or upcoming transactions. In the second case, on the other hand, the attacker manipulates some fields (e.g., the destination and the amount) of the transaction while the victim executes it. In this second scenario, we assume that there is a chance that the victim realises the original transaction was not successfull and, therefore, attempts it a second time. To model these different situations, we simulate, with a higher probability (75% in our case), the scenario in which the hijacking process occurs smoothly without encountering any errors. So, we replace the original transaction with the hijacked one. In the remaining cases (25% in our case), we assume that the victim recognises the original transaction failure and performs it again. So, we inject both the hijacked and the original version of the transaction into the dataset.

5.2.3. Transaction Aggregation

In banking datasets, each transaction is characterised by specific features that provide detailed information about it. However, to build a powerful model for fraud detection, it is necessary to train Machine Learning algorithms on a dataset that encompasses a broader range of relevant information. Original features alone are insufficient because they do not capture the complete picture of a user's spending patterns and behaviour over a certain period. Therefore, as we illustrate in Figure 6.1, we resort to transaction aggregation [51, 66] to obtain a more holistic understanding of the user. Transaction aggregation is a process in which we combine or summarise information from multiple individual transactions to create a new feature or representation that captures more relevant details from the original data. As an example, we can add a feature that indicates the average amount spent by each user within a predetermined timeframe. This additional data provides insights into consumers' spending habits, supporting the detection of anomalous behaviours that significantly diverge from their established norms. Aggregated features comprise a set of direct features derived from the input features of each transaction and features computed by aggregating transactions with past legitimate transactions belonging to the same user.

The features that can be directly derived are:

- **Amount**: the transaction amount, without any form of transformation;
- **time_x, time_y**: cyclic encoding of transaction execution time. A cyclic encoding of the time attribute is necessary when performing distance operations between different timestamps because time is inherently cyclical. The standard representation of time using linear values (e.g., seconds, minutes, hours) does not capture the cyclic nature of time, where the end of one day is followed by the start of another day. For instance, computing the distance between 10 p.m. and 11 p.m. using linear values gives 11 - 10 = 1 hour, but if we compute the distance between midnight and 11 p.m., 0 - 11 = -11 hour, which makes no sense. One approach to encode cyclical data involves transforming the data into two dimensions using a sine and cosine transformation.

$$t = ts_h * 3600 + ts_{min} * 60 + ts_{sec}$$

$$time\_x = \cos \frac{t * 2\pi}{86400}$$

$$time\_x = \sin \frac{t * 2\pi}{86400}$$

- **is_national_iban**: a binary value indicating whether the beneficiary IBAN has the same nationality as the online bank.
- **is_international**: a binary value indicating whether a transaction is international or not based on the country code of IBAN and Country code of ASN
- **confirm_SMS**: a binary value indicating whether a transaction required a confirmation code to be completed

Before delving into the explanation of how we aggregate transactions, we need to establish three sets: group, function, and time.

- *group*: this set consists of the original attribute names, which are **IBAN**, **IP**, **IBAN_CC**, **CC_ASN** and **SessionID**.
- *function*: this set comprises four mathematical operations:
  - count: returns the total number of instances
  - sum: calculates the sum of the transaction amounts
  - mean: computes the average amount of the given transactions
  - std: determines the standard deviation of the transaction amounts
- *time*: this set comprises various periods denoted as 1h, 1d, 7d, 14d, 30d, and 8760h, representing one hour, one day, seven days, fourteen days, thirty days, and one year, respectively

Considering values from the "group", "function", and "time" sets, the aggregated features are:

- *group_function_time*: We obtain these features by grouping the past transactions of a user based on the specified group attribute. Then, we apply a time window of length "time", and we use the designated function on the resulting set of transactions. For instance, **ip_mean_7d** represents the mean amount of transactions executed within the last 7 days.
- **time_since_same_***group*: This feature denotes the time elapsed in hours since the user's last transaction with the same group attribute value. For example, **time_since_same_iban** indicates the time elapsed since the user's last transaction executed towards the same IBAN.

- **time_from_previous_trans_global**: This feature represents the time elapsed in hours since the user's last transaction.
- **difference_from_*group*_mean_*time***: this feature calculates the difference in amount between the current transaction and the set of transactions within a time window of length "time" towards the same "group" attribute value.
- **is_new_*group***: this is a Boolean value indicating whether the user is making a transaction towards a specific attribute group value for the first time. For instance, **is_new_iban** indicates if it is the user's first transaction towards that IBAN.

Initially, we calculate the aggregated features for legitimate transactions to assess the regular spending patterns of users. Next, we compute the aggregated features for frauds, which necessitate considering the previous transactions of the victims. So, we merge legitimate and fraudulent transactions and we apply the computation.

5.2.4. Features Scaling

Feature scaling is a preprocessing technique used in Machine Learning to standardise or normalise the numerical features of a dataset. It is essential because many Machine Learning algorithms base their calculations and derivations on the Euclidean distance, which is sensitive to the scale of input features. When features have different scales, the ones that are on a much larger scale with respect to the others dominate the computation of the distance and have a greater impact on certain algorithms, leading to biased results and suboptimal model performance [5]. The primary purpose of feature scaling is to bring all the features to a common scale, usually within a specific range or distribution, so they all contribute equally to the learning process. Common features scaling methods include Min-Max scaling (also known as normalisation) and Standardization. The former technique scales the features to a specific range, typically between 0 and 1. It transforms the data by subtracting the minimum value and dividing it by the range (maximum value minus minimum value). The latter, instead, rescales the features to have a mean of 0 and a standard deviation of 1. It involves subtracting the mean and dividing by the standard deviation of each feature.
Due to the specific characteristics of banking datasets, we choose standardisation over Min-Max scaling as the feature scaling technique. Transaction amounts can exhibit a wide variance, ranging from very low amounts (e.g., 0.01 €) to very high amounts (e.g., 50000 €). Moreover, banking datasets typically comprise a larger number of transactions with smaller amounts and only a few with exceptionally high amounts. Given that, using Min-Max scaling could lead to a disproportionate compression of lower values towards

0. Consequently, the influence of transactions with lower amounts would be severely diminished, making them virtually indistinguishable.

5.3. Fraud Detection System Module

In FraudBench, we include the implementation of six commonly utilised algorithms in the literature for fraud detection: Logistic Regression [15], Support Vector Machine [15], Neural Network [15], XGBoost [22], Random Forests [17] and a variant of Active Learning [28, 41, 65]. Moreover, we incorporate two ensemble models based on two different approaches: Majority Voting [57] and Multiplicative Weight Update [8]. We refer to Section 2.2 for an in-depth description of each detection model. In this section, we provide a detailed explanation of the phases that, starting from the outputs of the Dataset Preprocessor module, lead to the development of the final Fraud Detection Systems. Firstly, we perform a preliminary feature skimming on the Aggregated Train Data to filter out less relevant features. Secondly, as shown in Figure 6.1, we provide the Scaled Train Data and the retained features as input to the component in charge of the model selection procedure. This component delivers the features and hyperparameters we use when we instantiate the final detection systems. The last step involves the training and testing of the created systems.

5.3.1. Features Filtering

The aggregation process we presented in Subsection 5.2.3 produces several potential features, but not all of them are equally valuable. Our goal is to identify the most informative and discriminative features that contribute significantly to the model's predictive power while discarding irrelevant or redundant ones. By reducing the feature set, we also enhance computational efficiency and address the challenge of the curse of dimensionality [36]. In a few words, the increase in the number of dimensions or features makes the data sparse. Consequently, the amount of data we need to accurately generalise the Machine Learning model increases exponentially.

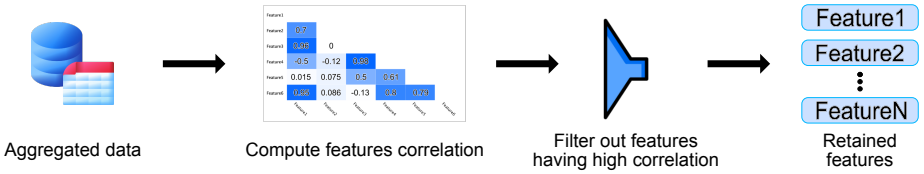


Figure 5.2: Correlation-based features filter scheme

The feature filtering component aims to perform a preliminary feature skimming. We develop a model-independent module so that we can apply it across a range of Machine Learning algorithms. We adopt a correlation-based filter method. The logic behind using correlation as a goodness measure is that relevant features are correlated to the target but uncorrelated among themselves; otherwise, they are redundant and do not provide any necessary additional information [71]. Our procedure, as illustrated in Figure 5.2, starts with the aggregated dataset. We compute the correlation matrix to understand how features are related, and based on those values, we select the most important features. To do this, we analyse one pair of features at a time, and if these two are highly correlated among themselves (more than 95% in our case), we remove the one that is less correlated with the target.

5.3.2. Model Selection

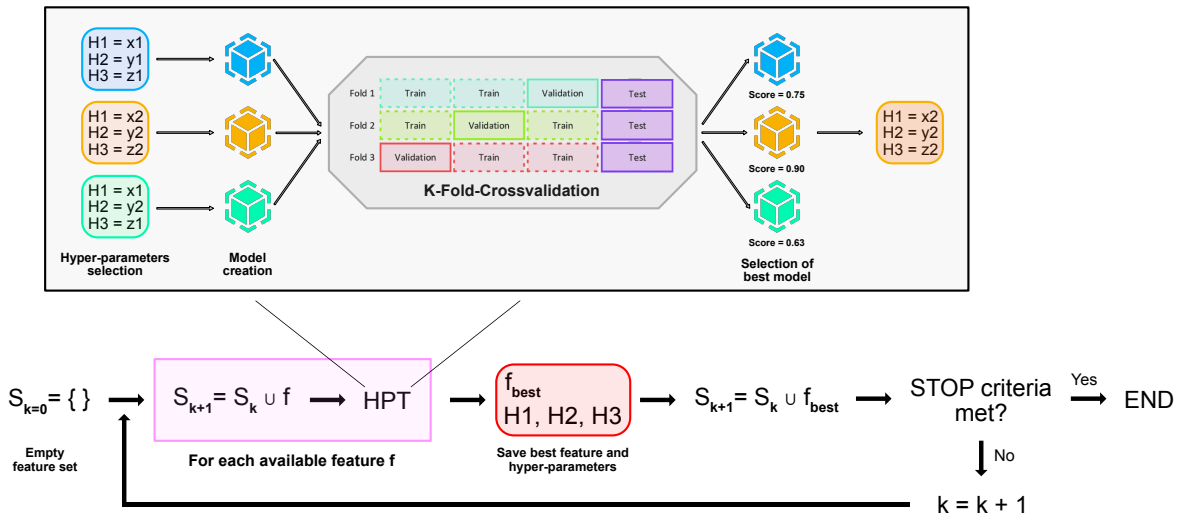


Figure 5.3: Model selection scheme

In the Model Selection component of FraudBench, we develop a procedure to perform the selection of the final feature set and hyperparameters for each detection system. Hyperparameters are external configuration variables data scientists employ to govern the training of Machine Learning models. These hyperparameters are different from internal parameters, which are automatically determined during the learning process and do not require manual specification by data scientists (e.g., the weights assigned by a Neural Network to each neuron) [70]. Examples of hyperparameters include the learning rate,

the maximum depth in a decision tree and the number of nodes and layers in a neural network. The objective of hyperparameter selection is to determine the optimal combination of hyperparameters values that either minimises a specific function (e.g., loss) or maximises it (e.g., accuracy). This involves the evaluation of various candidate hyperparameters combinations, resulting in the creation of multiple models. These models are then evaluated on validation data and compared based on their performance metrics. Ultimately, the best-performing model according to a predefined metric, along with its corresponding hyperparameters configuration, is chosen.

As Figure 5.3 illustrates, in our framework we provide a technique that combines hyperparameters and feature selection simultaneously. This method is based on forward selection and aims to find the best combination of hyperparameters and features for each detection algorithm, optimising both aspects together to enhance overall performance. Forward selection is a stepwise regression technique that starts with an empty model and progressively incorporates features. Each forward step adds the feature that contributes the most significant improvement (in terms of a predefined metric) to the model under optimisation. Based on this technique, we develop an algorithm that, starting from the set of features resulting from the filtering technique described in Subsection 5.3.1, assesses the contribution of each feature (when added to the final feature set) using a set of m models and k-Fold-Crossvalidation (with m and k configurable parameter). We determine the models' hyperparameters via a Random Search approach [12] on a predefined grid. After each iteration, we choose the best-performing feature in terms of a selectable metric (F1-score in our case) and its corresponding optimal hyperparameters, and we store the results. We then integrate the selected feature into the final feature set, and the process iterates again. We show the pseudocode of the procedure in Algorithm 5.1. We optimise the F1-score rather than the accuracy score because of the unbalanced nature of the dataset. When dealing with imbalanced datasets, the minority class often holds our primary interest. In the specific case of electronic banking frauds, datasets tend to be highly unbalanced, typically consisting of 99% legitimate transactions and only 1% fraudulent transactions. In such scenarios, even if a model fails to predict any fraudulent transaction, the overall accuracy can still be close to 99%. The reason is that this performance measure focuses on the overall correctness of predictions without considering the class distribution. Therefore, accuracy alone does not provide an accurate representation of model performance.

This model selection process has a significant computational cost, which is why we provide the possibility to restrict the number of attempts to add a new feature to the final feature set. If the inclusion of a new feature repeatedly fails to enhance the model's performance beyond a specified threshold, we terminate the process. This limitation ensures that the

process does not run indefinitely and prevents the excessive utilisation of computational resources.

Algorithm 5.1 Forward Model Selection

Input:

dataset_name: the name of the dataset to load

model_name: the name of the model to use

metric: the name of the metric to optimize

Procedure:

```
available_features = getFeaturesFromConfigFile(dataset_name)
tolerance = getParamFromConfigFile("tolerance")
max_attempts = getParamFromConfigFile("max_attempts")
random_models_num = getParamFromConfigFile("random_models_num")
folds_num = getParamFromConfigFile("folds_num")
df = loadStandardizedDataset(dataset_name)
X_train, y_train = splitTrainTest(df)
overall_best_score = 0
attempts = 0
current_feature_set = set()
while (attempts < max_attempts) ∧ (available_features \ current_feature_set ≠ ∅) do
    features_to_process = available_features \ current_feature_set
    for feature in features_to_process do
        temp_feature_set = current_feature_set ∪ feature
        models_params = generateModelsParams(model_name, random_models_num)
        cross_val_results = evaluateModelsWithCrossValidation([..., folds_num, ...])
        features_scores[feature] = getFeatureBestScore(cross_val_results, metric)
    end for
    best_feature, best_feature_score = getBestFeature(features_scores)
    current_feature_set = current_feature_set ∪ best_feature
    current_score = best_feature_score
    saveResultsToFile()
    if (current_score - overall_best_score) < tolerance then
        attempts = attempts + 1
    else
        attempts = 0
    end if
    if attempts = 0 then
        overall_best_score = current_score
    end if
end while
```

5.4. Training and Evaluation Module

Once we select the final set of features and hyperparameters and we instantiate the final Fraud Detection Systems, we proceed with the ultimate training and evaluation. In the context of banking transactions, statistical properties and underlying concepts of the target variable or feature distribution can change over time [25, 43]. This is a highly significant phenomenon known as concept drift. It refers to the dynamic nature of fraudulent and legitimate transaction patterns that evolve and shift over time. As historical data becomes outdated, models trained on such data may gradually lose their effectiveness in detecting fraudulent transactions. Therefore, we must take proactive measures to adapt the system to these changing dynamics. To mitigate the impact of concept drift, in this work, during the detection systems' training process, we adopt a weighted approach that assigns varying importance to samples based on their temporal occurrence. Specifically, we utilise a decreasing exponential function that accords greater weightage to more recent transactions:

$$weights = e^{-\frac{t}{k}}$$

The parameter t denotes the time interval, measured in hours, between the training timestamp and the timestamp of the transactions under consideration. Meanwhile, the parameter k governs the rate at which the weights diminish as time progresses. We determine the value of k based on the duration the dataset we have at our disposal. Specifically, k represents the number of hours encompassed within this dataset. By incorporating this weighting mechanism, we prioritise the influence of recent examples, recognising their higher relevance in capturing the latest trends and patterns.
To assess the performance of the Fraud Detection Systems, we evaluate them by processing one week of transactions at a time, starting from the week following the training time. These transactions comprise both legitimate and fraudulent instances, where we synthetically generate the fraudulent ones using the fraud generator we detailed in Section 5.2.2.2. We evaluate detection systems both in terms of standard metrics (we refer to Appendix C for their mathematical definitions) and a custom loss function described in Section 7.1.

6 | Implementation Details

In this chapter, we describe the implementation details of FraudBench. We explore the directory structure and provide the list of software tools, libraries, programming languages and hardware configurations employed to carry out our experiments.

6.1. Framework Architecture

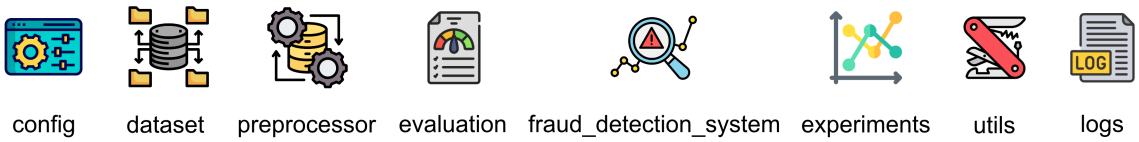


Figure 6.1: Framework Overview

Here, we present an overview of the high-level organisation of our framework. We organise the latter into eight modules, visually represented in Figure 6.1. This modularity enables the framework to be easily expandable and maintainable. In the following sections, we provide an in-depth description of the content of each module, giving a map to navigate through the codebase efficiently.

6.1.1. Configuration Module

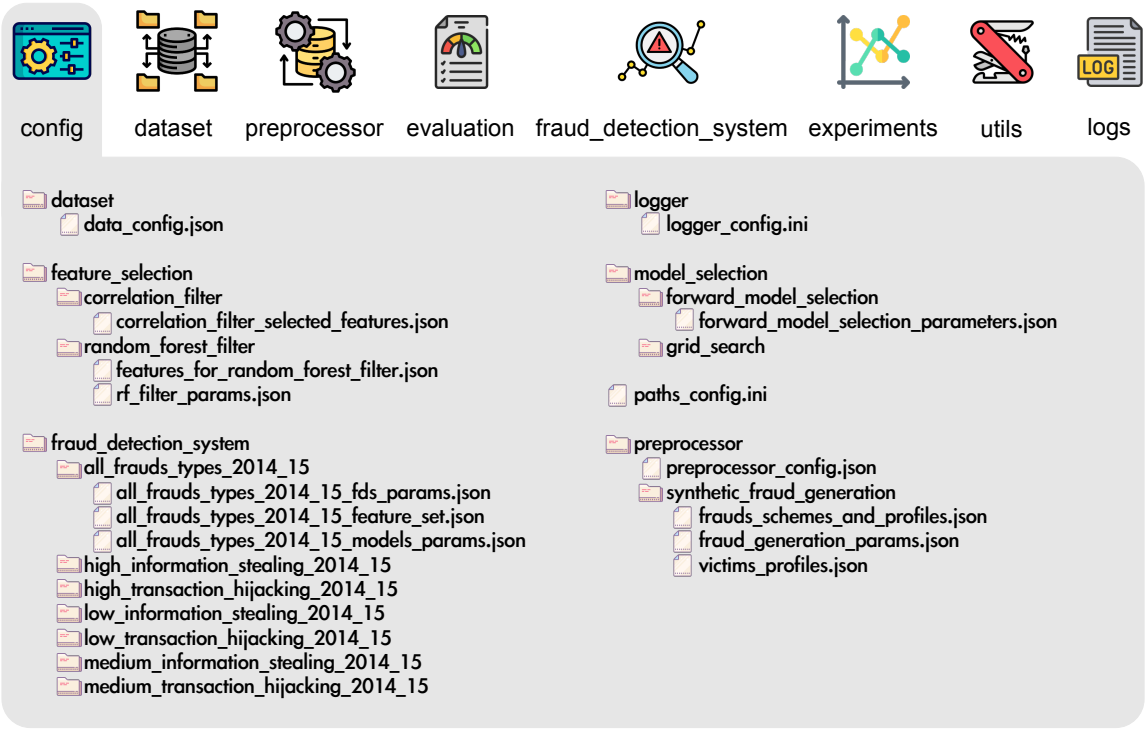


Figure 6.2: Configuration module content

As Figure 6.2 shows, the configuration folder houses numerous configuration files that enable customisation of key procedures within the system. These files provide the flexibility to tailor various aspects, including sources and results file paths, victim profiles, fraud generator parameters, model selection parameters, feature selection parameters, final detection systems features and parameters, and granularity of log messages. Researchers can fine-tune and adapt the system to their specific requirements by accessing and modifying these configuration files, ensuring a highly adaptable framework.

6.1.2. Dataset Manager Module

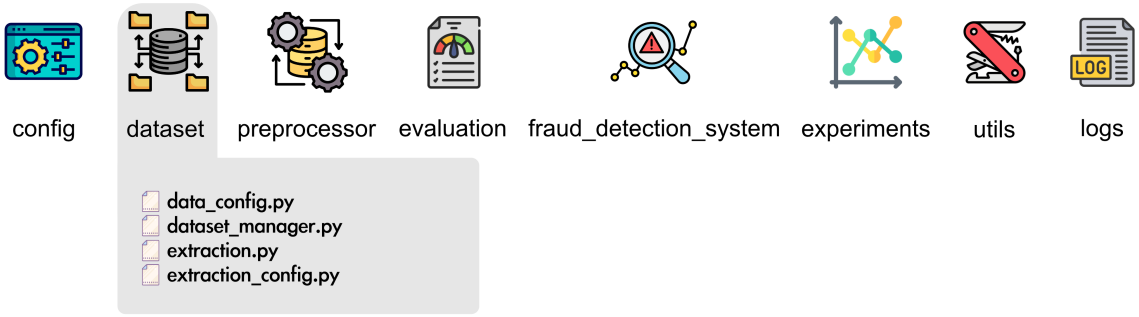


Figure 6.3: Dataset manager module content

This module encapsulates the necessary code to effectively handle the extraction, loading and saving of all databases involved in the framework procedures. It provides essential functionalities for easy access and manipulation of the data throughout the entire system. As depicted in Figure 6.3, there are four files. The ones having *config* as a suffix contain the code to read the required parameters from the configuration files described in Subsection 6.1.1. This is common to all modules, so we will avoid repeating it in the following sections. On the other hand, *extraction.py* and *dataset_manager.py* contain, respectively, the code to extract the original dataset from an SQL dump and the code to manage the previously extracted dataset.

