CentraleSupélec

Comparative Analysis of Natural Language Processing and Gradient Boosting Trees Approaches for Fraud Detection

Ludovica Lerma
Student ID: 976806

Supervision:

Fabrice Popineau	LISN/CentraleSupélec	Arpad Rimmel	LISN/CentraleSupélec
Marcello Restelli	Politecnico di Milano	Bich-Liên Doan	LISN/CentraleSupélec

Abstract

With the increasing popularity of digital payments, fraud detection systems have become indispensable in limiting monetary losses for both customers and card-provider companies. Recognizing the significance of this issue, online payment platforms actively incorporate robust fraud detection systems into their infrastructure. This thesis presents a straightforward approach to fraud detection inspired by natural language processing (NLP) techniques. The proposed methodology begins by leveraging the Continuous Bag-of-Attributes (CBOA) neural network embedding, which projects transactional data into a hyper-dimensional space, facilitating the extraction of an extended range of features. This embedding technique empowers the system to capture contextual relationships within the data. Subsequently, the embedded data undergoes processing through a Long Short-Term Memory (LSTM) layer, enabling the model to capture temporal correlations between sequential transactions. The LSTM layer adds a dynamic element to the fraud detection system, allowing it to adapt and learn from the sequential nature of transactional data. To classify transactions as fraudulent or legitimate, the processed data passes through two dense layers. These layers serve as the final decision-making components of the model, using classification algorithms to differentiate between fraudulent and legitimate transactions. Finally, this thesis conducts a direct comparison between the NLP-based approach and Gradient Boosting Trees (GBT), which is a type of Decision Trees. This comparison carries significant importance, as Decision Trees have proven to be a highly effective technique in credit card fraud detection. The evaluation encompasses both the conventional GBT approach and an exploration of its performance when augmented with NLP embeddings. Experimental results are shown, but more advanced techniques, such as transformers and attention mechanisms, promise to surpass the capabilities of the current methodologies. These advanced techniques possess the complexity to capture intricate patterns and dependencies within transactional data, which could potentially enhance the accuracy and efficacy of fraud detection systems further. Overall, this thesis offers a valuable contribution to the field of fraud detection, presenting a straightforward comparison between a novel approach and a classic approach to the problem, while highlighting the potential for further advancements through the use of more sophisticated techniques.

Sommario

Con l'aumentare della popolarità dei cosiddetti pagamenti digitali, i sistemi di rilevamento di frodi sono diventati indispensabili nel limitare le perdite monetarie sia per i clienti che per le società emittenti le carte. Conscie dell'importanza di questa problematica, le piattaforme di pagamento online incorporano sistemi sempre più innovativi nelle proprie infrastrutture. Questa tesi presenta dunque un approccio per il rilevamento di frodi bancarie ispirato alle tecniche di Natural Language Processing (NLP). L'algoritmo presentato è diviso in due fasi. In una prima fase la rete neurale Continuous Bag-of-Attributes (CBOA) proietta i dati delle transazioni in uno spazio iper-dimensionale, facilitando l'estrazione di features che catturarano le relazioni contestuali all'interno dei dati. Successivamente, i dati così trasformati vengono elaborati attraverso un layer LSTM (Long Short-Term Memory), consentendo al modello di catturare le correlazioni temporali tra transazioni sequenziali. Infine, per classificare le transazioni come fraudolente piuttosto che legittime, i dati elaborati passano attraverso due dense layers. Questi layers fungono da componenti decisionali finali del modello. In ultimo, questa tesi effettua un confronto diretto tra l'approccio basato su NLP e Gradient Boosting Trees (GBT), un tipo di Decision Trees. Questo confronto riveste un'importanza significativa, poiché i Decision Trees si sono dimostrati una tecnica altamente efficace nella rilevazione di frodi con carte di credito. La valutazione comprende sia l'approccio GBT convenzionale sia una esplorazione delle sue prestazioni quando viene potenziato con gli embedding di NLP. I risultati sperimentali vengono dunque presentati, tuttavia l'utilizzo di tecniche più avanzate, come Tranformers e metodi basati sull'"Attention mechanism", promette di sorpassare i metodi correnti. Tali tecniche possiedono la complessità necessaria per catturare dipendenze più complesse all'interno delle transazioni, potenziando ulteriormente l'efficacia dei sistemi di rilevamento di frodi. Complessivamente, questa tesi offre un prezioso contibuto nel campo del rilevamento di frodi bancarie, illustrando un confronto diretto tra un approccio innovativo ed un approccio classico al problema e mettendo in evidenza i possibili sviluppi nel campo attraverso l'uso di tecniche più avanzate.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Extended Word
GBT	Gradient Boosting Trees
LSTM	Long-Short Term Memory
NLP	Natural Language Processing
P-RCE	Parallel Resilient Back-Propagation with Constrained Energy
SVDD	Support Vector Data Description
SVM	Support Vector Machine
CBOW	Continuous-Bag-of-Words model
CBOA	Continuous-Bag-of-Attributes model
GRU	Gated Recurrent Unit
PCA	Principal Component Analysis
t-SNE	T-distributed Stochastic Neighbor embedding
std	Standard Deviation
nbstd	Cumulative Standard Deviation
LGBM	Light Gradient Boosting Machine

Chapter

1 Introduction

1.1 The Credit Card Fraud Problem

Over the past two decades, advancements in technology have not only facilitated but also popularized **digital payments**.

However, the outbreak of the covid-19 pandemic has further accelerated this trend, emphasizing its significance not only for economic prosperity, but also as a valuable resource for maintaining public health. In fact, the European area alone witnessed a notable increase in **non-cash payments**, which reached €101.6 billion, with *an additional*

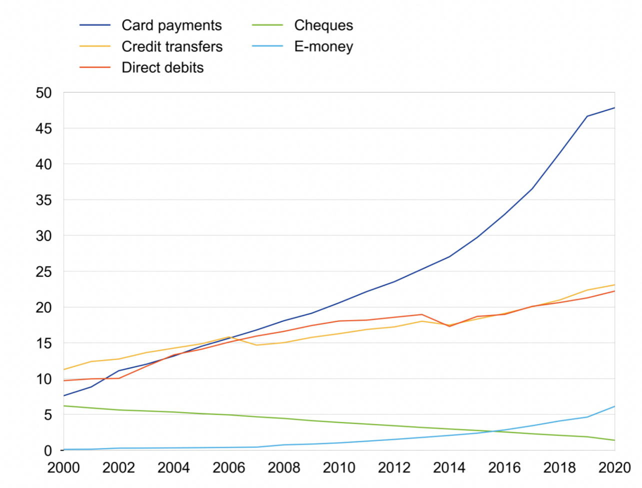


Figure 1.1: Use of the main payment services in the euro area, source: Payments statistics: 2020

€3.7 billion added in recent years. The chart in figure 1.1 shows the development of the main payment services in the Euro area from 2000 to 2020[1].

Unfortunately, **fraudsters** have also been quick to adapt to these developments, leading to a corresponding **rise in digital fraud incidents**. Globally, the total monetary losses due to fraud amount to €36.8 billion[2]. Furthermore, the inability to effectively combat fraud not only leads to financial losses, but also damages the credibility and reliability of card-provider companies, exacerbating the overall impact.

Given this context, the importance of developing robust fraud detection systems becomes evident.

1.2 Solution and goals

The **main objective** of this work is to propose an **innovative fraud detection system** that can be integrated into a high-performance online transactional platform developed by a global software and services provider in the cashless payments industry.

The **currently adopted approach** relies on leveraging the expertise of domain experts to extract meaningful features from the data, which are then fed into a simple classifier like **Gradient Boosted Trees** (GBT). However, this relatively simple model faces **significant challenges** in effectively addressing the complexities associated with fraud detection.

Therefore, this thesis proposes a **robust solution** in the form of a **deep-learning based model**. The model architecture consists of multiple cascaded networks: the first network utilizes the **Bag-Of-Attributes** model to extract features from the dataset. The second network employs a **Long Short-Term Memory** (LSTM) to capture patterns from credit card histories. Finally, two Dense Layers are employed to classify the instances as anomalous or not.

1.3 Original Contribution

This thesis makes an original contribution by employing and analyzing natural language processing (NLP) methodologies in the context of fraud detection. While NLP models have been successfully applied in various domains, such as facial expression recognition[3], 3D object recognition[4], and time-series classification[5], only a limited number of studies have explored their application in the specific domain of fraud detection.

This thesis distinguishes itself by conducting a direct comparison between an NLP-based approach and Gradient Boosting Trees (GBT), which is an implementation of the

Decision Trees methodology. This comparison carries significant importance, as Decision Trees have proven to be a highly effective technique employed in credit card fraud detection[6][7][8]. The evaluation encompasses both the conventional GBT approach and an exploration of its performance when augmented with NLP embeddings.