6.1.3. Preprocessor Module

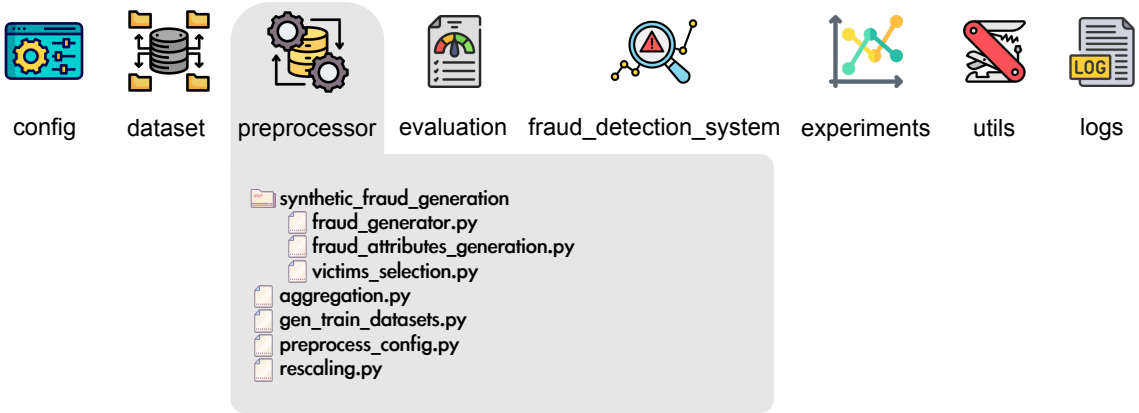


Figure 6.4: Preprocessor module content

This is the module in charge of all the preprocessing steps needed to prepare the dataset for the following computations. The `synthetic_fraud_generation` folder contains the code to perform victim selection, the implementation of the fraud generator and the logic responsible for generating each field of a fraudulent transaction. The generator is fully configurable and does not rely on hardcoded parameters. This allows for greater flexibility and adaptability over the generation of synthetic fraudulent transactions. As Figure 6.4 shows, the Preprocessor module consists of three additional files. The *aggregation.py* file handles the transaction aggregation process as described in Subsection 5.2.3, while in *rescaling.py*, we implement the data scaling procedure explained in Subsection 5.2.4. Lastly, *gen_ train_ datasets.py* contains the code we use to generate the train datasets on which we train the models we evaluate (see Section 7.4). The researchers can edit this code to personalise the way training datasets are generated according to their needs.

6.1.4. Evaluation Module

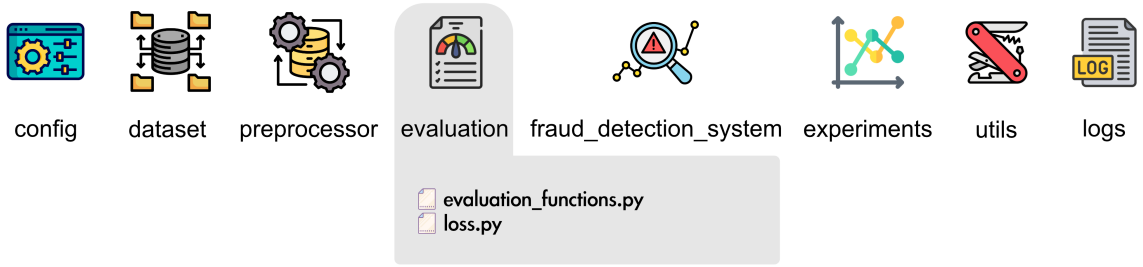


Figure 6.5: Evaluation module content

As Figure 6.5 shows, this module contains two files. Inside *loss.py*, we enclose the implementation of the loss function (see Section 7.1) we employ in our experiments. In *evaluation_function.py* instead, we implement the k-fold Cross Validation procedure together with various performance metrics, including Precision, Recall, F1-score, False Positive Rate, AUC-ROC, AUC-PRC, Matthew Correlation Coefficient, and a custom accuracy consisting of a weighted miss-prediction cost that ranks FN higher with respect to FP (see Appendix C).

6.1.5. Fraud Detection System Module

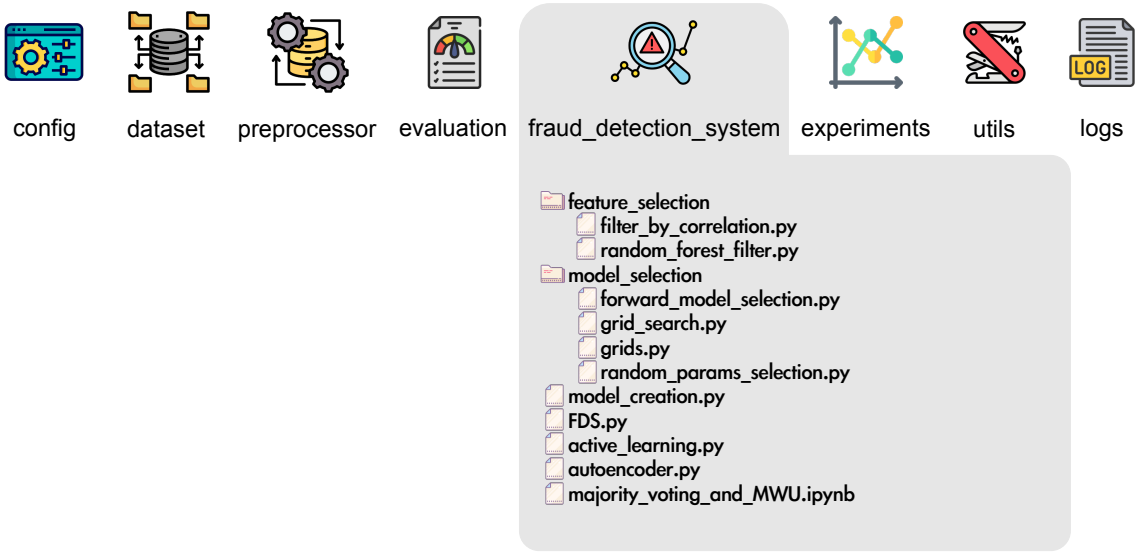


Figure 6.6: Fraud Detection System module content

This module incorporates the implementation of the fraud detection system architecture (*FDS.py*) and its underlying base models. Inside *model_creation.py* we define the models' instantiation procedures. The *majority_voting_and_MWU.ipynb* notebook contains the implementation of the two Ensemble models we defined in Subsection 2.2.3, while the *active_learning.py* file contains the implementation of the Active Learning model. The implementation we refer to is similar to the one designed by A. Dignani [28], and it consists of two main components: an Autoencoder (implemented in *autoencoder.py*) and a Random Forest model. The first component (Autoencoder) is a type of neural network that is trained to reconstruct its input data. It consists of an encoder network that maps the input data to a lower-dimensional representation (latent space) and a decoder network that reconstructs the input data from the latent space representation. The autoencoder objective is to learn how to reconstruct the legitimate class so that when a transaction deviates too much from that class, the model assigns a high anomaly score value. The second component (Random Forest) represents the supervised learning part of the model. Differently from the Autoencoder, this model is trained using both legitimate and fraudulent transactions. The final prediction of the Active Learning model is a weighted combination of the outputs from both the Autoencoder and the Random Forest. As depicted in Figure 6.6, within this module, we can also include two folders, each dedicated to a specific selection procedure as indicated by their names. The *feature_selection*

folder encompasses the implementation of the correlation filter (*filter_by_correlation.py*), discussed in Subsection 5.3.1. Meanwhile, the *model_selection* folder contains *forward_model_selection.py*, which collects the code for the model selection procedure explained in Subsection 5.3.2. These folders also include supplementary files. In particular, *grid_search.py* implements a grid search procedure, allowing further refinement of model parameters. On the other hand, *random_forest_filter.py* contains code to perform an alternative feature selection based on the feature ranking of a random forest model. Finally, *grids.py* and *random_param_selection.py* contains the necessary code to define the grids for the grid search and forward model selection procedures, respectively.

6.1.6. Experiment Module

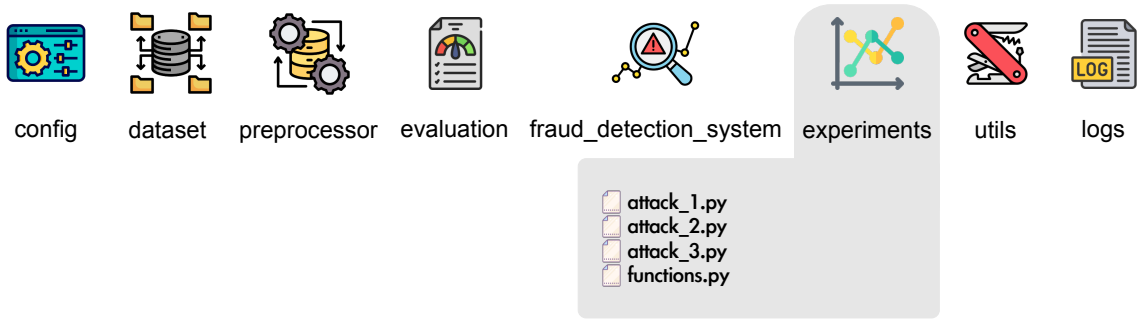


Figure 6.7: Experiments module content

As Figure 6.7 shows, this module contains the snippets of code in which we implement the necessary steps to run the attacks described in Section 7.7. In addition to that, there is a file containing functions that support the execution (*functions.py*). Within this module, researchers can place their own files containing custom code to exploit the framework's features, perform additional experiments, or develop new types of attacks.

6.1.7. Utils Module

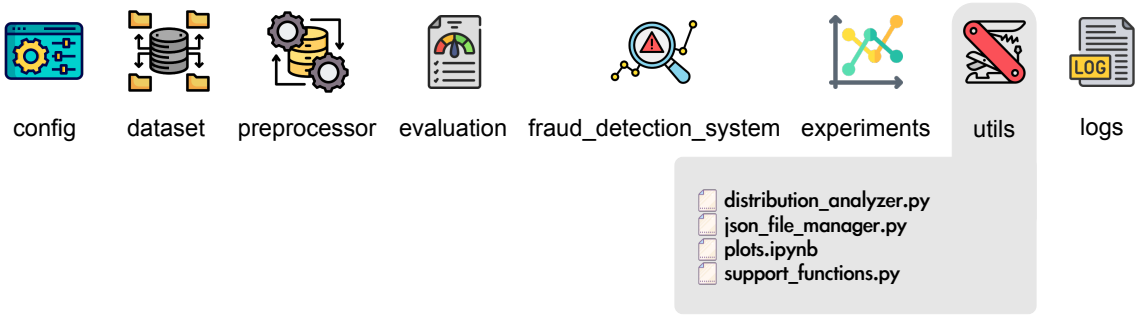


Figure 6.8: Utils module content

As we illustrate in Figure 6.8, this module contains utility functions that are common to all the other framework modules. We group functions to manage JSON files in *json_file_manager.py*, functions dedicated to the creation of plots in *plots.ipynb* notebook and functions to analyse dataset attribute distribution in *distribution_analyzer.py*. The *support_functions.py* file collects the most consistent part of the code for this module. In this file, we implement functions that support operation on dataframes, generation of reports, multiprocessing management, and scalers management.

6.1.8. Logs Module

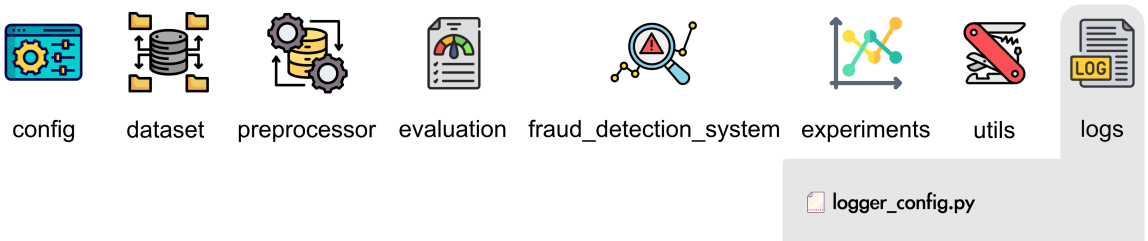


Figure 6.9: Utils module content

As depicted in Figure 6.9, this module contains only one file, in which we provide the possibility to set the granularity of log messages.

6.2. Execution Environment

We carry out the development of the entire system using Python, specifically version 3.10. To facilitate the coding process, we utilise PyCharm 2022.2.3 as our Integrated Development Environment (IDE).

We leverage a range of widely used libraries to implement and test our approach. The main ones are:

- Dask 2023.3.2: is a powerful library for parallel computing that enables scalable and efficient handling of large datasets by leveraging task scheduling and distributed computing.
- Joblib 1.2.0: is a library that provides tools for efficient caching and parallel computing, particularly useful for optimising the execution time of computationally intensive tasks, such as Machine Learning model training and evaluation.
- Keras 2.10.0: is a popular deep learning framework that provides a user-friendly interface for building and training neural networks.
- Matplotlib 3.7.1: is a widely-used data visualisation library in Python, offering a comprehensive set of tools for creating high-quality plots, charts, and graphs.
- NumPy 1.23.5: is a fundamental library for numerical computing in Python. It provides efficient data structures and functions for handling large multi-dimensional arrays and performing various mathematical operations.
- Pandas 1.5.3: is a versatile data manipulation and analysis library.
- Scikit-learn 1.2.2: is a comprehensive Machine Learning library that provides a wide array of tools and algorithms for various tasks, including classification, regression, clustering, and dimensionality reduction. It offers a user-friendly interface and robust implementations of popular Machine Learning algorithms.
- TensorFlow 2.10.0: is a widely-used deep learning framework that offers a flexible and scalable ecosystem for building and deploying Machine Learning models. It provides a range of tools and APIs for constructing neural networks, training models, and performing advanced computations on GPU and TPU architectures.
- XGBoost 1.7.5: is a powerful and widely-used gradient boosting library designed for Machine Learning tasks. It provides an efficient implementation of gradient boosting algorithms, offering high performance and scalability.

We perform all operations, from the initial stages to the final outcomes, on a machine having the following specifications:

- Model: MacBook Pro 14-inch
- OS: macOS Ventura
- Chip: Apple M1 Pro
- Memory: 16 GB
- Storage: 512 GB

7 | Experimental Validation

7.1. Evaluation Metric

While the primary goal of a fraud detection system is to detect and prevent as many fraudulent transactions as possible, it is essential to establish a threshold for the acceptable number of false positives. When a non-fraudulent transaction is incorrectly classified as fraud, there can be significant impacts for both the bank institution and the customer [60]. For the bank institution, false positives can lead to unnecessary disruptions in normal banking operations. The trigger of additional verification processes could increase operational costs, as they may need to allocate resources to investigate and resolve these cases. From the customers' perspective, the security measures adopted in response to a false positive (like account freezing and/or transaction blocking) can erode their trust in the bank, as they may perceive it as unreliable or overly cautious. This can lead to dissatisfaction and potentially drive customers to seek services from other financial institutions.

We assess the impact of frauds on each model by employing a custom loss function that takes into account both false negatives (frauds predicted as legitimate transactions) and false positives (legitimate transactions predicted as frauds). This specialised loss function assigns a specific monetary value to each type of mispredicted transaction. In particular, we estimated a cost for the bank institution of 100 € in case of a false positive ($true_{label} = 0$ and $predicted_{label} = 1$), and in the case of a false negative ($true_{label} = 1$ and $predicted_{label} = 0$), the loss is equal to the actual amount of the transaction.

$$Loss = (true_{label} - predicted_{label})^2 * (100 * predicted_{label} + Amount * true_{label})$$

7.2. Goals

The primary objective of our experimental validation is to conduct a comprehensive performance evaluation of multiple Fraud Detection Systems (FDSs) based on different algorithms and under different training settings. We build all of the FDSs using FraudBench.

Our goals can be summarised as follows:

- Study the impact of different types of fraudulent activities on various FDSs.
- Study the performance of both FDSs trained on distinct fraud types and model ensemble strategies in countering attackers that change their fraudulent behaviour.

To accomplish these goals, we design three distinct experiments, each focusing on exposing FDSs to a different attack scenario.

7.3. Experimental Settings

To achieve our research objectives effectively, we select a set of models comprising Random Forest (RF) [17], Logistic Regression (LR) [24], Support Vector Machine (SVM) [23], Neural Network (NN) [16], XGBoost (XGB) [22], and a variant of Active Learning (AL) [28, 41, 65]. The selection of these models is based on their proven effectiveness in the literature and their ability to address various challenges faced by financial institutions. To explore the impact of different fraudulent activities on these models, we adopt a systematic approach. Starting from a common dataset, we augment it by introducing two distinct schemes of fraud: Information Stealing (IS) and Transaction Hijacking (TH). As we explained in Section 5.2.2.2, we further subdivide each fraud scheme into three fraud profiles: Low (L), Medium (M), and High (H). This categorisation leads to the creation of six unique datasets, each representing a specific combination of fraud schemes and profiles. We refer to these datasets as augmented_*fraudProfile*_*fraudType* (for instance, the name for the dataset containing only High-profile Information Stealing frauds is `augmented_H_IS`). To optimise the performance and efficiency of our models, we employ a two-step process for each dataset. Initially, we perform feature reduction by evaluating the correlation among the features, resulting in a refined set of features specific to each dataset. Then, by applying a model selection procedure to each model-dataset combination, we fine-tune the models to capture the unique patterns and dynamics associated with different fraud schemes and profiles. This procedure encompasses the selection of both features and hyperparameters. As a result of this approach, we develop a total of 36 distinct models, corresponding to the Cartesian product of the six datasets (`augmented_H_IS`, `augmented_H_TH`, `augmented_M_IS`, `augmented_M_TH`, `augmented_L_IS`, and `augmented_L_TH`) and the six model types (RF, LR, SVM, NN, AL, and XGB). To provide a comprehensive comparison between models trained on specific types of fraud and models exposed to a broader range of frauds, we expand our analysis by including an additional set of six models. These models undergo the same procedure as the other 36 models, following the same methodology. However, in this case,

we train them on a comprehensive dataset that includes instances of all types of fraud (`augmented_AF`). We state models performances by testing their resistance to three types of attacks, in which we generate frauds each time following a different policy. In addition to the range of individual models, we explore two of ensemble models based on Majority Voting (MV) [57] and Multiplicative Weight Update (MWU) [8]. In their prediction process, the latter combine the outcomes of all 42 individual models. We compare the ensemble models against a baseline model which is the empirical mean of the losses of the base FDS learners. In all attacks scenarios, the loss is calculated according to the metric defined in Section 7.1.

7.4. Experimental Datasets

To recreate realistic and representative datasets for training and evaluating our fraud detection models, we initially cleaned the original dataset (from now on, we refer to it as `original_14_15`) by removing all fraudulent transactions that were already present, retaining only legitimate transactions. One of the objectives of this research is to evaluate the performance of models trained on individual types of fraud. To achieve this, it was essential to begin with a pristine dataset, free from any previous influences or biases. Following a data generation approach, we used the configurable fraud generator described in Section 5.2.2.2 to synthesise various types of fraudulent transactions, aiming to achieve a target fraction of 1%. The `original_14_15` dataset covers 18 weeks; we use the first six weeks to train our detection systems and the following 12 weeks to analyse their behaviour. According to that, starting from the cleaned `original_14_15` dataset, we generate seven new training datasets, one for each combination of fraud scheme and profile. We also produce one more training dataset that includes all fraud schemes and profiles. Finally, we generate three test datasets, one for each attack we present in Section 7.7. Table 7.1 shows the characteristics of the resulting datasets. Datasets containing frauds of the type of TH have a slightly lower fraud ratio compared to those containing only frauds of type IS because frauds belonging to the former scheme depend on users' original transactions.

Dataset ID	Transactions	Frauds Ratio	Time Window
augmented_H_IS	171128	1%	6 weeks
augmented_H_TH	169804	0.93%	6 weeks
augmented_M_IS	171128	1%	6 weeks
augmented_M_TH	169829	0.97%	6 weeks
augmented_L_IS	171122	0.99%	6 weeks
augmented_L_TH	169819	0.96%	6 weeks
augmented_AF	170852	0.93%	6 weeks
attack_1	303928	0.79%	12 week
attack_2	303567	0.69%	12 week
attack_3	303543	0.70%	12 week

Table 7.1: Experimental datasets description

7.5. Feature Engineering

As discussed in Subsection 5.2.3, relying solely on raw features does not provide sufficient information for conducting an accurate analysis. Therefore, following the augmentation process, we provide each training dataset as input to the Aggregation module to perform transaction aggregation and extract new potential features. After the execution of the aggregation phase, all seven datasets contain a substantial number of features, reaching a total of 205. However, depending on the specific dataset, not all of them are essential, as they can potentially be redundant. We initially streamline our feature sets by exploiting the Features Filtering module detailed in Subsection 5.3.1. To determine which feature to remove in a correlated pair, we set the correlation threshold to 95%. This means that if the correlation coefficient between two features exceeds 95%, we remove from the feature set the feature between the two that is less correlated with the target variable. We show selected feature sets for each dataset in Table B.1 (for improved layout and organisation, we place the table in Appendix B).

7.6. Models Tuning and Performance Evaluation

To fine-tune our models and improve their predictive capabilities, we use the Forward Model Selection approach we described in Subsection 5.3.2. With this step-by-step model selection strategy, we aim to find the optimal combination of features and hyperparameters that maximises predictive capabilities. In each step, we evaluate a range of 8 to 13 different model configurations using 3-fold cross-validation on their specific training dataset. We then save the feature set and the hyperparameters that results in the highest F1-score value. To ensure a thorough exploration of the hyperparameters space, we randomly select model parameters from a predefined grid. The iterative procedure continues until we observe that adding new features no longer significantly improves the F1-score. To determine this, we set a criterion for improvement, requiring the F1-score to increase by at least 0.001 within five consecutive iterations. If we do not meet this condition, we conclude the iterative process. This hyperparameter optimisation step represents the most time-consuming part of the entire benchmarking process. Depending on the specific model and chosen parameters, the optimisation process can span up to 4 days for a single model on our datasets. Upon completion of the process, we identify the feature set and corresponding model hyperparameters that achieved the highest F1-score. In Table 7.2 we show the FDSs performances in terms of standard metrics after the tuning phase. To identify the classifiers, we adopt a naming convention where we formulate the name by combining the acronym of *base model*, *fraud profile*, and *fraud scheme* associated with the system. For instance, if we consider a detection system that uses Random Forest as its base model and has been trained to identify High-profile Information Stealing frauds, we name the system as `RF_H_IS`.

Model ID	Precision	Recall	F1-score	FPR	AUC ROC	AUC PR	Weighted MCC	Cost Accuracy
AL_AF	0.315018	0.849592	0.453519	0.018375	0.925664	0.588637	0.839013	0.915608
AL_H_IS	0.856352	0.930158	0.890566	0.001455	0.981805	0.945657	0.930929	0.964351
AL_H_TH	0.731097	0.734423	0.730754	0.002536	0.920886	0.759885	0.759545	0.865943
AL_M_IS	0.721	0.798346	0.757472	0.003031	0.922896	0.730119	0.811684	0.897658
AL_M_TH	0.311155	0.648562	0.420153	0.014693	0.826008	0.341087	0.673183	0.816935
AL_L_IS	0.533036	0.728879	0.60744	0.006214	0.875732	0.585513	0.749513	0.861333
AL_L_TH	0.26301	0.600862	0.364345	0.016073	0.793326	0.283064	0.63345	0.792394
LR_AF	0.242985	0.735252	0.362694	0.022416	0.894705	0.385875	0.734977	0.856418
LR_H_IS	0.865066	0.984618	0.91966	0.001495	0.999605	0.982592	0.983307	0.991561
LR_H_TH	0.568807	0.948595	0.710193	0.006917	0.994936	0.688127	0.942635	0.970839
LR_M_IS	0.293978	0.826351	0.424531	0.020603	0.964061	0.537287	0.815594	0.902874
LR_M_TH	0.280937	0.556615	0.360585	0.01642	0.857763	0.32617	0.598233	0.770098
LR_L_IS	0.276432	0.831425	0.405876	0.021886	0.9558	0.494474	0.818855	0.904769
LR_L_TH	0.366515	0.611133	0.450132	0.010128	0.90479	0.264415	0.64944	0.800502
NN_AF	0.539119	0.550442	0.541275	0.004789	0.969304	0.486664	0.609396	0.772826
NN_H_IS	0.861738	0.968923	0.90925	0.001513	0.997172	0.976577	0.967879	0.983705
NN_H_TH	0.791664	0.826034	0.808364	0.002001	0.998487	0.840746	0.836665	0.912017
NN_M_IS	0.754639	0.841236	0.793287	0.002636	0.994727	0.824939	0.849325	0.9193
NN_M_TH	0.681768	0.594244	0.630509	0.003054	0.982387	0.546867	0.645993	0.795595
NN_L_IS	0.429607	0.708046	0.523592	0.009142	0.965543	0.573207	0.728943	0.849452
NN_L_TH	0.431277	0.528754	0.472295	0.006599	0.91745	0.34024	0.589986	0.761078
RF_AF	0.893243	0.748192	0.810639	0.000886	0.991114	0.859549	0.772932	0.873653
RF_H_IS	0.974355	0.988425	0.981266	0.000236	0.999976	0.997378	0.988256	0.994095
RF_H_TH	0.850972	0.920908	0.884356	0.001526	0.999482	0.942113	0.922184	0.959691
RF_M_IS	0.939785	0.892944	0.915688	0.000511	0.998498	0.965747	0.897596	0.946216
RF_M_TH	0.735426	0.629847	0.678125	0.00234	0.979833	0.684832	0.674849	0.813753
RF_L_IS	0.831233	0.766699	0.79258	0.001475	0.9828	0.843984	0.787108	0.882612
RF_L_TH	0.558306	0.517505	0.52967	0.004001	0.898616	0.455515	0.584842	0.756752
SVM_AF	0.215407	0.771153	0.331975	0.028145	0.950983	0.422872	0.759164	0.871504
SVM_H_IS	0.693491	1.0	0.812694	0.004186	0.999852	0.979406	0.995824	0.997907
SVM_H_TH	0.423082	1.0	0.593886	0.01302	0.997945	0.808169	0.987067	0.99349
SVM_M_IS	0.234123	0.944962	0.371794	0.030269	0.982044	0.587418	0.915033	0.957346
SVM_M_TH	0.223877	0.61394	0.327961	0.021412	0.855594	0.211329	0.636355	0.796264
SVM_L_IS	0.23894	0.790363	0.353879	0.025811	0.929242	0.317643	0.779548	0.882276
SVM_L_TH	0.214633	0.609607	0.316942	0.021026	0.86979	0.251579	0.633535	0.794291
XGB_AF	0.830813	0.78581	0.805235	0.001556	0.983755	0.828596	0.803364	0.892127
XGB_H_IS	0.971563	0.992152	0.981706	0.000275	0.99996	0.996012	0.991906	0.995938
XGB_H_TH	0.839137	0.942169	0.887106	0.001744	0.999546	0.951437	0.941927	0.970213
XGB_M_IS	0.907331	0.938287	0.922467	0.000905	0.999127	0.971729	0.939406	0.968691
XGB_M_TH	0.646875	0.644709	0.645596	0.003529	0.984189	0.660626	0.685131	0.82059
XGB_L_IS	0.798892	0.829322	0.806398	0.001926	0.992705	0.885799	0.840852	0.913698
XGB_L_TH	0.596887	0.476124	0.52255	0.00311	0.928435	0.436094	0.5541	0.736507

Table 7.2: Models performance after hyperparameters tuning

7.7. Experiments

In this section, we present the experiments we perform to assess the behaviour of the models. We design each experiment to evaluate the performance of the models under different attack scenarios. We build our test dataset by injecting fraudulent transactions on a weekly basis, and at the end of each week, we calculate the loss using the loss function defined in Section 7.1.

7.7.1. Attack 1

In this experiment, we simulate an attacker who commits fraudulent transactions by following a random policy, that is, randomly choosing fraud schemes and profiles. Figure 7.1 illustrates the loss trend of the 42 detection systems under test over the 12-week simulation period. We use colours to distinguish between different base-model families. For example, we represent detection systems built on Random Forest in green, while those built on Support Vector Machine in yellow, and so on.
In the long run, the best-performing detection system is the one based on the random forest model and trained on all types of fraud (`RF_AF`), which registered an overall loss of 1,052,771.86 €. As Table 7.3 shows, this model has the highest F1-score value (∼80%) and even if it is not the one that manages to detect the greatest number of frauds, probably the fact that it keeps the rate of false positives very low (0.08%), lead it to be the best overall. The model with the highest number of detected frauds is the SVM trained on frauds belonging to the Information Stealing scheme and having a High-profile (`SVM_H_IS`), which scores a recall of 84.3%. However, at the same time, it has a very low precision (13.9%), resulting in a high false positive rate (2.6%) and a greater loss (2,373,895.89 €) when compared to the previous model. All the detection systems trained on all types of fraud, except for `AL_AF`, exhibits relatively low overall losses regardless of the base model used. The high losses recorded by `AL_AF` could be due to the extremely high number of false positives. Analysing the Table 7.3 we note that this behaviour is common to all systems based on this base model. Unsurprisingly, almost all systems trained exclusively on High-profile Information Stealing fraud instances manages to keep the monetary loss to a low level. This outcome was expected since a comprehensive overview of the graph reveals that this specific type of fraud results in the highest increase in terms of monetary losses. Therefore, having the ability to effectively mitigate this type of fraud presents a significant advantage.
Regarding ensemble models, as we depict in Figure 7.2, both Majority Voting (to which

we refer as `MV`) and Multiplicative Weight Update (to which we refer as `MWU`) approaches achieve significantly lower losses compared to the average loss of the individual models (*Mean* in the figure). The `MV` ensemble achieves a final loss of 7,270,732.92 €, which represents a 49% reduction when compared to *Mean*. This ensemble demonstrates exceptional precision, correctly identifying a transaction as fraudulent in 95% of the cases. However, it falls short in terms of recall, only capturing 37.6% of the actual number of fraudulent activities. On the other hand, `MWU` exhibits a slightly lower precision (88%) but, at the price of an increased false positive rate (0,05% compared to 0.01% of `MV`), can identify a larger number of fraudulent transactions, reaching a recall of 76.5% and a final loss of 1,135,369.99 €, 92% lower compared to *Mean*.

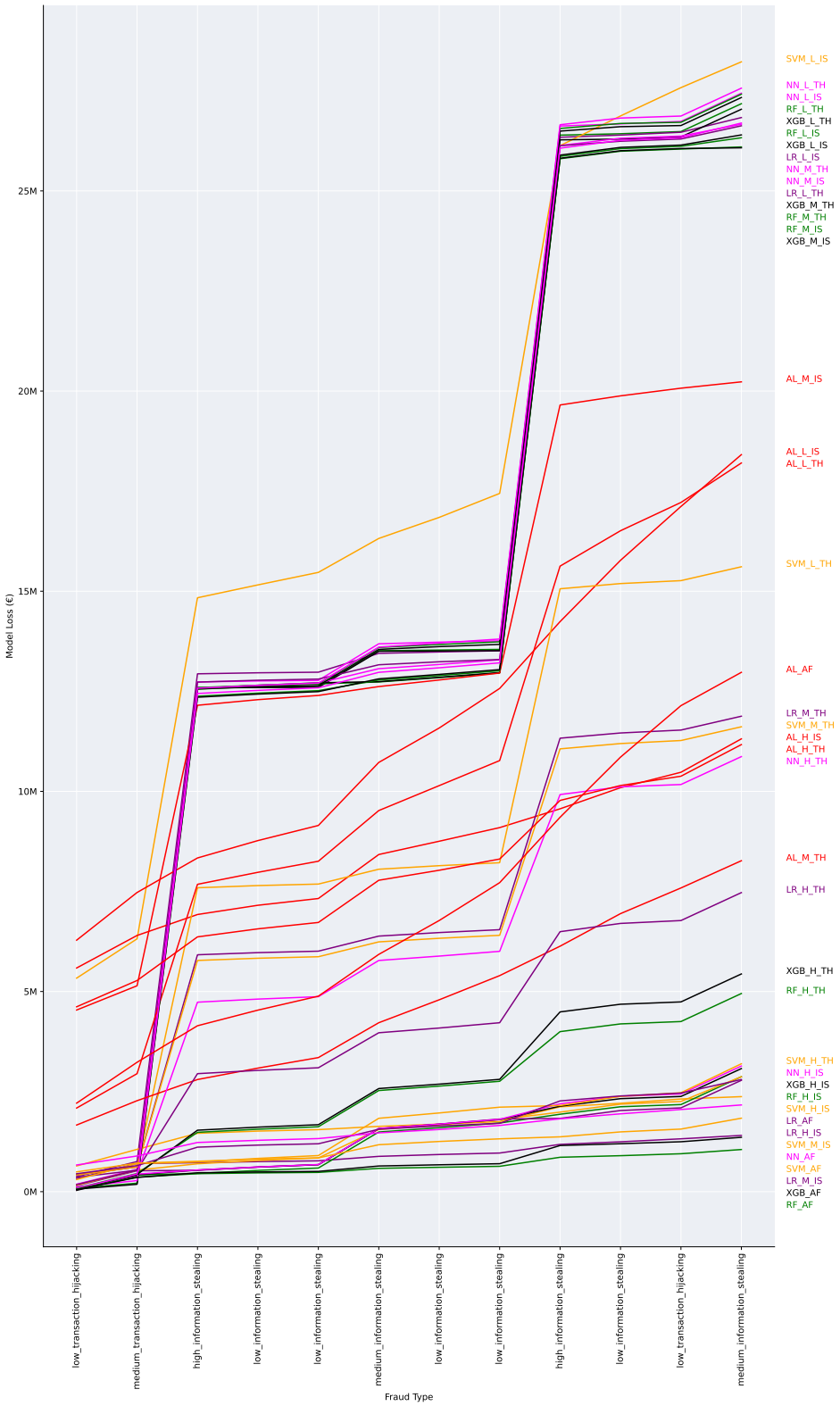


Figure 7.1: Attack 1: models losses (in Millions of Euros) injecting fraud types following a random policy

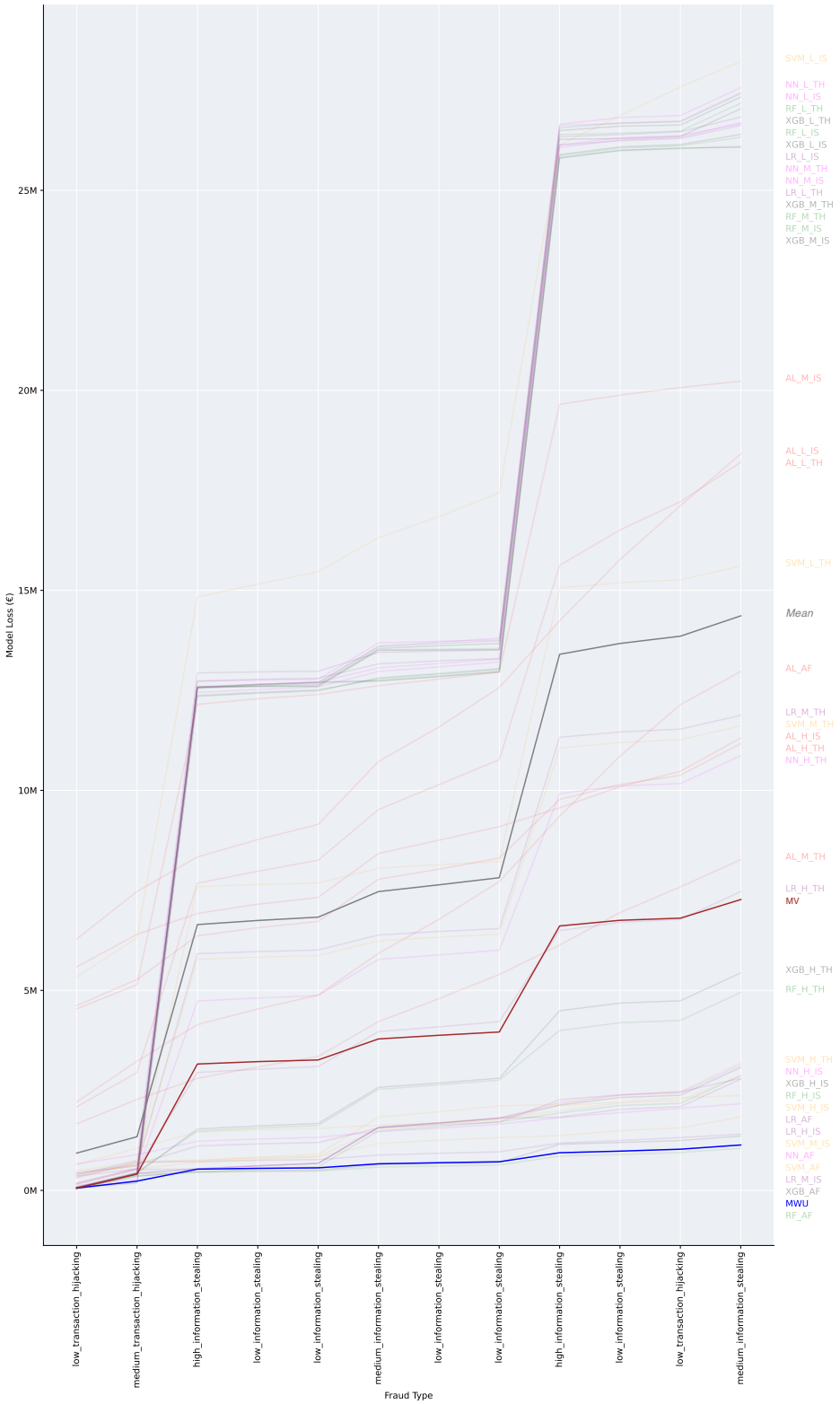


Figure 7.2: Attack 1: ensemble models losses (in Millions of Euros) injecting fraud types following a random policy

Model ID	Precision	Recall	F1-score	FPR	Weighted MCC	Cost Accuracy
AL_AF	0.0159	0.8141	0.0311	0.2561	0.5593	0.7790
AL_H_IS	0.0084	0.3103	0.0164	0.1856	0.1443	0.5623
AL_H_TH	0.0093	0.2550	0.0179	0.1378	0.1476	0.5586
AL_L_IS	0.0099	0.6960	0.0196	0.3513	0.3451	0.6724
AL_L_TH	0.0130	0.4489	0.0252	0.1730	0.2980	0.6380
AL_M_IS	0.0205	0.5302	0.0395	0.1283	0.4275	0.7009
AL_M_TH	0.0173	0.5142	0.0334	0.1485	0.3885	0.6829
LR_AF	0.1663	0.7014	0.2689	0.0178	0.7122	0.8418
LR_H_IS	0.2303	0.2751	0.2507	0.0047	0.3899	0.6352
LR_H_TH	0.1521	0.2190	0.1795	0.0062	0.3366	0.6064
LR_L_IS	0.1704	0.4816	0.2517	0.0119	0.5447	0.7348
LR_L_TH	0.1449	0.4523	0.2195	0.0135	0.5190	0.7194
LR_M_IS	0.2700	0.7688	0.3997	0.0105	0.7775	0.8792
LR_M_TH	0.1455	0.6030	0.2345	0.0180	0.6322	0.7925
NN_AF	0.1226	0.8400	0.2140	0.0305	0.8164	0.9048
NN_H_IS	0.5751	0.2550	0.3534	0.0010	0.3803	0.6270
NN_H_TH	0.4022	0.1713	0.2402	0.0013	0.3027	0.5850
NN_L_IS	0.2371	0.3317	0.2765	0.0054	0.4358	0.6631
NN_L_TH	0.4348	0.0628	0.1098	0.0004	0.1783	0.5312
NN_M_IS	0.6416	0.1214	0.2042	0.0003	0.2532	0.5605
NN_M_TH	0.1759	0.1290	0.1488	0.0031	0.2535	0.5630
RF_AF	0.8360	0.7580	0.7951	0.0008	0.7803	0.8786
RF_H_IS	0.8838	0.2580	0.3994	0.0002	0.3844	0.6289
RF_H_TH	0.4001	0.2366	0.2974	0.0018	0.3623	0.6174
RF_L_IS	0.7523	0.3472	0.4751	0.0006	0.4572	0.6733
RF_L_TH	0.2969	0.1876	0.2299	0.0023	0.3162	0.5927
RF_M_IS	0.8060	0.1809	0.2955	0.0002	0.3148	0.5903
RF_M_TH	0.3423	0.1595	0.2177	0.0016	0.2903	0.5790
SVM_AF	0.1620	0.7182	0.2644	0.0188	0.7249	0.8497
SVM_H_IS	0.2222	0.2910	0.2520	0.0052	0.4024	0.6429
SVM_H_TH	0.0960	0.2609	0.1403	0.0125	0.3616	0.6242
SVM_L_IS	0.0133	0.6436	0.0260	0.2425	0.4037	0.7005
SVM_L_TH	0.1366	0.5729	0.2206	0.0184	0.6076	0.7773
SVM_M_IS	0.1390	0.8430	0.2387	0.0265	0.8235	0.9082
SVM_M_TH	0.1429	0.6122	0.2317	0.0186	0.6387	0.7968
XGB_AF	0.8317	0.7450	0.7860	0.0008	0.7695	0.8721
XGB_H_IS	0.7864	0.2559	0.3861	0.0004	0.3823	0.6278
XGB_H_TH	0.5416	0.2316	0.3244	0.0010	0.3596	0.6153
XGB_L_IS	0.7916	0.3865	0.5194	0.0005	0.4885	0.6930
XGB_L_TH	0.4844	0.2207	0.3032	0.0012	0.3495	0.6097
XGB_M_IS	0.5811	0.1876	0.2836	0.0007	0.3200	0.5935
XGB_M_TH	0.3762	0.1495	0.2140	0.0013	0.2808	0.5741
MV	0.9544	0.3765	0.5399	0.0001	0.4814	0.6882
MWU	0.8869	0.7655	0.8218	0.0005	0.7868	0.8825

Table 7.3: Attack 1: models performance at the end of the 12 weeks

7.7.2. Attack 2

In this experiment, we simulate attackers who commit fraudulent transactions according to the policy that causes the greatest impact on the best-performing model of every week. We aim to simulate a situation in which the attacker exerts maximum pressure on the best-performing model to test its resistance. To accomplish this, we adopt a dynamic approach. Except for the first week, during which we randomly select the type of fraud to inject, for each subsequent week, we generate a set of fraudulent transactions for each type of fraud. We then evaluate their impact on the model that shows the best performance until that point. Finally, we select the set containing the fraud instances that results in the most significant increase in the estimated loss, and we add them definitively to the test set.