Therefore, this thesis offers a comprehensive analysis and evaluation of the effectiveness of an NLP-based approach in fraud detection. Furthermore, the inclusion of a benchmark comparison with the GBT approach provides meaningful insights into the relative performance of these methods.

One notable **paper** that **served as inspiration** for the approach implemented in this report is "FraudMemory: Explainable Memory-Enhanced Sequential Neural Networks for Financial Fraud Detection" by Yang et al.[9]. By leveraging NLP techniques, this research demonstrates promising advancements in the field of fraud detection. **Building upon this foundation**, the proposed model in this thesis incorporates similar concepts to enhance fraud detection capabilities in the credit card fraud detection task.

1.4 Thesis Structure

The following of this thesis is structured as follows:

- the ***Problem formulation section***[2]: This section presents the formulation of the problem and provides an overview of existing works in the literature that address similar tasks.
- the ***Proposed Model section***[3]: In this section, the proposed solution is introduced and explained in detail.
- the ***Dataset section***[4]: This section examines the characteristics of the dataset used in the study.
- the ***Technical Implementation section***[5]: The technical implementation section describes the step-by-step process followed to develop the final model, including snippets of code and relevant technical details.
- the ***Experiments section***[6.3]: This section presents the experiments conducted and provides commentary on the results obtained.
- the ***Results section***[7]: The results section describes the results of the experiments and presents the findings.
- the ***Discussion section***[8]: In this section, a comparison and analysis of the results obtained are presented, along with relevant discussions.

- The ***Conclusion section***[9]: The conclusion section summarizes the main findings of the study and offers potential directions for future research and development of the proposed model.

Chapter

2 Problem Formulation

In this section, we will provide a deeper insight into the problem at hand.

Firstly, the problem is stated. We will analyze what type of input we are dealing with, and what type of output we are expected to produce.

Secondly, we provide an overview of the **related work** already carried out on the topic. Relevant research, studies, and approaches already explored in the field will be discussed and reviewed.

2.1 Problem Statement and Goals

As previously mentioned, the task involves designing and implementing a **fraud detection system**. Our system is expected to determine, given an input payment transaction, whether the transaction is legitimate or fraudulent.

To accomplish this task, we have access to a large dataset of **100 million labeled transactions** from a real payment service. Each transaction consists of several attributes, including both categorical and continuous variables. Of particular interest are the *"card"* attribute, which identifies the card used for the transaction and links it to the series of transactions associated with the same card, and the *"fraud"* flag attribute, which indicates whether a transaction is classified as fraudulent.

It is worth noting that only a **small fraction** (less than 1%) of the transactions in our dataset are flagged as fraud. This makes the dataset highly imbalanced, presenting a challenge in developing an effective classification model for time-series input data.

Given the characteristics of the problem, we can include it in the **anomaly detection** framework. Anomaly detection refers to the task of detecting unusual or anomalous instances within a dataset.

Our model aims to surpass the performance of the current fraud detection system implemented on the platform, which uses a Gradient Boosting Trees (GBT) classification model. More details about GBT will be provided in the dedicated section[3.3].

Next, we will proceed to formally define the task.

Fraud Detection

Consider a dataset $D_n = (x_i, y_j)_{i=1}^n$ where $x_i \in X \subseteq \mathbb{R}^d$ and $y_i \in Y = 0, 1$. Each payment i is described by its features x_i (*e.g. amount, date,...etc*) and its label y_i flagging whether it is a fraud, $y_i = 1$, or not, $y_i = 0$.
Fraud detection consists of estimating a function $f : X \mapsto Y$ using D_n, i.e. a function which predicts whether a payment i is fraudulent based on its features x_i.
Fraud Detection is considered a challenging task for three main reasons:

- **Imbalanced learning**: non-fraudulent payments are significantly more numerous
$$Card((x_i, y_i)|y_i \in D_n, y_i = 0) \gg Card((x_i, y_i)|y_i \in D_n, y_i = 1)$$
than fraudulent ones.
- **Concept drift**: fraudsters change their behaviors due to a cat-and-mouse game.
$$D_n^{train} \sim p_{train}(x, y)$$
$$D_n^{test} \sim p_{test}(x, y)$$
$$p_{test}(x, y) \neq p_{train}(x, y)$$
- **Explainability**: for regulatory reasons, it is necessary to account for a model's prediction.

Due to these characteristics, in particular extreme imbalanced datasets and concept drift, standard machine learning methods perform poorly. In the following section, the most popular methods to perform fraud detection are investigated.