Surprisingly, in the long run, the SVM trained on all fraud types (`SVM_AF`) emerges as the best-performing model despite its simplicity. Even subjecting it to the most challenging fraud campaigns, this model manages to maintain an overall estimated loss just below 5 million Euros (4,977,643.33 €). However, as we show in Table 7.4, this detection system has a very low precision (14.4%) and a high false positive rate (∼2%). The detection system with the highest F1-score (84.1%) is `RF_AF`. When comparing its standard metric values with those of the overall best system, `RF_AF` outperforms it in every aspect. Nevertheless, it does not prove to be the best when we evaluate it using our specific loss function. This discrepancy could be attributed to the monetary value we place on false negatives. A fraudulent transaction that is incorrectly predicted as legitimate incurs a monetary cost equal to its amount. When analysing the false negatives amounts of `SVM_AF` and `RF_AF`, we observe that, on average, the latter's false negatives amounts are approximately three times higher (26,886.44 € vs 1,656.15 €). This probably explains why, at the end of the period, the loss of `RF_AF` is higher.

Regarding the ensemble models, as illustrated in Figure 7.4, `MWU` once again demonstrates superior performance with respect to `MV`, highlighting the effectiveness of an online learning approach in this context. `MWU`, despite having a low precision (23.8%) and a 77 times higer FPR compared to `MV`, results in an estimated loss of 511,762.51 €, only 2.8% higher with respect to the best-performing model (`SVM_AF`) and 75% lower than `MV`. This time, the `MV` ensemble fails to keep the overall estimated loss below the average loss of the individual models, surpassing it between weeks 8 and 9. At the end of the 12-week period, `MV` registers an estimated loss of 20,538,228.16 €, which is a 5.62% increase compared to *Mean*.

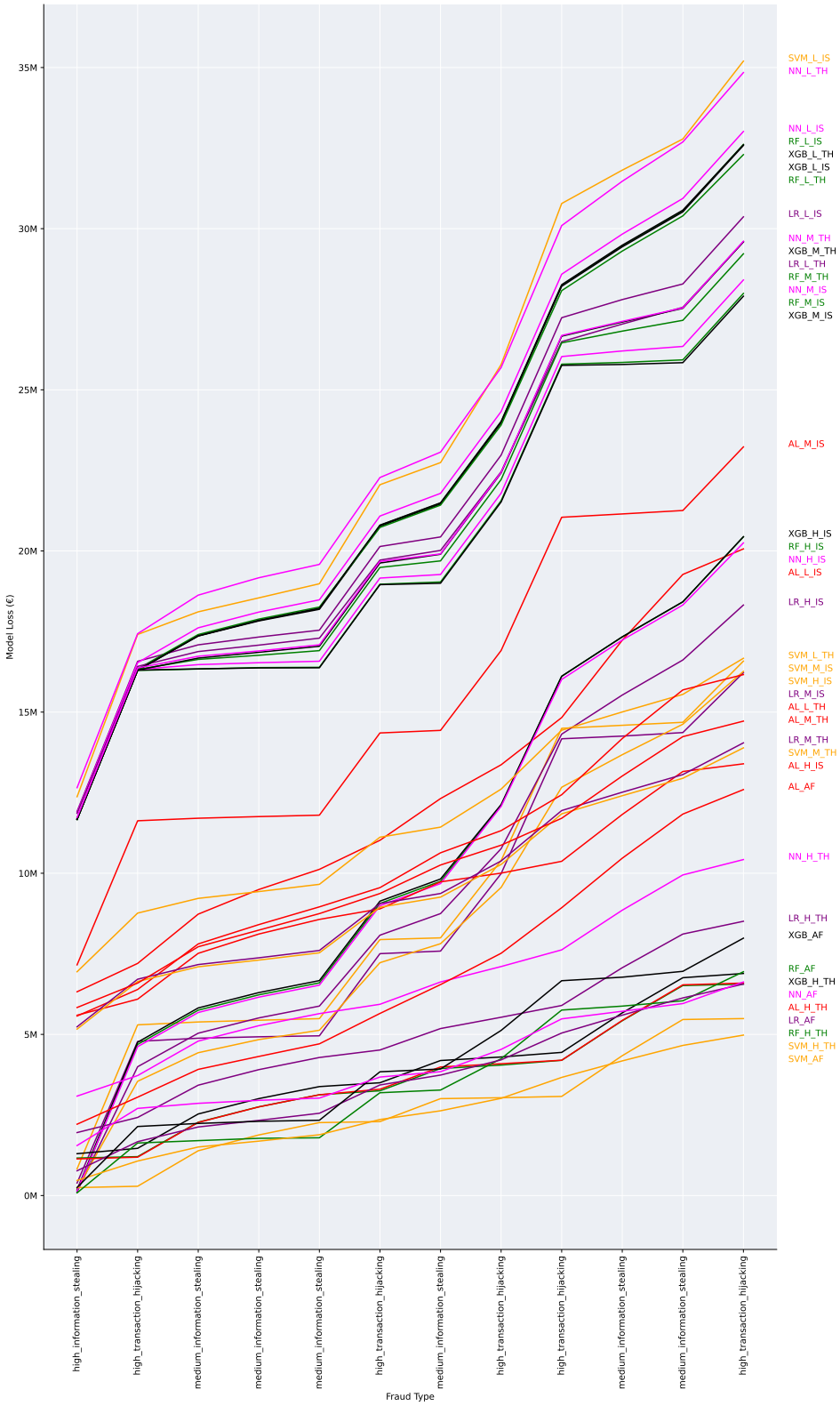


Figure 7.3: Attack 2: models losses (in Millions of Euros) injecting fraud types with the highest impact on the previous week's best model

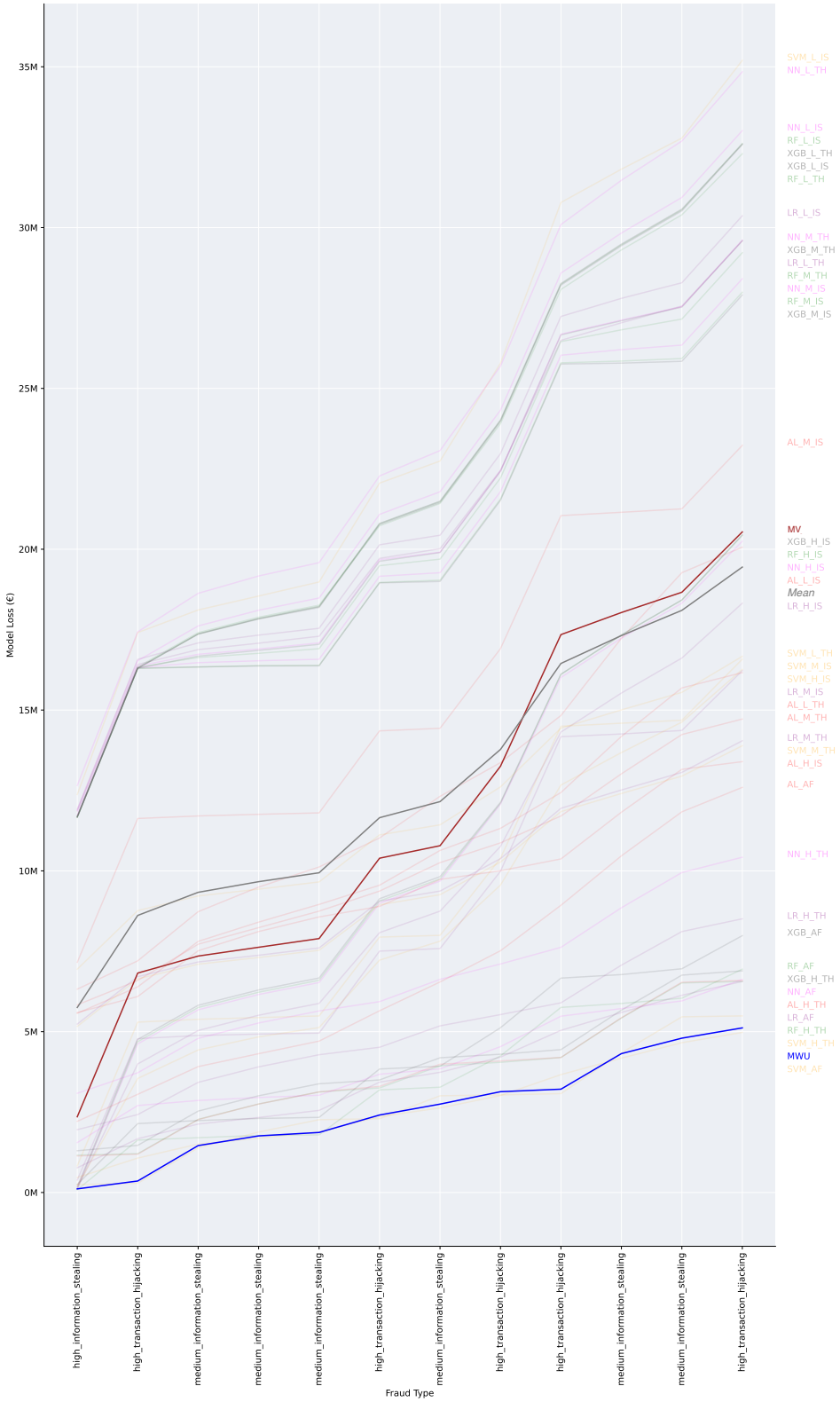


Figure 7.4: Attack 2: ensemble models losses (in Million of Euros) injecting fraud types with the highest impact on the previous week's best model

Model ID	Precision	Recall	F1-score	FPR	Weighted MCC	Cost Accuracy
AL_AF	0.0161	0.9408	0.0316	0.2563	0.6982	0.8423
AL_H_IS	0.0093	0.3885	0.0181	0.1851	0.2249	0.6017
AL_H_TH	0.2244	0.3131	0.2614	0.0048	0.4216	0.6542
AL_L_IS	0.0069	0.5456	0.0136	0.3500	0.1966	0.5978
AL_L_TH	0.0110	0.4339	0.0215	0.1730	0.2837	0.6304
AL_M_IS	0.5491	0.6668	0.6023	0.0024	0.7040	0.8322
AL_M_TH	0.0115	0.6926	0.0227	0.2645	0.4285	0.7140
LR_AF	0.1442	0.6754	0.2376	0.0178	0.6909	0.8288
LR_H_IS	0.1517	0.1871	0.1676	0.0047	0.3098	0.5912
LR_H_TH	0.1851	0.3160	0.2335	0.0062	0.4214	0.6549
LR_L_IS	0.1276	0.3905	0.1923	0.0119	0.4722	0.6893
LR_L_TH	0.1158	0.3986	0.1794	0.0135	0.4760	0.6925
LR_M_IS	0.2443	0.7652	0.3704	0.0105	0.7744	0.8773
LR_M_TH	0.1263	0.5833	0.2076	0.0180	0.6165	0.7827
NN_AF	0.1552	0.8401	0.2620	0.0203	0.8279	0.9099
NN_H_IS	0.6208	0.1484	0.2396	0.0004	0.2820	0.5740
NN_H_TH	0.3276	0.2654	0.2932	0.0024	0.3861	0.6315
NN_L_IS	0.0000	0.0000	0.0000	0.0091	-0.0676	0.4955
NN_L_TH	0.0000	0.0000	0.0000	0.0479	-0.1566	0.4760
NN_M_IS	0.6696	0.5776	0.6202	0.0013	0.6354	0.7881
NN_M_TH	0.3552	0.4296	0.3889	0.0035	0.5173	0.7131
RF_AF	0.8334	0.8477	0.8405	0.0008	0.8569	0.9235
RF_H_IS	0.7902	0.1456	0.2459	0.0002	0.2797	0.5727
RF_H_TH	0.4355	0.3112	0.3630	0.0018	0.4258	0.6547
RF_L_IS	0.0000	0.0000	0.0000	0.0006	-0.0170	0.4997
RF_L_TH	0.0093	0.0048	0.0063	0.0023	0.0213	0.5013
RF_M_IS	0.9263	0.6243	0.7459	0.0002	0.6734	0.8121
RF_M_TH	0.5699	0.4611	0.5098	0.0015	0.5449	0.7298
SVM_AF	0.1447	0.7165	0.2408	0.0188	0.7234	0.8488
SVM_H_IS	0.2029	0.2955	0.2406	0.0052	0.4062	0.6452
SVM_H_TH	0.1056	0.3308	0.1601	0.0125	0.4221	0.6592
SVM_L_IS	0.0101	0.5585	0.0199	0.2425	0.3224	0.6580
SVM_L_TH	0.1187	0.5561	0.1957	0.0184	0.5942	0.7689
SVM_M_IS	0.1191	0.8043	0.2074	0.0265	0.7892	0.8889
SVM_M_TH	0.1234	0.5895	0.2041	0.0186	0.6205	0.7854
XGB_AF	0.8283	0.8315	0.8299	0.0008	0.8427	0.9154
XGB_H_IS	0.6489	0.1456	0.2378	0.0004	0.2792	0.5726
XGB_H_TH	0.5798	0.3069	0.4014	0.0010	0.4238	0.6530
XGB_L_IS	0.0359	0.0043	0.0077	0.0005	0.0386	0.5019
XGB_L_TH	0.0035	0.0010	0.0015	0.0012	-0.0036	0.4999
XGB_M_IS	0.8062	0.6415	0.7145	0.0007	0.6863	0.8204
XGB_M_TH	0.5935	0.4105	0.4853	0.0013	0.5061	0.7046
MV	0.9708	0.4129	0.5794	0.0001	0.5100	0.7064
MWU	0.2378	0.5375	0.3297	0.0077	0.5949	0.7649

Table 7.4: Attack 2: models performance at the end of the 12 weeks

7.7.3. Attack 3

In this experiment, we systematically apply all fraud types cyclically to examine the impact of each type on the detection systems we analyse. We inject each different type of fraud one after the other, in a defined order. In the first week, we introduce frauds that belong to the Information Stealing scheme and have a Low profile. In the second week, instead, we move to another type of fraud, i.e., Medium-profile Information Stealing, and so on. Once we have processed all six fraud types, we start again from the first one, and we repeat the same procedure to cover the remaining weeks.
As Figure 7.6 shows, High-profile Information Stealing frauds are the ones that contribute the most to the increase in loss for most of the models, with an average increase of 5,688,022.48 € compared to the week before their introduction. Following them, we find High-profile Transaction Hijacking frauds that cause an average increase of 1,698,591.52 €, Medium-profile Information Stealing frauds that lead to an average increase of 672,657.10 €, Medium-profile Transaction Hijacking frauds that results in an average increase of 280,355.32 €, Low-profile Information Stealing frauds that contributes an average increase of 151,758.85 €, and finally, Low-profile Transaction Hijacking frauds that leads to an average increase of 122,322.37 €. In Figure 7.5 we illustrate, for each fraud type, how many models on average are able to correctly identify a fraud belonging to such type. About half of the models in our study can accurately identify frauds having a High profile and belonging to the Information Stealing scheme. However, given the high amounts associated with such transactions, it is not surprising to see that they exert the greatest impact on the overall losses. This is due to the fact that our custom loss function estimates the loss for each undetected fraud as equal to the amount of the transaction (see Section 7.1). Consequently, when these fraudulent transactions evade detection, they have a significant repercussion on the estimated loss. Consistently with previous findings, models trained on all fraud types exhibit lower losses, as do most models trained specifically on High-profile frauds. The system that manages to maintain the lowest monetary loss for almost the entire attack duration is `RF_AF`, which reported a final loss of 2,984,247.35 €. It is outperformed by `SVM_AF` only in the last week of testing, with a relatively small margin of 105,491.97 €. When considering ensemble models, in Figure 7.7, we can observe that, once again, `MWU` emerged as the best-performing approach. While both techniques achieves lower losses when compared to *Mean*, the difference between them remains highly pronounced. The loss incurred by `MV` (12,369,723.52 €) is approximately 4.2 times higher than the loss incurred by `MWU` (2,944,925.48 €).

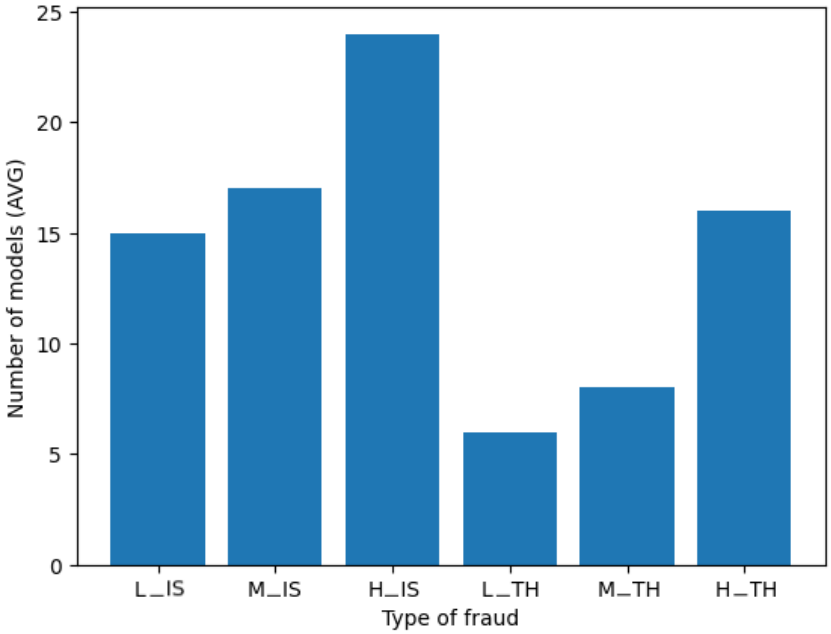


Figure 7.5: Average number of models able to correctly identify a fraudulent transaction belonging to a specific type

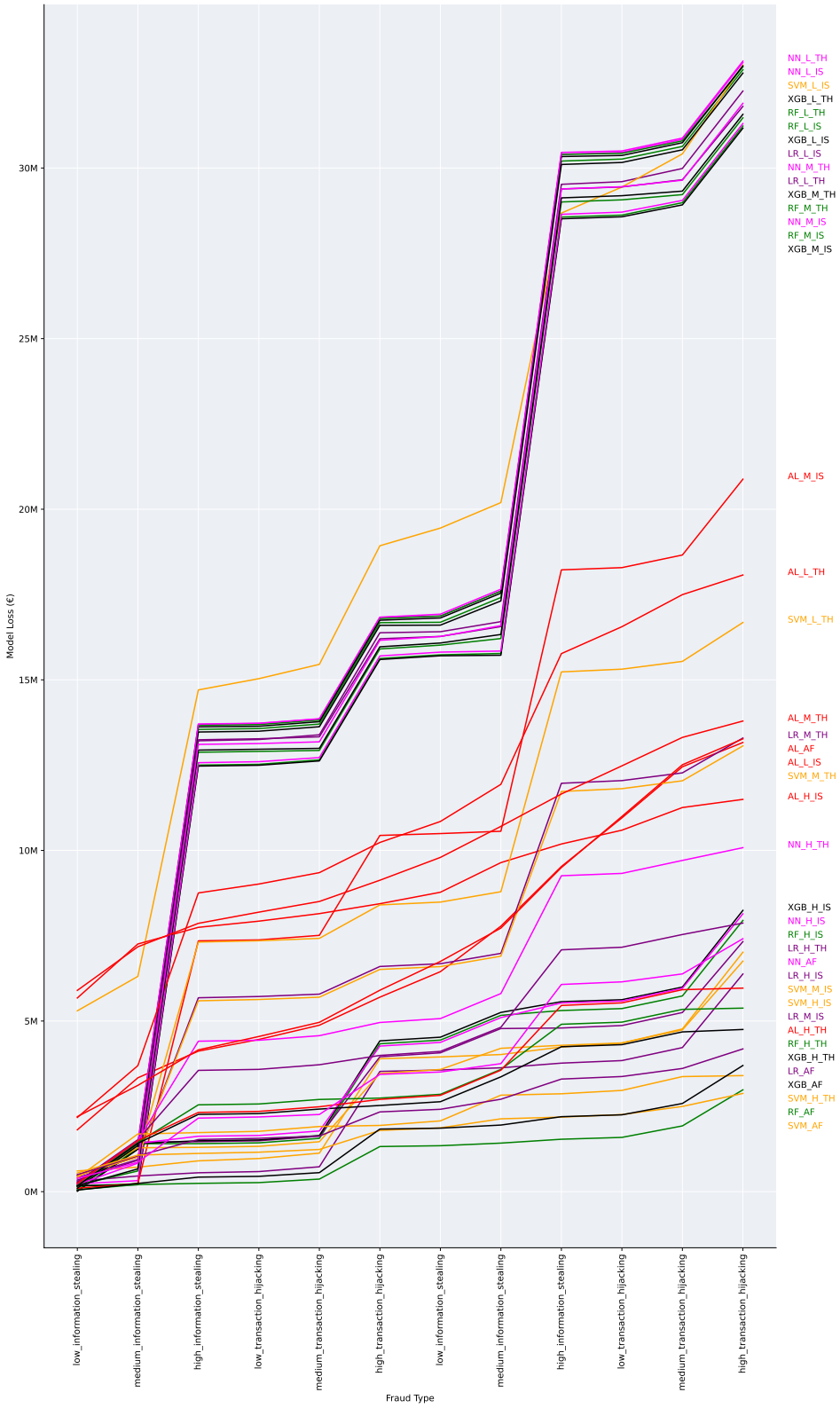


Figure 7.6: Attack 3: models losses (in Millions of Euros) when we cyclically injects fraud types