2.2 Related works

Fraud detection, as a domain within **anomaly detection**, focuses on identifying rare instances, such as fraudulent activities, that significantly deviate from most of the data, which comprises non-fraudulent instances.

The literature on anomaly detection is rich with a wide range of approaches and techniques. To gain a comprehensive understanding of the field, "Anomaly Detection:

A Survey" by Chandola et al. provides a detailed review of different methodologies, algorithms, and applications in anomaly detection[10].

Based on the availability of labeled data, anomaly detection techniques can be classified into supervised and unsupervised techniques.

Supervised techniques rely on labeled data, where both normal and anomalous instances are explicitly identified. These techniques involve training a model on the labeled data to learn the patterns and characteristics of normal instances, enabling them to classify new instances as normal or anomalous based on the learned information.

On the other hand, **unsupervised techniques** do not require labeled data and focus solely on the characteristics of the data itself. These techniques aim to identify patterns and structures that distinguish normal instances from anomalies without prior knowledge of the anomalies. Unsupervised techniques often rely on statistical methods, clustering algorithms, or density-based approaches to detect deviations from the norm.

Both supervised and unsupervised techniques have their strengths and weaknesses, and their suitability depends on the availability of labeled data and the specific requirements of the fraud detection task at hand.

Supervised methods

Traditional **supervised methods** focus on training a **classifier** using D_n by estimating either the joint distribution $p(x,y)$ for **generative models** or the conditional distribution $p(y|x)$ for **discriminative models**.

Among supervised techniques, those based on **neural networks**, have gained popularity in fraud detection tasks. One of the early works in this area is the paper *"Credit card fraud detection with a neural-network"* by Ghosh and Reilly[11]. They applied a neural network called **Parallel Resilient Back-Propagation with Constrained Energy**(P-RCE) to the fraud detection task. The authors performed feature extraction on raw input data fields to create more meaningful features, which were then fed into the P-RCE neural network. This approach demonstrated superior performance compared to classical statistical methodologies.

A similar approach was implemented by Patidar and Lokesh Sharma in their work *"Credit Card Fraud Detection Using Neural Network"*[12]. They utilized a **P-RCE** neural network along with a **genetic algorithm**. The genetic algorithm was employed to select the best performing neural network from an initial randomly generated pool of networks. This approach enhanced the effectiveness of the neural network in detecting credit card fraud.

In the paper *"Credit Card Fraud Detection Using Convolutional Neural Networks"* by Kang Fu et al.[13], the authors proposed the use of **convolutional neural networks** (CNN) for credit card fraud detection. They applied CNN and reshaped the data into feature matrices based on time windows, allowing the network to capture temporal patterns in the data. This technique demonstrated promising results in fraud detection tasks.

However, these approaches often yield relatively unsatisfactory results in fraud detection due to specific challenges, such as highly imbalanced learning and concept drift [14].

Researchers have proposed various alternative techniques to improve performance in anomaly detection scenarios. The methodologies applied in the following two papers are among the most interesting.

One-class Classification

One-Class Classification consists in estimating the support of the distribution of normal data, i.e. non-fraudulent payments in our case, in a well chosen latent space Z such that data points not belonging to that support are considered anomalous or fraudulent. One-Class Classification is a supervised method, since it requires labeled data to split the data into a training set only composed of non-fraudulent payments:

$$D_{train} = x|(x, y) \in D, y = 0$$

This approach offers several **advantages**:

- It is less susceptible to concept drift, which consists of the fact that the data distribution changes over time.
- It is effective in handling imbalanced datasets, where the positive class (fraudulent payments) is significantly underrepresented compared to the negative class (non-fraudulent payments).
- It is well-suited for tasks where the positive class does not exhibit a consistent pattern or structure within the feature space. This means fraudulent payments may not follow a specific set of characteristics, making it difficult to define a clear decision boundary between normal and fraudulent instances using traditional classification methods.

The typical methods employed in **One-Class Classification** are **kernel-based techniques** like **Support Vector Data Description** (SVDD) or **One-Class Support Vector Machine** (SVM). Let's consider a kernel mapping $\phi : X \mapsto Z$. In this context:

- **One-Class SVM**[15] involves identifying the hyperplane in Z that maximizes the distance from the origin and effectively separates all data points from the origin.

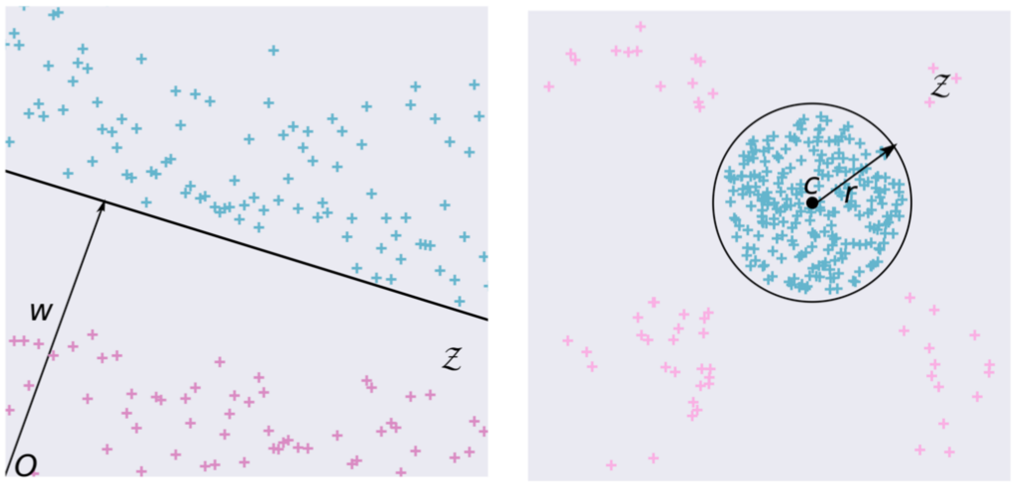


Figure 2.1: Left: One-Class SVM, Right: SVDD

- On the other hand, **SVDD**[16] aims to locate the smallest hyper-sphere in Z that encompasses all the data points.

The two approaches are shown visually in figure 2.1.

Due to their inherent computational complexity, methods like **One-Class SVM** and **SVDD**, being kernel-based models, are **elegant but often impractical for large datasets**. In the case of a dataset containing 100 million transactions, like the one used in this thesis, these approaches are not well-suited due to their intractability.

FraudMemory: Explainable Memory-Enhanced Sequential Neural Networks for Financial Fraud Detection

In their recent study titled *"FraudMemory: Explainable Memory-Enhanced Sequential Neural Networks for Financial Fraud Detection"* Yang et al.[9] propose a hybrid fraud detection system that combines sequential and memory-enhanced methods to enhance explainability.

The core idea of the paper is to model the normal behavior of each user and identify transactions that significantly deviate from this behavior as potential frauds.

The authors extract valuable information from transactions through two distinct representations: the **user profile** representation and the **log representation**.
The **user profile representation** involves constructing three graphs: the *recency graph* (GR), the *frequency graph* (GF), and the *money graph* (GM). In these graphs, users are represented as nodes, and the edges represent transactions between users. The weights

of the edges are determined based on the average time interval between transactions, the frequency of transactions, and the average monetary value of each transaction, respectively. A translation-based embedding method, **TransE**[17] is used to generate *latent vectors* from each graph for each user.

These vectors are then concatenated into a 3D matrix, creating a comprehensive representation for each user. Lastly, a **clustering method**, e.g. K-Nearest Neighbors, is applied to group users with similar behavior.
The **log representation**, on the other hand, aims to extract information from the attribute level of transactions. To achieve this, the authors implement a modified version of the Continuous-Bag-of-Words model (CBOW), called Continuous bag-of-attributes model (CBOA).

These two representations are then fed into a Gated Recurrent Unit (**GRU**) model to extract sequential patterns. The GRU model utilizes the log sequence $l_1^u, l_2^u, \dots, l_t^u$ for user *u* to calculate the current hidden state vector h_t^u at time-step t based on the previous hidden state vector h_{t-1}^u:

$$h_t^u = GRU(h_{t-1}^u, l_i^u, \theta)$$

where θ denotes the GRU parameters to learn.

Finally, a Multilayer Perceptron (**MLP**) is employed to evaluate the sequential representation and assign a fraud score to each transaction.

$$Score_{sequence} = MLP(h_t^u)$$

During the application phase, two additional memory networks are used to improve performance and address the concept drift problem. The sequential representation of incoming transactions is indeed used as a query for the two memory networks. The output latent memory vector of this query, denoted as m_t^u, represents the divergence between the sequential representation and the past behavior of the user. It is obtained through an attentive combination of memory networks:

$$m_t^u = \sum_{i=1}^{m^u} w_i^u \cdot m_i^u$$

Here, w_i^u is the attention weight of memory m_i^u. To determine the weights between the query and the memory, the Softmax function is utilized. This computation assigns higher weights to instances where the query and the memory exhibit greater similarity:

$$w_i^u = Softmax(-Distance(h_t^u, m_i^u)) = \frac{e^{-Distance(h_t^u, m_i^u)})}{e^{-Distance(h_t^u, m_j^u)})}$$

The dissimilarity between the query vector and memory slots is determined by using the Euclidean distance function.