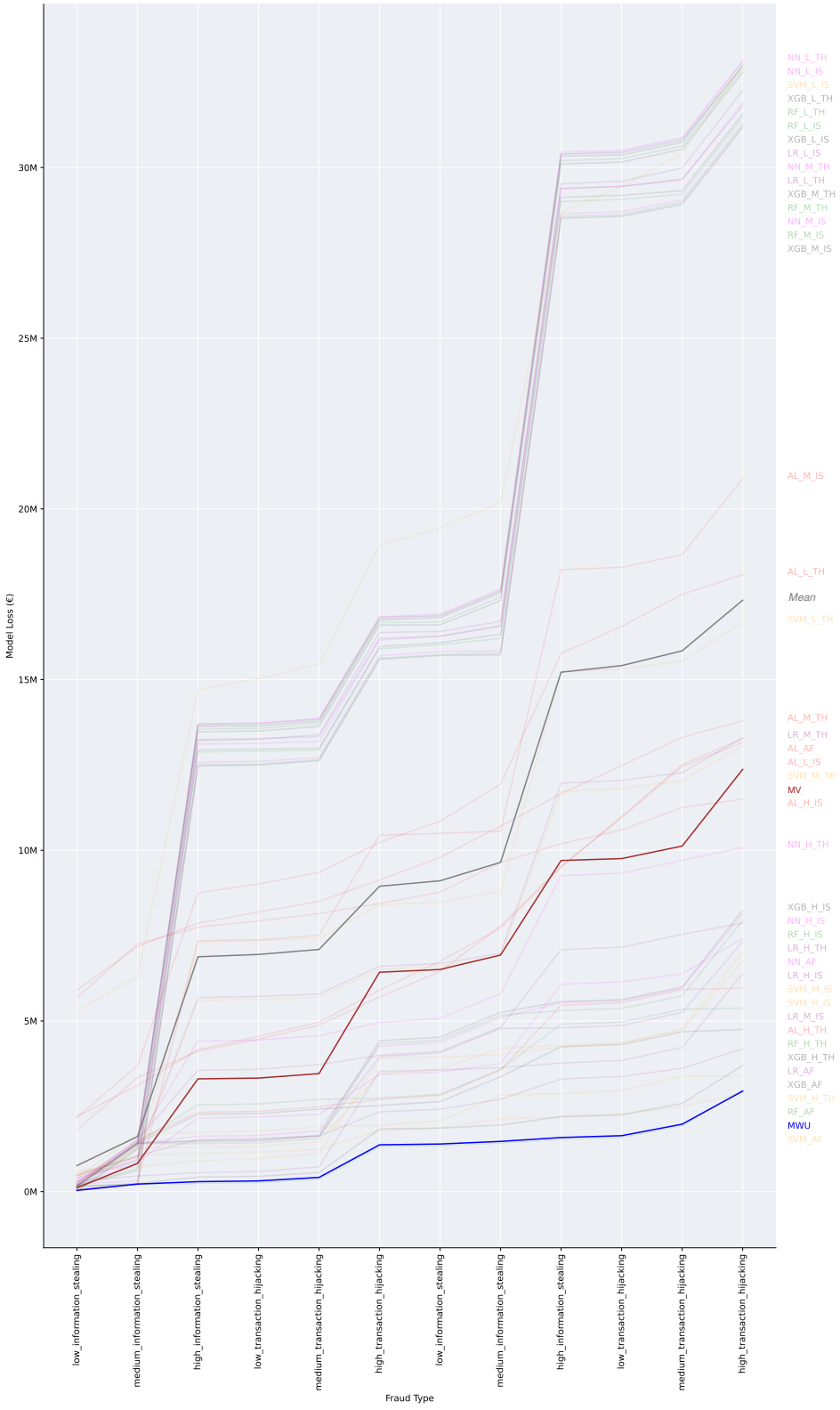


Figure 7.7: Attack 3: ensemble models losses (in Millions of Euros) when we cyclically injects fraud types

Model ID	Precision	Recall	F1-score	FPR	Weighted MCC	Cost Accuracy
AL_AF	0.0143	0.8359	0.0281	0.2562	0.5821	0.7898
AL_H_IS	0.0097	0.4094	0.0190	0.1851	0.2453	0.6121
AL_H_TH	0.2302	0.3244	0.2693	0.0048	0.4309	0.6598
AL_L_IS	0.0130	0.6689	0.0254	0.2266	0.4447	0.7212
AL_L_TH	0.0123	0.4828	0.0239	0.1730	0.3299	0.6549
AL_M_IS	0.4725	0.4914	0.4818	0.0024	0.5670	0.7445
AL_M_TH	0.0103	0.6188	0.0203	0.2642	0.3570	0.6773
LR_AF	0.1478	0.6942	0.2437	0.0178	0.7063	0.8382
LR_H_IS	0.2367	0.3244	0.2737	0.0047	0.4312	0.6599
LR_H_TH	0.1824	0.3101	0.2297	0.0062	0.4165	0.6520
LR_L_IS	0.1239	0.3779	0.1866	0.0119	0.4620	0.6830
LR_L_TH	0.1076	0.3664	0.1664	0.0135	0.4498	0.6764
LR_M_IS	0.2364	0.7323	0.3574	0.0105	0.7469	0.8609
LR_M_TH	0.1270	0.5864	0.2087	0.0179	0.6189	0.7842
NN_AF	0.1619	0.6746	0.2611	0.0155	0.6932	0.8295
NN_H_IS	0.7414	0.2968	0.4239	0.0005	0.4165	0.6481
NN_H_TH	0.2406	0.2739	0.2561	0.0038	0.3904	0.6350
NN_L_IS	0.1494	0.1579	0.1535	0.0040	0.2821	0.5770
NN_L_TH	0.1143	0.0825	0.0958	0.0028	0.1971	0.5398
NN_M_IS	0.2534	0.2700	0.2614	0.0035	0.3878	0.6332
NN_M_TH	0.2476	0.1837	0.2109	0.0025	0.3118	0.5906
RF_AF	0.8165	0.7538	0.7839	0.0008	0.7768	0.8765
RF_H_IS	0.8856	0.2991	0.4472	0.0002	0.4190	0.6495
RF_H_TH	0.4500	0.3306	0.3812	0.0018	0.4417	0.6644
RF_L_IS	0.6098	0.2028	0.3043	0.0006	0.3345	0.6011
RF_L_TH	0.1781	0.1093	0.1354	0.0022	0.2332	0.5535
RF_M_IS	0.8352	0.2514	0.3865	0.0002	0.3787	0.6256
RF_M_TH	0.3939	0.2266	0.2877	0.0016	0.3540	0.6125
SVM_AF	0.1445	0.7142	0.2403	0.0188	0.7216	0.8477
SVM_H_IS	0.2353	0.3573	0.2838	0.0052	0.4571	0.6761
SVM_H_TH	0.1143	0.3612	0.1736	0.0125	0.4473	0.6744
SVM_L_IS	0.0104	0.5711	0.0204	0.2426	0.3344	0.6643
SVM_L_TH	0.1181	0.5520	0.1946	0.0183	0.5910	0.7668
SVM_M_IS	0.1172	0.7891	0.2040	0.0265	0.7760	0.8813
SVM_M_TH	0.1240	0.5921	0.2051	0.0186	0.6226	0.7867
XGB_AF	0.8120	0.7400	0.7743	0.0008	0.7654	0.8696
XGB_H_IS	0.7888	0.2958	0.4303	0.0004	0.4159	0.6477
XGB_H_TH	0.6020	0.3378	0.4328	0.0010	0.4489	0.6684
XGB_L_IS	0.6653	0.2285	0.3402	0.0005	0.3580	0.6140
XGB_L_TH	0.3146	0.1226	0.1765	0.0012	0.2519	0.5607
XGB_M_IS	0.6313	0.2638	0.3721	0.0007	0.3884	0.6316
XGB_M_TH	0.4265	0.2104	0.2818	0.0013	0.3399	0.6046
MV	0.9819	0.4151	0.5835	0.0000	0.5117	0.7075
MWU	0.8390	0.7581	0.7965	0.0006	0.7805	0.8787

Table 7.5: Attack 3: models performance at the end of the 12 weeks

8 | Limitations and Future Works

In this section, we examine the main limitations inherent in our work and subsequently propose potential avenues for future research in this area.

8.1. Limitations

Our framework has the following limitations:

- We designed the aggregation procedure we employ in this study to be adaptable, yet it relies on a minimum set of features (TransactionId, Timestamp, UserID, Amount, IBAN_CC, CC_ASN, Fraud) to effectively operate with different types of datasets. Specific attributes in real-world banking datasets are often not revealed. However, they should comprise fields analogous to those we considered in our research, such as date/time stamps, transaction amounts, IBAN, and user identifiers.
- The fraud generator, despite the possibility of generating fraud following even more complex strategies, is currently implemented based on the features in the dataset used. This adherence to the current dataset makes it unusable with datasets that do not have the same features. Nevertheless, thanks to the framework's modular architecture, it can be easily replaced with another generator that suits the new dataset.
- Our analysis focuses on a dataset comprising transactions from 2014-2015. Over time, customer behaviour can change consistently, potentially affecting the obtained results. It would be valuable to conduct a fresh analysis using a more recent dataset and compare the outcomes to observe the nature and extent of these changes.
- During the detection systems evaluation phase, we do not take any immediate actions, such as account freezing, upon detecting a fraud. This countermeasure may limit the attacker's freedom of action in a real-world scenario.

8.2. Future Works

The field of fraud detection for online banking is constantly evolving, with ongoing research efforts. In light of this, we propose potential extensions to our work:

- Currently, our framework includes six of the most commonly utilised models in this field. As a potential avenue for future development, expanding the repertoire of models within the framework can provide a broader perspective. We can consider the inclusion of models such as Hidden Markov Models [13, 47, 61], Self-Organizing Maps [50], peer group analysis [37, 68], and LSTM (Long Short-Term Memory) [10, 11, 48], further enriching the framework's capabilities and enhancing the overall understanding of the subject matter.
- To enhance the completeness of the framework, we can incorporate additional feature selection and model selection procedures. By integrating new techniques and algorithms for feature selection, the framework can be equipped to handle a broader range of data characteristics and improve the effectiveness of model training. Similarly, incorporating advanced model selection procedures allows for the exploration of different models' hyper-parameters, enabling a more comprehensive analysis potentially leading to more accurate predictions.
- Apart from the ensemble strategies we showcase in this study, numerous other ensemble techniques can be implemented and incorporated to expand our analysis. These techniques include fuzzy rank-based ensemble [40], stacking generalisation [69] and different voting mechanisms [34].
- Another potential development and extension of the framework may involve the integration of modules that allow testing models against adversarial attacks. To the best of our knowledge, there is no framework specifically tailored for conducting targeted adversarial attacks in the field of fraud detection.

9 | Conclusions

In this thesis, we developed FraudBench, a versatile and highly customisable framework to evaluate a range of Fraud Detection Systems based on commonly employed Machine Learning models within the banking and financial industry context. Thanks to its modular design, various components can be easily modified, replaced, or extended to suit different research purposes. Moreover, its customisable nature allows researchers to fine-tune parameters, adjust configurations, and incorporate domain-specific knowledge. We validated our approach through an experimental evaluation conducted on a dataset from an Italian banking group, allowing us to obtain experimental results that demonstrate real-world applicability. Our results showed that a higher model complexity does not always guarantee higher performance. Contrary to expectations, we consistently observed that simpler detection systems based on Support Vector Machines and Logistic Regression can achieve comparable or, sometimes, even better performance compared to more complex models. Secondly, since in the banking and financial context, false positives and false negatives have distinct impacts in terms of monetary loss, commonly used evaluation metrics alone may not provide an accurate assessment of a detection system performance. In our second attack simulation (see Subsection 7.7.2), we demonstrated that the detection system with the best standard performance metrics values yielded a $\sim 40\%$ higher loss with respect to the model that scored the lowest loss. This points out the importance of selecting a model based on its suitability for the specific task at hand rather than relying solely on model complexity and standard evaluation metrics as performance indicators. Our experiments also highlighted the importance of carefully considering the choice of ensemble techniques. Surprisingly, the MV approach did not prove to be an effective alternative in minimising losses against an adaptive fraudulent behaviour. In contrast, a more informed strategy, such as the Multiplicative Weight Update ensemble technique, demonstrated its ability to dynamically adapt and learn from the data, resulting in an superior performance across the three attacks. On average, MWU scored a monetary loss 78% lower than MV and 82% lower than the average individual models loss. The main limitation of our work is that, although we designed our framework to be adaptable, the transaction aggregation and fraud generation procedures rely on a minimal set of features to work properly with

different types of datasets. However, thanks to the modular design, the involved components can be easily adapted or replaced. Future developments include the integration of new Machine Learning models and ensemble strategies, additional feature selection and model selection procedures, and the extension of the framework to allow testing models against adversarial attacks.

Bibliography

[1] Machine learning for fraud detection. URL https://www.ravelin.com/insights/machine-learning-for-fraud-detection.

[2] Payments statistics: 2021, 2022. URL https://www.ecb.europa.eu/press/pr/stats/paysec/html/ecb.pis2021~956efe1ee6.en.html.

[3] Provvedimento banca d'italia del 28 marzo 2022: richiesta di potenziamento delle misure antiriciclaggio, 2022. URL https://n26.com/it-it/provvedimento-banca-ditalia-2022.

[4] Top aml fines in 2022, 2023. URL https://complyadvantage.com/insights/aml-fines-2022/.

[5] M. M. Ahsan, M. A. P. Mahmud, P. K. Saha, K. D. Gupta, and Z. Siddique. Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, 9(3), 2021. ISSN 2227-7080. doi: 10.3390/technologies9030052. URL https://www.mdpi.com/2227-7080/9/3/52.

[6] M. A. Alia, N. Hussinb, and I. A. Abedc. E-banking fraud detection: A short. 2019.

[7] A. Alsayed and A. Bilgrami. E-banking security: Internet hacking, phishing attacks, analysis and prevention of fraudulent activities. *International Journal of Emerging Technology and advanced engineering*, 7(1):109–115, 2017.

[8] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of computing*, 8(1):121–164, 2012.

[9] F. M. Benati. An analysis of defence mechanisms against evasion attacks in the fraud detection domain. Master's thesis, Politecnico di Milano, 2020-21.

[10] I. Benchaji, S. Douzi, and B. Ouahidi. Credit card fraud detection model based on lstm recurrent neural networks. *Journal of Advances in Information Technology*, 12: 113–118, 01 2021. doi: 10.12720/jait.12.2.113-118.

[11] I. Benchaji, S. Douzi, B. Ouahidi, and J. Jaafari. Enhanced credit card fraud detec-

tion based on attention mechanism and lstm deep model. *Journal of Big Data*, 8, 12 2021. doi: 10.1186/s40537-021-00541-8.

[12] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, feb 2012. ISSN 1532-4435.

[13] V. Bhusari and S. Patil. Application of hidden markov model in credit card fraud detection. *International Journal of Distributed and Parallel systems*, 2, 11 2011. doi: 10.5121/ijdps.2011.2618.

[14] A. Bicknell. Cyber security threats to digital banking, 2022. URL `https://www.globalsign.com/en/blog/cyber-security-threats-digital-banking`.

[15] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007. ISBN 0387310738. URL `http://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0387310738`.

[16] J. Bishop. *HISTORY AND PHILOSOPHY OF NEURAL NETWORKS*, pages 22–96. 01 2015. ISBN 978-1780215204.

[17] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 10 2001. doi: 10.1023/A:1010950718922.

[18] J. Brownlee. *Ensemble learning algorithms with Python: Make better predictions with bagging, boosting, and stacking*. Machine Learning Mastery, 2021.

[19] M. Carminati, R. Caron, F. Maggi, I. Epifani, and S. Zanero. Banksealer: A decision support system for online banking fraud analysis and investigation. *Computers & Security*, 53:175–186, 2015. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2015.04.002. URL `https://www.sciencedirect.com/science/article/pii/S0167404815000437`.

[20] M. Carminati, M. Polino, A. Continella, A. Lanzi, F. Maggi, and S. Zanero. Security evaluation of a banking fraud analysis system. *ACM Trans. Priv. Secur.*, 21(3), apr 2018. ISSN 2471-2566. doi: 10.1145/3178370. URL `https://doi.org/10.1145/3178370`.

[21] F. Cartella, O. Anunciacao, Y. Funabiki, D. Yamaguchi, T. Akishita, and O. Elshocht. Adversarial attacks for tabular data: Application to fraud detection and imbalanced data, 2021.

[22] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. pages 785–794, 08 2016. doi: 10.1145/2939672.2939785.

[23] C. Cortes and V. Vapnik. Support-vector networks. *Chem. Biol. Drug Des.*, 297: 273–297, 01 2009. doi: 10.1007/%2FBF00994018.

[24] J. Cramer. The origins of logistic regression. *Tinbergen Institute, Tinbergen Institute Discussion Papers*, 01 2002. doi: 10.2139/ssrn.360300.

[25] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi. Credit card fraud detection and concept-drift adaptation with delayed supervised information. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015. doi: 10.1109/IJCNN.2015.7280527.

[26] M. dell'Economia e delle finanze. Rapporto statistico sulle frodi con le carte di pagamento, 2021. URL `https://www.dt.mef.gov.it/export/sites/sitodt/modules/documenti_it/antifrode_mezzi_pagamento/antifrode_mezzi_pagamento/Rapporto-statistico-sulle-frodi-con-le-carte-di-pagamento-edizione-2021.pdf`.

[27] S. Dhankhad, E. Mohammed, and B. Far. Supervised machine learning algorithms for credit card fraudulent transaction detection: A comparative study. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 122–125, July 2018. doi: 10.1109/IRI.2018.00025.

[28] A. Dignani. Fraudsdigger: an active learning tool for online banking fraud detection. Master's thesis, Politecnico di Milano, 2017-18.

[29] M. N. E-Arefin. A comparative study of machine learning classifiers for credit card fraud detection. *International Journal of Innovative Technology and Interdisciplinary Sciences*, 3(1):395–406, Mar. 2020. URL `https://www.ijitis.org/index.php/ijitis/article/view/46`.

[30] F. Fiordelisi, M.-G. Soana, and P. Schwizer. Reputational losses and operational risk in banking. *The European Journal of Finance*, 20(2):105–124, 2014. doi: 10.1080/1351847X.2012.684218. URL `https://doi.org/10.1080/1351847X.2012.684218`.

[31] J. Fong. Global banking fraud index 2023, 2023. URL `https://seon.io/resources/global-banking-fraud-index/#h-costs-of-fraud`.

[32] K. Fu, D. Cheng, Y. Tu, and L. Zhang. Credit card fraud detection using convolutional neural networks. In A. Hirose, S. Ozawa, K. Doya, K. Ikeda, M. Lee, and

D. Liu, editors, *Neural Information Processing*, pages 483–490, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46675-0.

[33] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2015.

[34] P. Grotti. Novel evasion attacks against banking fraud detection systems. Master's thesis, Politecnico di Milano, 2019-20.

[35] C. Guo, M. Rana, M. Cisse, and L. van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyJ7ClWCb.

[36] S. Karanam. Curse of dimensionality — a "curse" to machine learning, 2021. URL https://towardsdatascience.com/curse-of-dimensionality-a-curse-to-machine-learning-c122ee33bfeb.

[37] Y. Kim and S. Y. Sohn. Stock fraud detection using peer group analysis. *Expert Systems with Applications*, 39(10):8986–8992, 2012. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2012.02.025. URL https://www.sciencedirect.com/science/article/pii/S0957417412002692.

[38] G. K. Kulatilleke. Credit card fraud detection - classifier selection strategy, 2022.

[39] P. Kumar, R. Bhatnagar, K. Gaur, and A. Bhatnagar. Classification of imbalanced data:review of methods and applications. *IOP Conference Series: Materials Science and Engineering*, 1099(1):012077, mar 2021. doi: 10.1088/1757-899X/1099/1/012077. URL https://dx.doi.org/10.1088/1757-899X/1099/1/012077.

[40] R. Kundu, H. Basak, P. K. Singh, A. Ahmadian, M. Ferrara, and R. Sarkar. Fuzzy rank-based fusion of cnn models using gompertz function for screening covid-19 ct-scans. *Scientific reports*, 11(1):14133, 2021.

[41] D. Labanca, L. Primerano, M. Markland-Montgomery, M. Polino, M. Carminati, and S. Zanero. Amaretto: An active learning framework for money laundering detection. *IEEE Access*, 10:41720–41739, 2022. doi: 10.1109/ACCESS.2022.3167699.

[42] D. Liu, L. Wu, H. Zhao, F. Boussaid, M. Bennamoun, and X. Xie. Jacobian norm with selective input gradient regularization for improved and interpretable adversarial defense, 2022.

[43] D. Malekian and M. R. Hashemi. An adaptive profile based fraud detection framework for handling concept drift. In *2013 10th International ISC Confer-*

ence on Information Security and Cryptology (ISCISC), pages 1–6, 2013. doi: 10.1109/ISCISC.2013.6767338.

[44] H. Masnadi-Shirazi and N. Vasconcelos. Asymmetric boosting. In *Proceedings of the 24th international conference on Machine learning*, pages 609–619, 2007.

[45] K. Melcher. A friendly introduction to [deep] neural networks, 2021. URL `https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks`.

[46] F. Monti. Poisoning attacks against banking fraud detection systems. Master's thesis, Politecnico di Milano, 12 2019-20.

[47] M. N. Mr. Abjijeet More, Dnyaneshwari Khane and T. Bhoir. Online transaction fraud detection and prevention using hmm and behavior analysis. *Computer Integrated Manufacturing Systems*, 29(5):266–272, May 2023. URL `http://cims-journal.com/index.php/CN/article/view/897`.

[48] H. Najadat, O. Altiti, A. A. Aqouleh, and M. Younes. Credit card fraud detection based on machine and deep learning. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 204–208, 2020. doi: 10.1109/ICICS49469.2020.239524.

[49] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. M. Molloy, and B. Edwards. Adversarial robustness toolbox v1.0.0, 2019.

[50] D. Olszewski. Fraud detection using self-organizing map visualizing the user profiles. *Knowledge-Based Systems*, 70:324–334, 2014. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2014.07.008. URL `https://www.sciencedirect.com/science/article/pii/S0950705114002652`.

[51] T. Paladini. Rad-x: An adversarial training approach for fraud detection systems. Master's thesis, Politecnico di Milano, 2020-21.

[52] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks, 2016.

[53] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, R. Long, and P. McDaniel. Technical report on the cleverhans v2.1.0 adversarial examples library, 2018.

[54] R. Prati, G. Batista, and M.-C. Monard. Class imbalances versus class overlapping: An analysis of a learning system behavior. pages 312–321, 01 2004. ISBN 978-3-540-21459-5. doi: 10.1007/978-3-540-24694-7_32.

[55] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[56] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling. Deep learning detecting fraud in credit card transactions. In *2018 Systems and Information Engineering Design Symposium (SIEDS)*, pages 129–134, 2018. doi: 10.1109/SIEDS.2018.8374722.

[57] D. Ruta and B. Gabrys. Classifier selection for majority voting. *Information Fusion*, 6:63–81, 03 2005. doi: 10.1016/j.inffus.2004.04.008.

[58] E. Sandberg. The average number of credit card transactions per day and year, 2020. URL `https://www.cardrates.com/advice/number-of-credit-card-transactions-per-day-year/`.

[59] L. Santini. Evasion attacks against banking fraud detection systems. Master's thesis, Politecnico di Milano, 2018-19.

[60] SEON. False positives. URL `https://seon.io/resources/dictionary/false-positives/#h-why-are-false-positives-a-problem`.

[61] A. Srivastava, A. Kundu, S. Sural, and A. Majumdar. Credit card fraud detection using hidden markov model. *IEEE Transactions on Dependable and Secure Computing*, 5(1):37–48, 2008. doi: 10.1109/TDSC.2007.70228.

[62] Stripe. Stripe snapshot - online fraud trends and behavior, 2017. URL `https://b.stripecdn.com/site-srv/assets/files/blog/stripe-snapshot-fraud-5e46ef06938227a9e14d1632e7817b93e849ffc5.pdf`.

[63] SuperAnnotate. Webinar 1 | supercharge your cv pipeline with active learning, 2021. URL `https://www.superannotate.com/blog/supercharge-computer-vision-pipeline-with-active-learning`.

[64] D. Varmedja, M. Karanovic, S. Sladojevic, M. Arsenovic, and A. Anderla. Credit card fraud detection - machine learning methods. In *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–5, 2019. doi: 10.1109/INFOTEH.2019.8717766.

[65] K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li. Ai2: Training a big data machine to defend. In *2016 IEEE 2nd International Conference*

on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), pages 49–54, 2016. doi: 10.1109/BigDataSecurity-HPSC-IDS.2016.79.

[66] A. Ventura. Improving poisoning attacks against banking fraud detection systems. Master's thesis, Politecnico di Milano, 2021-22.

[67] P. Wang, Q. Wang, Y. Zhang, and Y. Wu. Defense mechanism against adversarial attacks based on chaotic map encryption. *Journal of Physics: Conference Series*, 2037(1):012025, sep 2021. doi: 10.1088/1742-6596/2037/1/012025. URL `https://dx.doi.org/10.1088/1742-6596/2037/1/012025`.

[68] D. Weston, D. Hand, N. Adams, C. Whitrow, and P. Juszczak. Plastic card fraud detection using peer group analysis. *Adv. Data Analysis and Classification*, 2:45–62, 04 2008. doi: 10.1007/s11634-008-0021-8.

[69] D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[70] L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2020.07.061. URL `https://www.sciencedirect.com/science/article/pii/S0925231220311693`.

[71] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 856–863, 2003.

A | Appendix A: Adversarial Neural Network Performance Evaluation

A.1. Adversarial Machine Learning

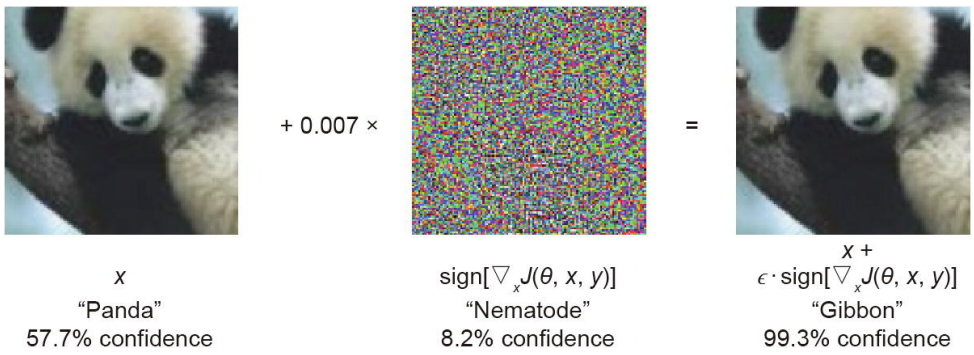


Figure A.1: A demonstration of an adversarial sample generated by applying FGSM to GoogleNet [33]

Adversarial Machine Learning is a field of study that focuses on understanding and defending against adversarial attacks on Machine Learning models. It explores the vulnerabilities of Machine Learning algorithms to intentional manipulation and develops techniques to enhance their robustness. In traditional Machine Learning, we train and test models on the assumption that the data they encounter during deployment comes from the same distribution as the training data. However, adversaries can exploit vulnerabilities in the models by intentionally modifying the input data to deceive or manipulate their behaviour. These modifications are known as adversarial examples. Adversarial examples are carefully crafted inputs that are slightly perturbed from the original data but can cause the Machine Learning model to make incorrect predictions or output misleading results. As Figure A.1 shows, the perturbations are often imperceptible to human observers but can have a significant impact on the model's decision-making process. Adversarial Machine

Learning aims to understand how and why adversarial attacks work, develop techniques to detect and mitigate them and design more robust and resilient Machine Learning models. Researchers in this field explore various defence strategies such as adversarial training, where they train models using both clean and adversarial examples to enhance their ability to handle such attacks. Other techniques include defensive distillation [52], input transformation [35], and regularization [42]. The study of Adversarial Machine Learning is crucial as Machine Learning models are increasingly used in critical applications such as autonomous driving, medical diagnosis, and cybersecurity. By understanding and mitigating adversarial attacks, researchers and practitioners can improve the reliability and security of these models, ensuring they perform as intended even in the presence of malicious actors.

A.2. Application of Adversarial Machine Learning to Fraud Detection

Adversarial examples originally emerged in the context of image recognition, where slight perturbations to input images can deceive Machine Learning models into misclassifying them. However, the concepts and techniques behind adversarial attacks can be extended and applied to other domains, including fraud detection. In contrast to the image classification domain, the banking fraud detection domain primarily deals with data samples that have a limited number of features. This unique characteristic poses challenges for applying conventional approaches. However, in recent years, researchers have introduced novel methodologies to carry out impactful adversarial attacks [34, 46, 59, 66], emphasizing the significance of addressing this problem. Alongside the exploration of novel methods to evade detection systems, an extensive research has been conducted to develop effective defence mechanisms against adversarial attacks [9, 28, 51]. To provide readers with an understanding of how a model equipped with a defence mechanism performs in a context similar to the one we tested our standard models on, we conducted the same experiments described in Section 7.7 employing a Neural Network that incorporates adversarial regularization. Adversarial training was initially introduced as adversarial regularization by Goodfellow et al. [33]. It is a technique aimed at fortifying a model against adversarial examples. The core idea of adversarial training is to embed the same procedure used to generate such examples into the training algorithm to train the model to identify adversarial examples. This is accomplished by incorporating a regularization term into the loss function of the Machine Learning model, as proposed by the authors.

A.3. Experimental Results

It is essential to recall that the experiments we conduct do not primarily focus on Adversarial Machine Learning. Consequently, the purpose of this comparison is not to evaluate the effectiveness of the implemented defence mechanism, as we are not specifically simulating an adversarial attack scenario. Rather, the primary objective is to assess the behaviour of a model equipped with a defence mechanism within each experimental context.

Figures A.2, A.3, and A.4 present the performance of the neural network with adversarial regularization in experiments 1, 2, and 3, respectively. Across all three experiments, the most effective FDS based on the Adversarial Neural Network is the one trained on all types of fraud, followed by FDSs trained on High-profile fraud cases. Conversely, the detection systems trained on Medium and Low-profile frauds, unsurprisingly, are the least successful ones.

In experiment 1, the adversarial neural network leading detection system (`ADV_NN_AF`) performs slightly worse (+0.38 %) than the corresponding version of standard Neural Network (`NN_AF`), reaching a loss of 2,174,298.89 €. Regarding the systems trained on specific types of frauds, apart from `ADV_NN_L_TH` and `ADV_NN_M_TH`, the performances are better compared to the version that does not incorporate the defence mechanism. The highest margin is reached by `ADV_NN_H_TH` that has been able to reduce the losses by 7,742,209.23 € (-71.23 %). It is also worth noting that, in the worst-case scenario, the increase in loss does not exceed 0.76 %.

In experiment 2, not only the `ADV_NN_AF` significantly outperforms its standard version by a wide margin (-67.68 %), but it also emerges as the overall best-performing model in the long run. With a final loss of 3,611,186.67 € `ADV_NN_AF` surpasses the previous top-performing model (`SVM_H_TH`) by a margin of 256,901.8 €. However, this comes at a cost. Although the percentage of detected fraud is the highest among all detection systems (94.8 %), the precision of `ADV_NN_AF` is very low (10 %) and, consequently, the value of false positives higher ($\sim$ 3x) compared to the best SVM and the standard NN corresponding version.

Experiment 3 yields a similar result, as `ADV_NN_AF` showcases significantly better performance than `NN_AF` (achieving a remarkable reduction in loss of 63.12 %) and also performs marginally better (by approximately 5 %) than the previous top-performing model which is the `SVM_AF`.

As Table A.1, A.1 and A.1 show, the detection system based on Adversarial Neural Network, in most of the cases demonstrates superior performance. However, there are a few instances where this is not true and these exceptions emphasize the fact that the integra-

tion of a defense mechanism does not always guarantee an improvement in the model's performance.

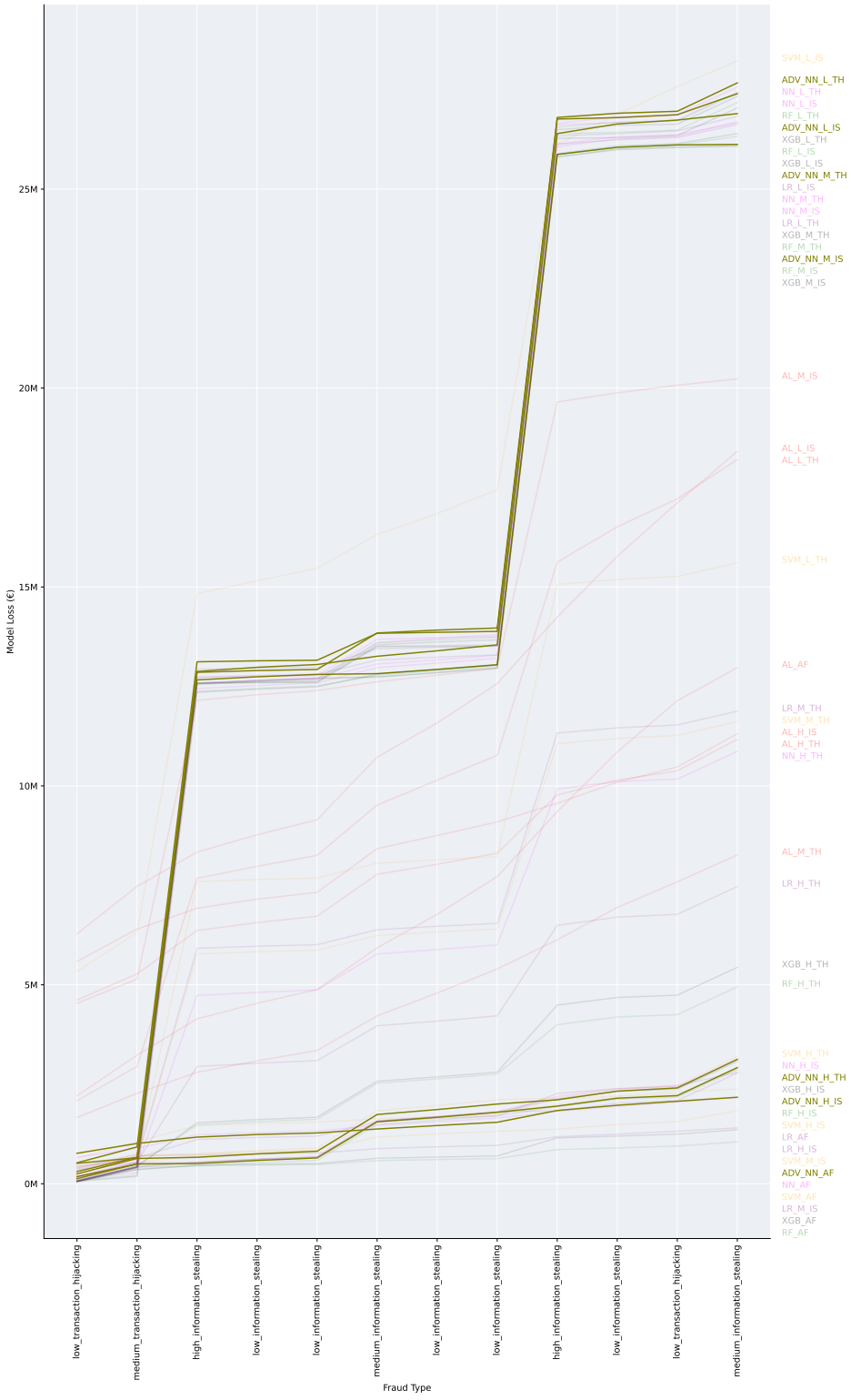


Figure A.2: Attack 1-bis: adversarial neural network models losses (in Millions of Euros) injecting fraud types following a random policy

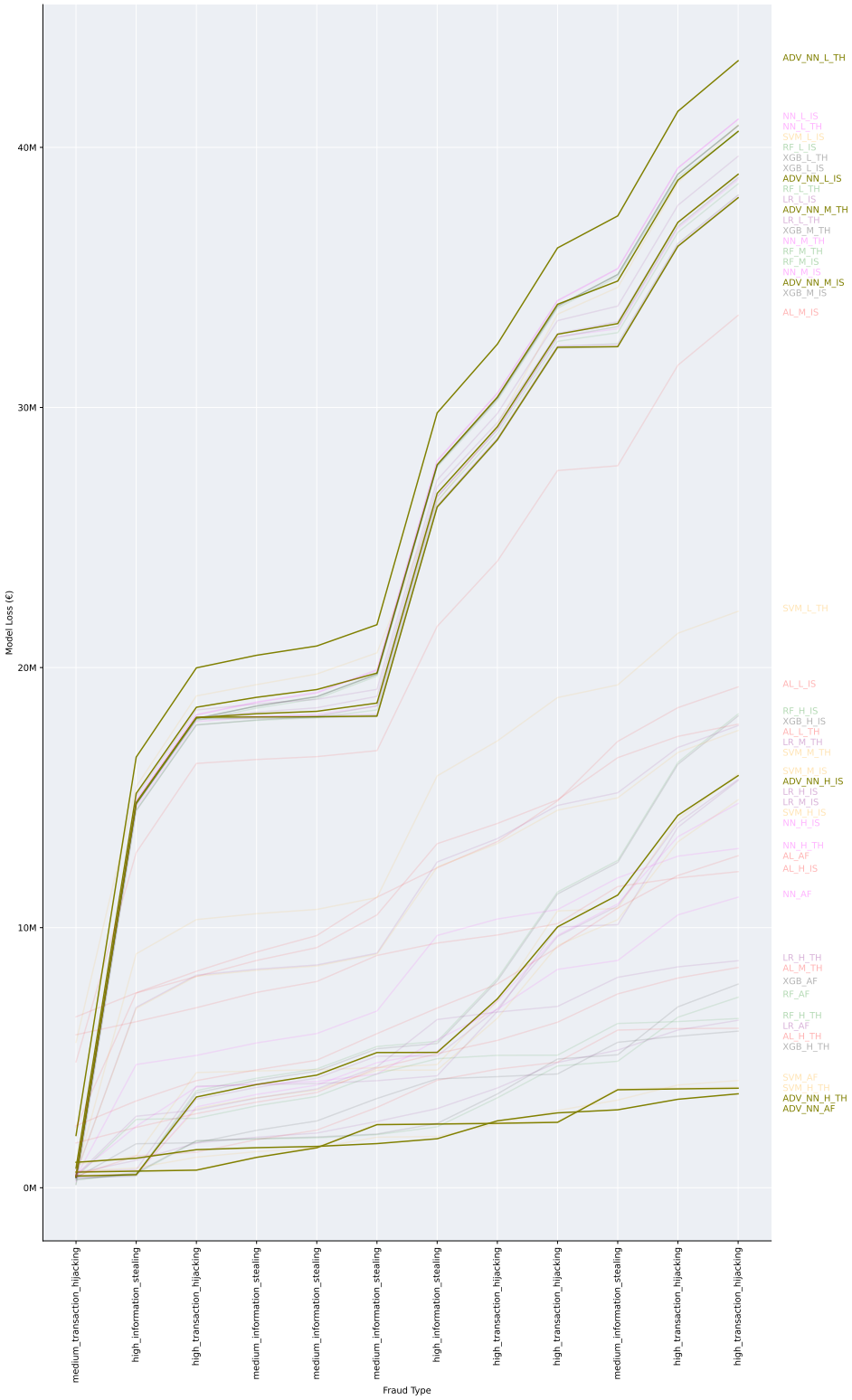


Figure A.3: Attack 2-bis: adversarial neural network models losses (in Million of Euros) injecting fraud types with highest impact on previous week's best model

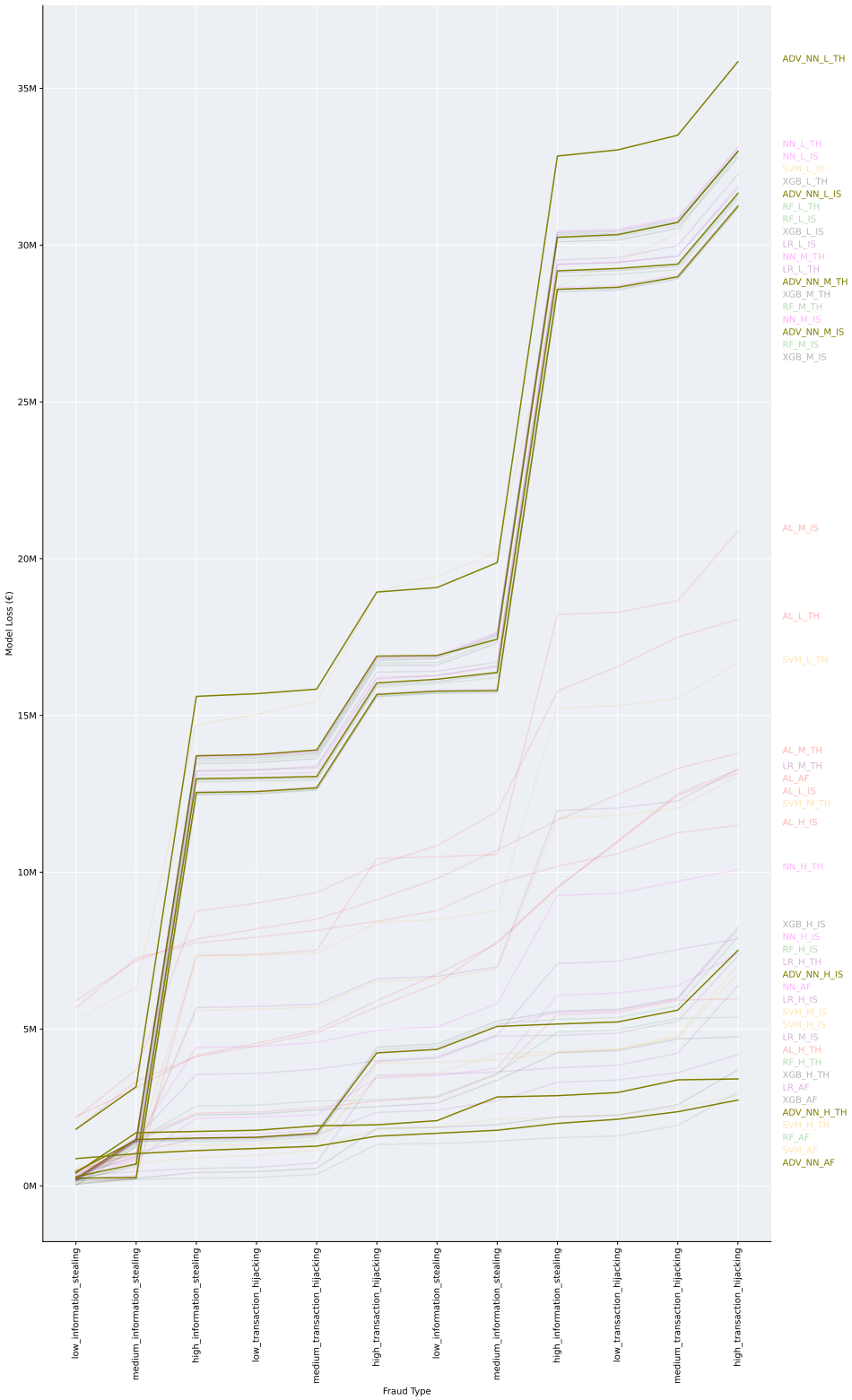


Figure A.4: Attack 3-bis: adversarial neural network models losses (in Millions of Euros) when we cyclically injects fraud types

Fraud type ID	Adversarial NN	Neural Network	Loss Gap
AF	2,174,298.97 €	2,166,160.84 €	+0.38 %
H_IS	2,919,045.4 €	3,137,902.43 €	-6.97 %
H_TH	3,126,027.06 €	10,868,236.29 €	-71.23 %
M_IS	26,125,037.99 €	26,664,110.93 €	-2.02 %
M_TH	26,897,671.83 €	26,695,408.53 €	+0.76 %
L_IS	27,399,166.75 €	27,440,748.01 €	-0.15 %
L_TH	27,670,569.68 €	27,566,497.84 €	+0.38 %

Table A.1: Loss difference, in experiment 1, between adversarial neural network and standard neural network-based detection systems.