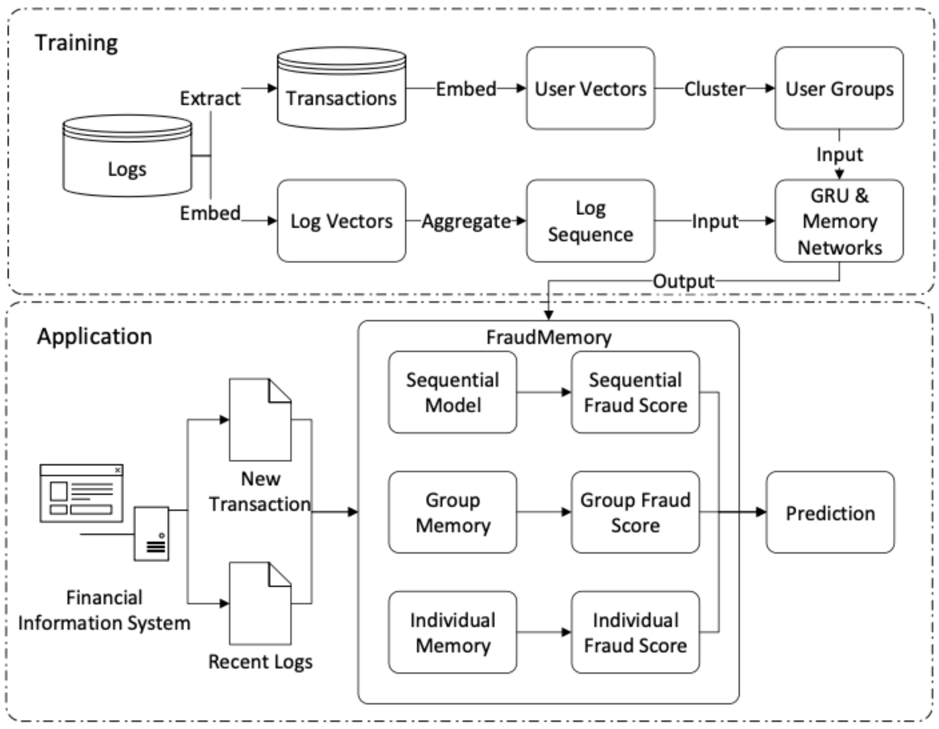


Figure 2.2: FraudMemory training and application, source [9]

A similar score is also calculated on the group history. The group history is the one obtained in the previous step, where users with similar behavior are aggregated through a clustering algorithm. This second score is helpful in cases where the individual history of the user is short and presents high variance between transactions. Finally, these two scores are concatenated with the sequential representation h_t^u to obtain a final anomaly score. This comprehensive score indicates the transaction's deviation from the expected user behavior. Furthermore, the memory networks are updated if the sequential representation produces significant change compared to the previous transaction history. This ensures the model can adapt to evolving patterns and effectively capture concept drift. The model training and prediction processes are shown in figure 2.2.

Therefore, the model presented by the authors is highly explainable, as the motivations that lead the classifier to flag a transaction as fraud are explicitly modeled, and can deal with concept drift.

However, the paper doesn't present the data and code in their experiments, making it less transparent.

Unsupervised methods

If we assume fraudulent and non-fraudulent payments originate from different distributions, **unsupervised anomaly detection** methods can be employed to identify fraudulent payments.

Several papers, such as *"A Linear Method for Deviation Detection in Large Databases"* by Arning et al.[18], *"Towards parameter-free data mining"* by Keogh et al. [19], and *"Graph-based anomaly detection"* by Noble et al. [20], explore the use of **information theory** in the anomaly detection task.

These techniques define the information content of a given dataset using information-theoretic measures such as Kolmogorov Complexity and rely on the assumption that anomalies induce irregularities in the information content of the dataset. In essence, anomalies add entropy to the dataset's information content. The goal in this framework is to maximize the entropy added by the minimum possible number of anomalous instances, thereby distinguishing them from the normal instances.