Fraud type ID	Adversarial NN	Neural Network	Loss Gap
AF	3,611,186.67 €	11,172,286.58 €	-67.68 %
H_IS	15,842,722.55 €	14,773,603.26 €	+7.23 %
H_TH	3,825,588.47 €	13,041,777.59 €	-70.67 %
M_IS	38,064,166.21 €	38,149,863.82 €	-0.22 %
M_TH	38,966,132.2 €	38,765,759.94 €	+0.52 %
L_IS	40,615,757.58 €	41,085,242.9 €	-1.14 %
L_TH	43,330,437.1 €	41,079,875.87 €	+5.47 %

Table A.2: Loss difference, in experiment 2, between adversarial neural network and standard neural network-based detection systems.

Model ID	Adversarial NN	Neural Network	Loss Gap
all_frauds_type	2,733,768.13 €	7,412,079.79 €	-63.12 %
H_IS	7,504,621.31 €	8,142,174.76 €	-7.83 %
H_TH	3,407,964.02 €	10,080,809.46 €	-66.19 %
M_IS	31,237,680.54 €	31,303,369.59 €	-0.21 %
M_TH	31,650,103.4 €	31,895,974.3 €	-0.77 %
L_IS	32,989,662.43 €	33,092,066.47 €	-0.31 %
L_TH	35,850,067.24 €	33,132,209.12 €	+8.2 %

Table A.3: Loss difference, in experiment 3, between adversarial neural network and standard neural network-based detection systems.

B | Appendix B: Support Tables

In this chapter, we provide some tables containing the outcomes of the feature filtering process detailed in Section 7.5 and the model selection outlined in Section 7.6.

Table B.1: Features filtering results

Dataset ID	Selected Features
augmented_HIS	Amount, Fraud, Time_x, Time_y, amount_mean14d, amount_mean7d, amount_std14d, amount_std1d, amount_std1h, amount_std30d, amount_std7d, amount_sum14d, amount_sum1d, amount_sum30d, amount_sum7d, cc_asn_count14d, cc_asn_count1d, cc_asn_count1h, cc_asn_count30d, cc_asn_count7d, cc_asn_mean1d, cc_asn_mean30d, difference_from_amount_mean1d, difference_from_amount_mean1h, difference_from_amount_mean30d, difference_from_amount_mean7d, difference_from_cc_asn_mean14d, difference_from_cc_asn_mean30d, difference_from_cc_asn_mean7d, difference_from_iban_cc_mean14d, difference_from_iban_cc_mean1d, difference_from_iban_cc_mean30d, difference_from_iban_cc_mean7d, difference_from_iban_mean14d, difference_from_iban_mean1d, difference_from_iban_mean1h, difference_from_iban_mean30d, difference_from_iban_mean7d, difference_from_ip_mean14d, difference_from_ip_mean1d, difference_from_ip_mean7d, difference_from_session_mean1d, iban_cc_mean30d, iban_count14d, iban_count1d, iban_count1h, iban_count30d, iban_count7d, iban_mean30d, iban_std1d, iban_sum14d, iban_sum1d, iban_sum1h, iban_sum30d, iban_sum7d, ip_count14d, ip_count30d, ip_mean7d, ip_std14d, ip_std1d, ip_std30d, ip_std7d, ip_sum14d, ip_sum30d, is_international, is_national_iban, is_new_cc_asn, is_new_iban, is_new_iban_cc, is_new_ip, session_count14d, session_sum1h, time_since_same_iban, time_since_same_iban_cc, time_since_same_session
augmented_HTH	Amount, Fraud, Time_x, Time_y, amount_count7d, amount_mean14d, amount_std30d, amount_sum30d, cc_asn_mean30d, difference_from_amount_mean30d, difference_from_amount_mean7d, difference_from_cc_asn_mean14d, difference_from_iban_cc_mean14d, difference_from_iban_cc_mean1d, difference_from_iban_cc_mean1h, difference_from_iban_cc_mean30d, difference_from_iban_cc_mean7d, difference_from_iban_mean14d, difference_from_iban_mean1h, difference_from_iban_mean30d, difference_from_iban_mean7d, difference_from_ip_mean14d, difference_from_ip_mean1d, difference_from_ip_mean1h, difference_from_ip_mean7d, difference_from_session_mean1h, iban_cc_count14d, iban_cc_count30d, iban_cc_mean1d, iban_cc_mean30d, iban_cc_sum14d, iban_count14d, iban_count1d, iban_count1h, iban_count30d, iban_count7d, iban_mean30d, iban_std1d, iban_std7d, iban_sum14d, iban_sum1d, iban_sum30d, iban_sum7d, ip_count14d, ip_count1d, ip_count30d, ip_count7d, ip_mean30d, ip_mean7d, ip_std14d, ip_std1d, ip_std30d, ip_std7d, ip_sum14d, ip_sum1d, ip_sum30d, ip_sum7d, is_international, is_national_iban, is_new_cc_asn, is_new_iban, is_new_iban_cc, is_new_ip, session_count1h, session_mean14d, session_std1h, session_sum1h, time_from_previous_trans_global, time_since_same_iban

Continued on next page

Table B.1 – Continued from previous page

Dataset ID	Selected Features
augmented_MIS	Amount, Fraud, Time_x, Time_y, amount_sum1d, cc_asn_count14d, cc_asn_count1d, cc_asn_count1h, cc_asn_count30d, cc_asn_count7d, cc_asn_mean1d, cc_asn_mean30d, cc_asn_mean7d, cc_asn_sum14d, cc_asn_sum1h, cc_asn_sum30d, cc_asn_sum7d, difference_from_amount_mean14d, difference_from_amount_mean1d, difference_from_amount_mean1h, difference_from_amount_mean30d, difference_from_amount_mean7d, difference_from_iban_mean14d, difference_from_iban_mean1d, difference_from_iban_mean1h, difference_from_iban_mean30d, difference_from_iban_mean7d, difference_from_ip_mean14d, difference_from_ip_mean7d, difference_from_session_mean1d, iban_cc_mean14d, iban_cc_std30d, iban_count14d, iban_count1d, iban_count1h, iban_count30d, iban_count7d, iban_mean30d, iban_std1d, iban_std7d, iban_sum14d, iban_sum1d, iban_sum30d, iban_sum7d, ip_count14d, ip_count30d, ip_mean30d, ip_std14d, ip_std1d, ip_std1h, ip_std30d, ip_std7d, ip_sum30d, is_international, is_national_iban, is_new_cc_asn, is_new_iban, is_new_iban_cc, is_new_ip, session_count14d, session_sum7d, time_since_same_cc_asn, time_since_same_iban, time_since_same_session
augmented_MTH	Amount, Fraud, Time_x, Time_y, amount_count7d, cc_asn_std14d, cc_asn_std30d, cc_asn_std7d, cc_asn_sum14d, cc_asn_sum30d, difference_from_amount_mean14d, difference_from_amount_mean30d, difference_from_amount_mean7d, difference_from_iban_mean14d, difference_from_iban_mean1d, difference_from_iban_mean1h, difference_from_iban_mean30d, difference_from_iban_mean7d, difference_from_ip_mean14d, difference_from_ip_mean1d, difference_from_ip_mean1h, difference_from_ip_mean7d, difference_from_session_mean1h, iban_cc_count14d, iban_cc_count30d, iban_cc_mean1d, iban_cc_mean30d, iban_cc_mean7d, iban_count14d, iban_count1d, iban_count1h, iban_count30d, iban_count7d, iban_mean30d, iban_std1d, iban_std7d, iban_sum14d, iban_sum1d, iban_sum30d, iban_sum7d, ip_count14d, ip_count1d, ip_count30d, ip_count7d, ip_mean30d, ip_std1d, ip_sum1d, ip_sum1h, ip_sum30d, ip_sum7d, is_international, is_national_iban, is_new_cc_asn, is_new_iban, is_new_iban_cc, is_new_ip, session_count1h, session_std1h, session_sum1h, time_from_previous_trans_global, time_since_same_iban
augmented_LIS	Amount, Fraud, Time_x, Time_y, amount_std1h, cc_asn_count14d, cc_asn_count1d, cc_asn_count1h, cc_asn_count30d, cc_asn_count7d, cc_asn_mean1d, cc_asn_mean30d, cc_asn_mean7d, cc_asn_sum14d, cc_asn_sum1d, cc_asn_sum1h, cc_asn_sum30d, cc_asn_sum7d, difference_from_amount_mean14d, difference_from_amount_mean1d, difference_from_amount_mean30d, difference_from_amount_mean7d, difference_from_cc_asn_mean1h, difference_from_iban_mean14d, difference_from_iban_mean1d, difference_from_iban_mean1h, difference_from_iban_mean30d, difference_from_iban_mean7d, difference_from_ip_mean14d, difference_from_ip_mean7d, difference_from_session_mean30d, iban_cc_mean14d, iban_cc_std30d, iban_count14d, iban_count1d, iban_count1h, iban_count30d, iban_count7d, iban_mean30d, iban_std1d, iban_std7d, iban_sum14d, iban_sum1h, iban_sum30d, iban_sum7d, ip_count14d, ip_count30d, ip_mean30d, ip_std14d, ip_std1d, ip_std30d, ip_std7d, ip_sum30d, is_international, is_national_iban, is_new_cc_asn, is_new_iban, is_new_iban_cc, is_new_ip, session_count14d, session_sum1d, time_since_same_cc_asn, time_since_same_iban, time_since_same_session
augmented_LTH	Amount, Fraud, Time_x, Time_y, difference_from_cc_asn_mean14d, difference_from_cc_asn_mean1d, difference_from_cc_asn_mean30d, difference_from_cc_asn_mean7d, difference_from_iban_mean14d, difference_from_iban_mean1d, difference_from_iban_mean1h, difference_from_iban_mean30d, difference_from_iban_mean7d, difference_from_ip_mean14d, difference_from_ip_mean1h, difference_from_ip_mean7d, difference_from_session_mean1h, iban_cc_count14d, iban_cc_count30d, iban_cc_count7d, iban_cc_mean1d, iban_cc_mean30d, iban_cc_mean7d, iban_cc_std14d, iban_cc_std1d, iban_cc_std30d, iban_cc_std7d, iban_cc_sum14d, iban_cc_sum1d, iban_cc_sum1h, iban_cc_sum30d, iban_cc_sum7d, iban_count14d, iban_count1d, iban_count1h, iban_count30d, iban_count7d, iban_mean30d, iban_std1d, iban_std7d, iban_sum14d, iban_sum1h, iban_sum30d, iban_sum7d, ip_count14d, ip_count1d, ip_count30d, ip_count7d, ip_mean30d, ip_std14d, ip_std30d, ip_sum14d, ip_sum30d, is_international, is_national_iban, is_new_cc_asn, is_new_iban, is_new_iban_cc, is_new_ip, session_count1h, session_std1h, session_sum1h, time_from_previous_trans_global, time_since_same_iban

Continued on next page

Table B.1 – Continued from previous page

Dataset ID	Selected Features
augmented_AF	Amount, Fraud, Time_x, Time_y, amount_mean30d, amount_std14d, amount_std1d, amount_std1h, amount_std30d, amount_std7d, amount_sum14d, amount_sum1d, amount_sum30d, amount_sum7d, cc_asn_count14d, cc_asn_count1d, cc_asn_count30d, cc_asn_count7d, cc_asn_mean1d, cc_asn_mean30d, cc_asn_mean7d, difference_from_amount_mean14d, difference_from_amount_mean1d, difference_from_amount_mean1h, difference_from_amount_mean30d, difference_from_amount_mean7d, difference_from_cc_asn_mean14d, difference_from_cc_asn_mean30d, difference_from_cc_asn_mean7d, difference_from_iban_cc_mean14d, difference_from_iban_cc_mean1d, difference_from_iban_cc_mean30d, difference_from_iban_cc_mean7d, difference_from_iban_mean14d, difference_from_iban_mean1d, difference_from_iban_mean1h, difference_from_iban_mean30d, difference_from_iban_mean7d, difference_from_ip_mean14d, difference_from_ip_mean1d, difference_from_ip_mean7d, difference_from_session_mean30d, iban_cc_count1h, iban_cc_mean14d, iban_count14d, iban_count1d, iban_count1h, iban_count30d, iban_count7d, iban_mean30d, iban_std1d, iban_std30d, iban_sum14d, iban_sum1d, iban_sum1h, iban_sum30d, iban_sum7d, ip_count14d, ip_count30d, ip_mean14d, ip_std14d, ip_std30d, ip_std7d, ip_sum14d, ip_sum30d, is_international, is_national_iban, is_new_cc_asn, is_new_iban, is_new_iban_cc, is_new_ip, session_count30d, session_sum1h, time_since_same_cc_asn, time_since_same_iban, time_since_same_session

Table B.2: Hyperparameters for models trained on all types of fraudulent transactions

Model	Hyperparameters	Value
Active Learning	ae_epochs	60
	ae_batch_size	32
	ae_encoding_size	296
	ae_bottleneck_size	37
	ae_dropout_rate	0.6489810128741015
	ae_output_activation	sigmoid
	ae_lambda_reg	0.00010074478798116343
	ae_activation	relu
	ae_weight	0.2
	rf_random_state	721077
	rf_n_estimators	148
	rf_max_depth	178
	rf_criterion	gini
	rf_class_weight	balanced
	rf_min_samples_split	2
	rf_weight	0.8
Logistic Regression	random_state	721077
	penalty	l2
	C	102.98823881396297
	class_weight	balanced
	tol	0.9396885694252239
	solver	newton-cholesky
	l1_ratio	None
	max_iter	8000
Neural Network	epochs	20
	batch_size	2048
	fl_neurons	73
	sl_neurons	157
	activation_funct	relu
	dropout_rate	0.5253662735613859
Random Forest	random_state	721077
	n_estimators	361
	max_depth	57
	max_features	sqrt
	class_weight	balanced
	criterion	entropy
	min_samples_split	3
	min_samples_leaf	3

Continued on next page

Table B.2 – Continued from previous page

Model	Hyperparameters	Value
Support Vector Machine	random_state	721077
	penalty	l1
	loss	squared_hinge
	dual	False
	tol	0.35167343607943163
	class_weight	balanced
	C	146.007103517384
	max_iter	10000
XGBoost	random_state	721077
	max_depth	37
	learning_rate	0.4602119738318374
	booster	gbtree
	gamma	0.02625618289109055
	n_estimators	139
	scale_pos_weight	102.75506072874494

Table B.3: Hyperparameters for models trained on fraudulent transactions characterized by High profile and Information Stealing scheme

Model	Hyperparameters	Value
Active Learning	ae_epochs	20
	ae_batch_size	1024
	ae_encoding_size	32
	ae_bottleneck_size	15
	ae_dropout_rate	0.2566292114877229
	ae_output_activation	sigmoid
	ae_lambda_reg	0.00025
	ae_activation	relu
	ae_weight	0.2
	rf_random_state	721077
	rf_n_estimators	153
	rf_max_depth	27
	rf_criterion	gini
	rf_class_weight	balanced
	rf_min_samples_split	2
	rf_weight	0.8

Continued on next page

Table B.3 – Continued from previous page

Model	Hyperparameters	Value
Logistic Regression	random_state	721077
	penalty	l1
	C	13.991915778357995
	class_weight	balanced
	tol	0.003558411769099488
	solver	saga
	max_iter	2000
Neural Network	epochs	100
	batch_size	1024
	fl_neurons	53
	sl_neurons	112
	activation_funct	relu
	dropout_rate	0.5646034511403906
Random Forest	random_state	721077
	n_estimators	167
	max_depth	16
	class_weight	balanced
	criterion	entropy
	min_samples_split	3
	min_samples_leaf	2
Support Vector Machine	random_state	721077
	penalty	l2
	loss	squared_hinge
	dual	False
	class_weight	balanced
	C	15.153576957788962
	max_iter	10000
XGBoost	random_state	721077
	max_depth	14
	learning_rate	0.1670259859121837
	booster	gbtree
	gamma	0.872235009618874
	n_estimators	65
	scale_pos_weight	101.676

Table B.4: Hyperparameters for models trained on fraudulent transactions characterized by High profile and Transaction Hijacking scheme

Model	Hyperparameters	Value
Active Learning	ae_epochs	20
	ae_batch_size	64
	ae_encoding_size	188
	ae_bottleneck_size	163
	ae_dropout_rate	0.5980383087152149
	ae_output_activation	sigmoid
	ae_lambda_reg	0.0002494876101233998
	ae_activation	relu
	ae_weight	0.2
	rf_random_state	721077
	rf_n_estimators	746
	rf_max_depth	20
	rf_criterion	entropy
	rf_class_weight	balanced
	rf_min_samples_split	3
	rf_weight	0.8
Logistic Regression	random_state	721077
	penalty	l2
	C	51.43355647433214
	class_weight	balanced
	tol	0.875199888324913
	solver	newton-cholesky
	max_iter	10000
Neural Network	epochs	100
	batch_size	16
	fl_neurons	137
	sl_neurons	97
	activation_funct	tanh
	dropout_rate	0.28739790199418624
Random Forest	random_state	721077
	n_estimators	535
	max_depth	22
	max_features	sqrt
	class_weight	balanced
	criterion	gini
	min_samples_split	5
	min_samples_leaf	1

Continued on next page

Table B.4 – Continued from previous page

Model	Hyperparameters	Value
Support Vector Machine	random_state	721077
	penalty	l1
	loss	squared_hinge
	dual	False
	tol	0.7400423006314985
	class_weight	balanced
	C	105.2165127713016
	max_iter	2000
XGBoost	random_state	721077
	max_depth	18
	learning_rate	0.1399184624440806
	booster	gbtree
	gamma	0.521099695780107
	n_estimators	147
	scale_pos_weight	106.92584745762711

Table B.5: Hyperparameters for models trained on fraudulent transactions characterized by Medium profile and Information Stealing scheme

Model	Hyperparameters	Value
Active Learning	ae_epochs	20
	ae_batch_size	512
	ae_encoding_size	32
	ae_bottleneck_size	15
	ae_dropout_rate	0.07988442863966094
	ae_output_activation	sigmoid
	ae_lambda_reg	0.00025
	ae_activation	relu
	ae_weight	0.2
	rf_random_state	721077
	rf_n_estimators	188
	rf_max_depth	29
	rf_criterion	gini
	rf_class_weight	balanced
	rf_min_samples_split	2
	rf_weight	0.8

Continued on next page

Table B.5 – Continued from previous page

Model	Hyperparameters	Value
Logistic Regression	random_state	721077
	penalty	l1
	C	72.7326645818656
	class_weight	balanced
	tol	6.394901244509599e-05
	solver	saga
	max_iter	2000
Neural Network	epochs	40
	batch_size	2048
	fl_neurons	105
	sl_neurons	138
	activation_funct	relu
	dropout_rate	0.06070387763473942
Random Forest	random_state	721077
	n_estimators	419
	max_depth	30
	class_weight	balanced
	criterion	entropy
	min_samples_split	5
	min_samples_leaf	1
Support Vector Machine	random_state	721077
	penalty	l2
	loss	squared_hinge
	dual	False
	class_weight	balanced
	C	17.509600061409767
	max_iter	2000
XGBoost	random_state	721077
	max_depth	3
	learning_rate	0.4133885597326414
	booster	gbtree
	gamma	0.11324944314460808
	n_estimators	208
	scale_pos_weight	101.47105788423154

Table B.6: Hyperparameters for models trained on fraudulent transactions characterized by Medium profile and Transaction Hijacking scheme

Model	Hyperparameters	Value
Active Learning	ae_epochs	40
	ae_batch_size	2048
	ae_encoding_size	84
	ae_bottleneck_size	62
	ae_dropout_rate	0.04381970146136949
	ae_output_activation	sigmoid
	ae_lambda_reg	0.0002819049114776747
	ae_activation	relu
	ae_weight	0.2
	rf_random_state	721077
	rf_n_estimators	658
	rf_max_depth	96
	rf_criterion	gini
	rf_class_weight	balanced
	rf_min_samples_split	2
	rf_weight	0.8
Logistic Regression	random_state	721077
	penalty	l1
	C	85.00950251543314
	class_weight	balanced
	tol	0.8736750777153988
	solver	saga
	max_iter	2000
Neural Network	epochs	40
	batch_size	1024
	fl_neurons	63
	sl_neurons	62
	activation_funct	relu
	dropout_rate	0.6842075366156709
Random Forest	random_state	721077
	n_estimators	299
	max_depth	96
	max_features	None
	class_weight	balanced
	criterion	entropy
	min_samples_split	7
	min_samples_leaf	4

Continued on next page

Table B.6 – Continued from previous page

Model	Hyperparameters	Value
Support Vector Machine	random_state	721077
	penalty	l2
	loss	squared_hinge
	dual	False
	tol	0.7238909625636086
	class_weight	balanced
	C	87.53495774656727
	max_iter	4000
XGBoost	random_state	721077
	max_depth	38
	learning_rate	0.49593870881234225
	booster	gbtree
	gamma	0.24650779542991613
	n_estimators	307
	scale_pos_weight	99.29133858267717

Table B.7: Hyperparameters for models trained on fraudulent transactions characterized by Low profile and Information Stealing scheme

Model	Hyperparameters	Value
Active Learning	ae_epochs	20
	ae_batch_size	1024
	ae_encoding_size	66
	ae_bottleneck_size	171
	ae_dropout_rate	0.48650864873603566
	ae_output_activation	sigmoid
	ae_lambda_reg	0.0002609748709776789
	ae_activation	relu
	ae_weight	0.2
	rf_random_state	721077
	rf_n_estimators	661
	rf_max_depth	32
	rf_criterion	gini
	rf_class_weight	balanced
	rf_min_samples_split	2
	rf_weight	0.8

Continued on next page

Table B.7 – Continued from previous page

Model	Hyperparameters	Value
Logistic Regression	random_state	721077
	penalty	l2
	C	61.96308798143124
	class_weight	balanced
	tol	0.00019286809105870724
	solver	saga
	max_iter	2000
Neural Network	epochs	40
	batch_size	256
	fl_neurons	83
	sl_neurons	107
	activation_funct	relu
	dropout_rate	0.001663151923775885
Random Forest	random_state	721077
	n_estimators	259
	max_depth	55
	max_features	sqrt
	class_weight	balanced
	criterion	entropy
	min_samples_split	9
	min_samples_leaf	1
Support Vector Machine	random_state	721077
	penalty	l1
	loss	squared_hinge
	dual	False
	tol	0.5823182681658402
	class_weight	balanced
	C	25.404624354640724
	max_iter	8000
XGBoost	random_state	721077
	max_depth	7
	learning_rate	0.23536024437644162
	booster	gbtree
	gamma	0.41776215860889465
	n_estimators	308
	scale_pos_weight	101.0596421471173

Table B.8: Hyperparameters for models trained on fraudulent transactions characterized by Low profile and Transaction Hijacking scheme

Model	Hyperparameters	Value
Active Learning	ae_epochs	60
	ae_batch_size	4096
	ae_encoding_size	143
	ae_bottleneck_size	79
	ae_dropout_rate	0.48544289436896004
	ae_output_activation	sigmoid
	ae_lambda_reg	0.000352019707688131
	ae_activation	relu
	ae_weight	0.2
	rf_random_state	721077
	rf_n_estimators	370
	rf_max_depth	42
	rf_criterion	entropy
	rf_class_weight	balanced
	rf_min_samples_split	2
	rf_weight	0.8
Logistic Regression	random_state	721077
	penalty	elasticnet
	C	77.31916559691692
	class_weight	balanced
	tol	0.22511130670513804
	solver	saga
	l1_ratio	0.43341508347790103
	max_iter	4000
Neural Network	epochs	80
	batch_size	4096
	fl_neurons	145
	sl_neurons	117
	activation_funct	relu
	dropout_rate	0.16425027421033847
Random Forest	random_state	721077
	n_estimators	304
	max_depth	42
	max_features	None
	class_weight	balanced
	criterion	entropy
	min_samples_split	8
	min_samples_leaf	4

Continued on next page

Table B.8 – Continued from previous page

Model	Hyperparameters	Value
Support Vector Machine	random_state	721077
	penalty	l1
	loss	squared_hinge
	dual	False
	tol	0.5434789046958924
	class_weight	balanced
	C	141.02468595833872
	max_iter	2000
XGBoost	random_state	721077
	max_depth	31
	learning_rate	0.2770533045246825
	booster	gbtree
	gamma	0.020860882761651944
	n_estimators	227
	scale_pos_weight	105.57949790794979

Model	Features
Active Learning	difference_from_iban_cc_mean1d, is_national_iban, time_since_same_iban, difference_from_iban_cc_mean14d, cc_asn_count7d, difference_from_iban_mean7d, amount_std7d, difference_from_cc_asn_mean7d, cc_asn_count30d, difference_from_amount_mean14d, difference_from_cc_asn_mean30d, time_since_same_cc_asn
Logistic Regression	iban_sum30d, difference_from_amount_mean7d, cc_asn_count1d, iban_sum7d, difference_from_cc_asn_mean14d, Time_y, iban_cc_mean14d, is_new_cc_asn, is_national_iban, iban_sum1d, ip_std14d, ip_std30d, difference_from_cc_asn_mean7d, ip_sum30d, ip_std7d, ip_sum14d, is_new_ip, time_since_same_cc_asn, iban_sum14d, difference_from_cc_asn_mean30d, time_since_same_iban
Neural Network	iban_mean30d, is_international, is_national_iban, iban_count1d, ip_count14d, iban_count30d, difference_from_iban_mean1h, cc_asn_mean1d
Random Forest	time_since_same_session, amount_std7d, Time_y, amount_std1h, iban_std30d, Time_x, is_national_iban, cc_asn_mean1d, time_since_same_iban, is_international, difference_from_cc_asn_mean14d, difference_from_iban_cc_mean14d, amount_sum7d, Amount, iban_std1d, difference_from_ip_mean1d, difference_from_amount_mean14d, iban_count14d, cc_asn_count30d, iban_count1h
Support Vector Machine	iban_sum14d, is_national_iban, ip_sum14d, difference_from_cc_asn_mean7d, difference_from_cc_asn_mean14d, ip_std30d, difference_from_iban_mean1h, ip_std14d, time_since_same_cc_asn, is_new_iban_cc, time_since_same_iban, difference_from_amount_mean14d
XGBoost	difference_from_cc_asn_mean30d, amount_std7d, cc_asn_count1d, iban_cc_count1h, amount_sum30d, time_since_same_iban, session_sum1h, difference_from_amount_mean7d, Time_y, ip_mean14d, is_international, amount_mean30d, ip_std7d, is_national_iban

Table B.9: Features for models trained on all types of fraudulent transactions

Model	Features
Active Learning	is_new_iban_cc, cc_asn_count30d, iban_std1d, ip_std7d, ip_count30d, difference_from_cc_asn_mean14d, difference_from_cc_asn_mean7d, difference_from_amount_mean30d, is_new_iban, amount_std14d
Logistic Regression	cc_asn_count1h, is_international, is_new_iban, Time_x, session_count14d, cc_asn_mean30d, ip_std30d, cc_asn_count1d, amount_mean14d, amount_std14d, difference_from_ip_mean14d, is_new_iban_cc
Neural Network	amount_mean7d, cc_asn_count1h, ip_count14d, difference_from_iban_cc_mean7d, is_national_iban, iban_cc_mean30d, cc_asn_mean30d, amount_mean14d, ip_mean7d
Random Forest	iban_sum7d, time_since_same_iban_cc, cc_asn_mean30d, iban_count7d, iban_count1d, amount_mean14d, is_new_cc_asn, Amount, is_international, Time_y
Support Vector Machine	iban_sum1h, iban_sum1d, amount_mean14d, cc_asn_mean30d, is_international, ip_std14d, difference_from_ip_mean14d, amount_std1h, Time_x, difference_from_ip_mean1d, iban_sum7d, amount_mean7d
XGBoost	amount_mean14d, is_international, ip_std7d, difference_from_amount_mean1h, difference_from_session_mean1d, cc_asn_mean30d

Table B.10: Features for models trained on fraudulent transactions characterized by High profile and Information Stealing scheme

Model	Features
Active Learning	amount_std30d, ip_count7d, difference_from_iban_cc_mean1d, difference_from_amount_mean30d, is_new_cc_asn, difference_from_ip_mean1h, iban_std1d, session_count1h, ip_count1d, iban_cc_count14d, iban_count1h, difference_from_cc_asn_mean14d, ip_std1d, difference_from_session_mean1h, ip_count30d, difference_from_iban_mean1h, iban_count1d, difference_from_iban_mean30d, difference_from_ip_mean1d, difference_from_ip_mean7d, session_std1h
Logistic Regression	is_new_iban_cc, iban_sum1d, difference_from_iban_mean7d, ip_sum1d, iban_std7d, is_national_iban, difference_from_ip_mean1h, session_std1h, Time_x
Neural Network	Amount, iban_cc_sum14d, difference_from_ip_mean1h, is_new_ip, iban_std1d, is_national_iban
Random Forest	difference_from_iban_mean7d, Amount, iban_count1d, iban_sum30d, is_national_iban, difference_from_ip_mean1h, ip_sum7d, iban_count1h, is_new_iban_cc
Support Vector Machine	ip_count7d, iban_std7d, is_new_iban_cc, difference_from_session_mean1h, Amount, is_national_iban
XGBoost	difference_from_amount_mean7d, session_sum1h, iban_sum14d, is_international, Amount, iban_std7d, is_new_iban_cc, session_std1h, difference_from_iban_mean1h, ip_sum7d, iban_count14d, iban_count30d, iban_mean30d

Table B.11: Features for models trained on fraudulent transactions characterized by High profile and Transaction Hijacking scheme

Model	Features
Active Learning	iban_count30d, difference_from_iban_mean14d, is_new_iban_cc, difference_from_ip_mean14d, cc_asn_count7d, difference_from_amount_mean7d, cc_asn_count14d, time_since_same_iban, ip_std30d, difference_from_amount_mean14d, iban_cc_std30d, cc_asn_count1h
Logistic Regression	ip_std7d, time_since_same_iban, cc_asn_count1h, time_since_same_cc_asn, difference_from_amount_mean14d, cc_asn_count14d, iban_count1d, difference_from_iban_mean14d, time_since_same_session, iban_std7d, difference_from_iban_mean7d, is_national_iban, ip_std1h, difference_from_amount_mean30d, difference_from_amount_mean1h, difference_from_ip_mean7d, session_count14d, ip_std30d, difference_from_amount_mean1d, ip_std1d, difference_from_ip_mean14d, iban_std1d, difference_from_iban_mean30d, iban_count1h, ip_std14d
Neural Network	is_national_iban, time_since_same_cc_asn, iban_std1d, cc_asn_mean7d, ip_std14d, difference_from_amount_mean30d, difference_from_amount_mean7d, difference_from_iban_mean7d
Random Forest	Time_x, is_new_iban_cc, cc_asn_mean7d, iban_sum14d, Time_y, time_since_same_cc_asn, iban_count1d, time_since_same_iban, difference_from_iban_mean14d, is_new_cc_asn, ip_std7d, difference_from_amount_mean7d, iban_std1d, is_international, ip_count30d
Support Vector Machine	session_sum7d, iban_sum1d, iban_std1d, difference_from_session_mean1d, iban_sum7d, iban_std7d, is_new_iban_cc, time_since_same_cc_asn, time_since_same_iban, is_new_ip, cc_asn_sum7d, difference_from_iban_mean14d, is_national_iban, cc_asn_sum1h, is_new_cc_asn
XGBoost	cc_asn_mean30d, difference_from_iban_mean14d, iban_std1d, difference_from_amount_mean14d, is_new_ip, time_since_same_iban, Time_x, cc_asn_sum1h, iban_mean30d, iban_count1d, iban_sum14d, cc_asn_sum30d, cc_asn_count30d, iban_std7d, time_since_same_session, is_international, cc_asn_count1d, Time_y

Table B.12: Features for models trained on fraudulent transactions characterized by Medium profile and Information Stealing scheme

Model	Features
Active Learning	is_new_ip, is_international, cc_asn_std30d, time_from_previous_trans_global, iban_cc_count30d, difference_from_ip_mean1d, ip_count30d, iban_std1d, is_new_iban, is_new_iban_cc, difference_from_iban_mean1h, iban_count30d, difference_from_amount_mean30d, is_national_iban, ip_count14d
Logistic Regression	is_new_ip, is_national_iban, time_from_previous_trans_global
Neural Network	iban_cc_mean7d, is_national_iban, iban_std1d, Amount
Random Forest	time_since_same_iban, ip_sum1h, difference_from_iban_mean30d, amount_count7d, iban_cc_count30d, difference_from_ip_mean1d, is_national_iban, difference_from_iban_mean14d, Amount, difference_from_amount_mean30d, ip_sum1d, Time_y, cc_asn_sum14d, is_new_iban, difference_from_amount_mean14d, ip_std1d, ip_sum7d, ip_count30d, iban_cc_count14d, cc_asn_sum30d, difference_from_iban_mean1h, iban_count1h, difference_from_iban_mean7d, Time_x, iban_mean30d, iban_sum1d, ip_sum30d, iban_count7d
Support Vector Machine	amount_count7d, ip_count14d, is_international, is_new_ip, iban_cc_count30d, is_national_iban
XGBoost	ip_std1d, ip_sum1d, difference_from_iban_mean1d, difference_from_ip_mean1h, iban_cc_mean30d, time_since_same_iban, Amount, iban_count14d, iban_cc_mean7d, is_national_iban, ip_sum30d, iban_count7d, iban_count1d, ip_sum1h, difference_from_amount_mean14d, difference_from_iban_mean7d, Time_x, difference_from_ip_mean14d, time_from_previous_trans_global, is_international, ip_sum7d

Table B.13: Features for models trained on fraudulent transactions characterized by Medium profile and Transaction Hijacking scheme

Model	Features
Active Learning	is_new_iban_cc, iban_count7d, difference_from_amount_mean7d, cc_asn_count7d, ip_std7d, difference_from_amount_mean30d, difference_from_ip_mean14d, ip_count14d, time_since_same_cc_asn, iban_count14d, iban_std7d, time_since_same_iban, difference_from_iban_mean1d, iban_count30d
Logistic Regression	ip_std7d, cc_asn_mean1d, time_since_same_iban, ip_count14d, cc_asn_count1h, time_since_same_cc_asn, difference_from_amount_mean14d, iban_count1d, iban_std7d, is_national_iban, iban_count30d, difference_from_amount_mean30d, cc_asn_count1d, iban_count7d, cc_asn_sum30d, session_count14d, difference_from_amount_mean1d, cc_asn_sum7d, iban_sum30d, difference_from_ip_mean14d, difference_from_iban_mean1d, cc_asn_sum14d, Amount, iban_count1h, ip_std14d, cc_asn_sum1h
Neural Network	cc_asn_count14d, cc_asn_count1d, ip_std1d, is_national_iban, iban_sum14d, time_since_same_iban, cc_asn_count1h, difference_from_ip_mean7d, ip_mean30d
Random Forest	difference_from_amount_mean7d, iban_count30d, Time_y, cc_asn_count14d, is_new_iban_cc, iban_cc_std30d, ip_std14d, difference_from_cc_asn_mean1h, is_international, difference_from_amount_mean14d, session_count14d, ip_std7d, cc_asn_count1h, ip_count30d, ip_std30d, Amount, time_since_same_cc_asn, iban_std7d, difference_from_ip_mean7d, cc_asn_count7d, difference_from_iban_mean1h, iban_count1h, iban_std1d, difference_from_iban_mean1d, ip_count14d, ip_std1d, difference_from_iban_mean30d, session_sum1d, is_new_iban, is_new_ip, ip_mean30d, difference_from_amount_mean1d, cc_asn_count1d, cc_asn_sum1h, cc_asn_sum14d, difference_from_ip_mean14d, iban_count7d, cc_asn_count30d, iban_count14d, iban_count1d, difference_from_iban_mean7d, time_since_same_iban, amount_std1h, difference_from_session_mean30d, time_since_same_session, is_new_cc_asn, difference_from_iban_mean14d
Support Vector Machine	ip_std14d, ip_std30d, iban_sum7d, is_national_iban, iban_cc_mean14d, ip_std7d, cc_asn_sum1d, difference_from_iban_mean1h, difference_from_session_mean30d, difference_from_iban_mean7d, difference_from_iban_mean1d, difference_from_amount_mean7d, iban_sum14d, iban_sum1h, ip_std1d, cc_asn_sum14d, difference_from_iban_mean14d, cc_asn_sum30d, difference_from_iban_mean30d, ip_sum30d, difference_from_cc_asn_mean1h
XGBoost	cc_asn_count30d, difference_from_iban_mean1d, ip_std1d, difference_from_amount_mean14d, Amount, ip_std14d, difference_from_session_mean30d, cc_asn_sum1d, ip_count30d, iban_std1d, difference_from_cc_asn_mean1h, time_since_same_iban, iban_cc_std30d, iban_count14d, iban_cc_mean14d, difference_from_amount_mean7d, session_sum1d, amount_std1h, iban_sum30d, cc_asn_mean30d, iban_mean30d, iban_count30d, ip_std30d, iban_sum14d, cc_asn_count14d, is_new_ip, ip_std7d, difference_from_iban_mean14d, session_count14d, is_new_iban, difference_from_amount_mean30d, cc_asn_sum1h, is_international, difference_from_iban_mean1h, difference_from_ip_mean14d, ip_count14d, difference_from_iban_mean30d, difference_from_ip_mean7d, cc_asn_sum7d, cc_asn_sum14d, difference_from_iban_mean7d, cc_asn_count1d, is_new_cc_asn, iban_count1d, cc_asn_count1h, iban_count7d, iban_count1h, time_since_same_session, time_since_same_cc_asn, cc_asn_count7d, is_national_iban, Time_x, ip_sum30d, iban_sum7d, iban_std7d, Time_y, difference_from_amount_mean1d

Table B.14: Features for models trained on fraudulent transactions characterized by Low profile and Information Stealing scheme

Model	Features
Active Learning	is_new_iban_cc, iban_std1d, iban_cc_count30d, is_international, iban_cc_std30d, ip_count14d, is_new_cc_asn, session_std1h, is_new_ip, is_national_iban, difference_from_iban_mean1h, ip_count7d, difference_from_cc_asn_mean7d, difference_from_cc_asn_mean1d, time_from_previous_trans_global, difference_from_iban_mean7d
Logistic Regression	is_national_iban, is_new_ip, iban_sum14d, difference_from_iban_mean14d, time_since_same_iban, iban_std1d, difference_from_iban_mean1h, is_new_cc_asn
Neural Network	ip_count30d, iban_cc_mean1d, iban_cc_std1d, iban_mean30d, is_new_cc_asn, is_national_iban, Amount, is_international, iban_sum30d, iban_std1d, is_new_ip, iban_cc_std30d, iban_sum7d
Random Forest	iban_cc_sum1d, iban_count1h, is_new_cc_asn, Time_y, iban_cc_std14d, session_sum1h, iban_cc_mean7d, is_national_iban, iban_cc_sum7d, is_new_iban, ip_mean30d, ip_count1d, time_from_previous_trans_global
Support Vector Machine	iban_std7d, difference_from_iban_mean1d, is_new_ip, iban_std1d, iban_cc_mean7d, is_national_iban, time_from_previous_trans_global, ip_count1d
XGBoost	iban_cc_mean7d, difference_from_ip_mean1h, iban_cc_mean30d, ip_sum30d, iban_cc_std1d, session_sum1h, time_since_same_iban, difference_from_iban_mean7d, difference_from_iban_mean1d, is_national_iban, ip_count30d, iban_std1d, iban_cc_sum1h, Time_x, iban_count1h, is_international, iban_cc_sum1d, ip_std14d, difference_from_cc_asn_mean7d, time_from_previous_trans_global, iban_cc_std14d

Table B.15: Features for models trained on fraudulent transactions characterized by Low profile and Transaction Hijacking scheme

C | Appendix C: ML Performance Metrics

We define the Machine Learning performance metrics that we employ in our research to rate the effectiveness of ML models. We categorize fraudulent transactions as class 1 (positive class) and legitimate transactions as class 0 (negative class). Thus, True Positives (TP) represent correctly classified frauds, True Negatives (TN) represent correctly classified legitimate transactions, False Positives (FP) represent legitimate transactions classified as frauds, and False Negatives (FN) represent frauds classified as legitimate transactions. Based on that, we define the following metrics:

- **Precision**: quantifies the proportion of true positive predictions out of all positive predictions.

$$precision = \frac{TP}{TP + FP}$$

- **Recall**: also known as sensitivity or true positive rate, measures the proportion of true positive predictions out of all actual positive instances.

$$recall = \frac{TP}{TP + FN}$$

- **F1-score**: combines precision and recall into a single metric, providing a balanced measure of a model's performance. It is the harmonic mean of precision and recall.

$$F1 - score = 2 * \frac{2TP}{2TP + FN + FP}$$

- **The False Positive Rate (FPR)**: calculates the proportion of positive predictions out of all negative instances. It represents the rate of incorrectly classifying negative instances as positive.

$$FPR = \frac{FP}{FP + TN}$$

- **Area Under the Receiver Operating Characteristics Curve (AUC-ROC)**: is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. The Higher the AUC, the better the model is at distinguishing between classes.

$$ROC = \int_{x=0}^{1} Recall * (FPR^{-1}(x))dx$$

- **Area Under the Precision-Recall Curve (AUC-PRC)**: computes precision-recall pairs for different probability thresholds.

$$PRC = \int_{x=0}^{1} Precision * (Recall^{-1}(x))dx$$

- **Matthew Correlation Coefficient (MCC)**: is a measure of the quality of binary classifications, taking into account true positive, true negative, false positive, and false negative predictions. It ranges from -1 to 1, where 1 indicates perfect prediction, 0 indicates random prediction, and -1 indicates complete disagreement between predictions and actual values.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

  In this study, we also employ a **Weighted MCC** that assigns samples of the two classes a weight inversely proportional to their presence in the dataset:

$$w_0 = \frac{TP + FN}{TP + FN + FP + TN}, w_1 = \frac{TN + FP}{TP + FN + FP + TN}$$

$$WMCC = \frac{w_1 TP * w_0 TN - w_0 FP * w_1 FN}{\sqrt{(w_1 TP + w_0 FP) w_1 (TP + FN) w_0 (TN + FP)(w_0 TN + w_1 FN)}}$$

When dealing with imbalanced data the value of the accuracy score does not provide a meaningful result. For instance, if we consider a dataset where the positive class is present only in 10% of the total instances, a dumb classifier that simply predicts the negative class all the time, could get 90% accuracy. This occurs because accuracy, as a performance metric, emphasizes the overall correctness of predictions without taking into account the distribution of classes. In place of the standard accuracy metric we use a custom accuracy, called **Cost Accuracy** [46], which gives more weight to the correct classification of frauds (False Negative, True Positives) with respect to legitimate transactions. We define the

Cost Accuracy as:

$$CostAccuracy = 1 - \frac{FP + k * FN}{FP + TN + k * (TP + FN)}$$

where k is:

$$k = \frac{TN + FP}{TP + FN}$$

List of Figures

List of Tables