Recent advancements in anomaly detection have also explored the use of **self-supervised learning**. Self-supervised learning involves training a model on auxiliary tasks that exploit the inherent structure of the data itself as a learning signal. Instead of relying on human-provided labels, the model is trained to perform proxy tasks that leverage data attributes that can be inferred without any labels. By doing so, the model learns representations that capture the underlying factors of variation within the data, which can be beneficial for detecting anomalies.

Two recent papers, "Classification-Based Anomaly Detection for General Data (GOAD)" by Bergman and Hoshen in 2020 and "Neural Transformation Learning for Deep Anomaly Detection Beyond Images" by Qiu et al. at ICML 2021, propose methods that utilize transformations using self-supervision to detect anomalies.

Classification-Based Anomaly Detection for General Data (GOAD) - Bergman and Hoshen, 2020

Firstly, **Bergman and Hoshen**(2020)[21] propose to transform each data sample in the training set into M subspaces. Note that the training set is exclusively composed of normal data. A feature space is learned such that intra-class separation is small (i.e. data samples transformed by the same transformation are close to each other in the feature space) while inter-class separation is large (i.e. data samples transformed by different transformations are distant from each other in the feature space). **The authors' key idea** is that the **distance to the cluster center in the feature space is correlated with the likelihood of anomaly**, thus can be used as a proxy to determine whether a point is abnormal. Formally, authors formulate the hypothesis that all normal points lie in a subspace X while anomalous points do not:

$$\forall (x, y) \in D_n : y = 0, x \in X \subset \mathbb{R}^d, \forall (x, y) \in D_n : y = 1, x \notin X \subset \mathbb{R}^d$$

Authors propose to perform M transformations:

$$X \xrightarrow{\text{M Transformations}} \begin{matrix} X_1 \\ \vdots \\ X_M \end{matrix}$$

The idea here is that for all points $x \in X \subset X$, each transformed sample x_m should lie in a subspace $X_m \subset X$.

$$\forall x \in X \sim \underbrace{T(x, 1)}_{\text{Transfo.}}, \ldots, T(x, M)$$

where

$$T(x, m) \in X_m$$

Transformations are set to belong to the class of affine transformations, which works for general data types and especially tabular data types

$$T(x, m) = W_m x + b_m$$

where W_m,b_m are sampled from a random distribution. A feature extractor $f : X \to Z$, in the form of a neural network, is trained to map each data point in $X \subset R_d$ to a subspace Z. Similarly to SVDD[16], each subspace X_m is mapped to a feature space $f(x)|x \in X_m$ in a hyper-sphere with center c_m. Through the optimization of the **center triplet loss**, f undergoes training to acquire clustering capabilities that emphasize reduced variation within classes and enhanced variation between classes.

$$\mathbf{L} = \mathbb{E}_{x \sim \mathbb{D}_n^{train}} \max\{||f(T(x, m) - c_m||^2 + s + \min_{m \neq m'} ||f(T(x, m) - c_{m'}||^2, 0\}$$

where s is a margin regularizing the distance between clusters.

The **classification task** is considered by authors as an **auxiliary task** for creating an anomaly score, which can be used to identify anomalous points. A classifier is trained to predict the transformation $m \in 1, \ldots, M$ applied to $T(x, m)$, aiming to detect the specific transformation applied to obtain $T(x, m)$ from x. In this process, if the transformations T are one-on-one and x does not belong to X, the transformed samples $T(x, m)$ will not belong to the appropriate subspace $X \notin X_m$. As a result, the estimated probabilities $P(m|T(x, m))$ will be low for these anomalous points, because the classifier is exclusively trained on points within X. This implies that the anomalous points will have a high anomaly score. For example, the anomaly score is defined as

$$Score(x) = -\log \mathbb{P}(x \in X) = -\sum_{m} \log \tilde{\mathbb{P}}(T(x,m) \in X_m) = -\sum_{m} \log \tilde{\mathbb{P}}(m|T(x,m))$$

where $\tilde{P}(.)$ is defined in their paper. Higher scores indicate a more anomalous data sample.

Neural Transformation Learning for Deep Anomaly Detection Beyond Images, Qiu et al. (ICML, 2021)

Authors propose an approach relatively similar to the paper presented above in the sense that it relies on **transformations**. In NeuTraL AD[22], authors propose to learn simultaneously the **feature extractor and the transformations applied to data samples**, instead of simple affine transformations, as shown in figure 2.3. Moreover, their approach relies on a **contrastive loss** and does not involve any classification tasks as in GOAD[21] to construct the anomaly score. In a nutshell, the applied transformation takes the form of a neural network and is trained such that:

- Encoded transformed and untransformed data are not distant from each other in the embedding space.
- Encoded transformed data from different transformations should be distant from each other in the embedding space.

Formally, the approach considers M transformations denoted as $T : X \cdot 1, \dots, M \mapsto X$, which are represented by neural networks $\phi(., \theta_m)$ with learnable parameters θ_m. Simultaneously, an encoder or feature extractor $f_\phi : X \mapsto Z$ is learned in a joint manner. The transformations should satisfy the following criteria:

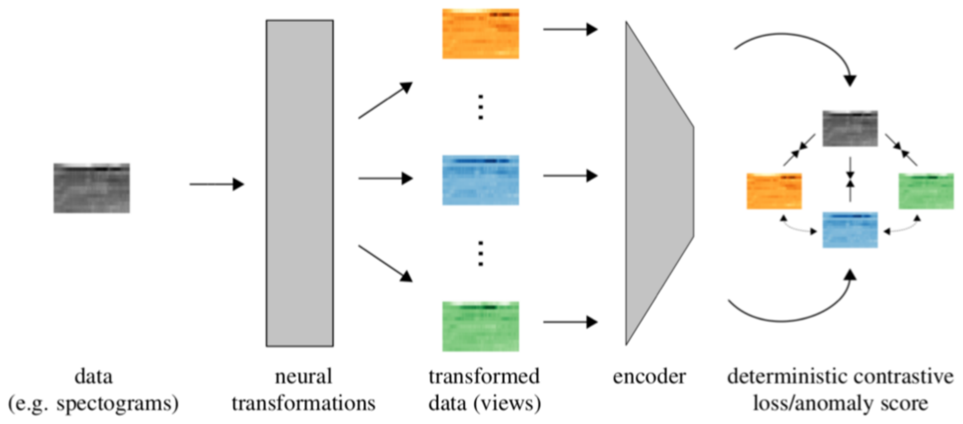


Figure 2.3: Both encoder and transformations are learned simultaneously, source [22]

1. The produced views should retain relevant semantic information shared with the original data: $\forall m \in 1, \dots, M$, the transformed sample $x_m := T(x, m)$ and the original sample x should be close to each other in the embedding space Z. In other words, the distance measure $d(f_\phi(x_m), f_\phi(x))$ between their respective embeddings f_ϕ should be low.

2. The transformations should generate diverse views of each sample. This implies that for any two distinct transformations $l, m | (l \neq m)$, where l, $m \in 1, \dots M$, the distance measure $d(f_\phi(x_l), f\phi(x_m))$ between their corresponding embeddings f_ϕ should be high.

$$h(x_m, x_l) = \exp(\frac{sim(f_\phi(T(x,l)), f_\phi(T(x,m)}{\tau})$$

.

where $sim(.,.)$ is the cosine similarity, and τ is a temperature parameter. $h(.,.)$ correlates with the similarity of two vectors in the embedding space. Based on the formulated objectives, you want to maximize $h(x_m, x)$ $\forall m$ while minimizing $h(x_l, x_m) \forall l \neq m$. Thus, the parameters of NeuTraL AD, $\phi, \theta_1, \dots, \theta_m$ are optimized by minimizing the following *deterministic contrastive* loss:

$$\mathbb{L} = \mathbb{E}_{x \sim D_n^{train}} - [\sum_{m=1}^{M} \log \frac{h(x, x_m)}{h(x, x_m) + \sum_{l \neq m} h(x_m, x_l)}]$$

.

Numerator encourages objective (i) and denominator pushes towards (ii). The loss can itself be used as an anomaly score.

Chapter

3

Proposed Model

In this section, we present the proposed model. The section is structured as follows:

- Conceptual Overview: We start with a brief conceptual overview to introduce the main ideas behind the model and its purpose.
- NLP and its Relation to the Model: In the second part, we delve into the details of Natural Language Processing (NLP) and explore how it is intricately linked to our model. We explain the fundamental principles and techniques of NLP and highlight their relevance to our fraud detection system.
- Overview of Gradient Boosting Tree Architecture: Finally, we provide an overview of the Gradient Boosting Tree (GBT) architecture. We explain its key features, advantages, and limitations. Understanding GBT will serve as a basis for comparing its performance with our proposed model and evaluating the advancements achieved.

Through this section, we aim to provide a comprehensive understanding of the model, its underlying concepts, and its application to the fraud detection task at hand.

3.1 Model Overview

The model draws significant inspiration from the FraudMemory model [2.2]. The proposed model can be divided into **two distinct phases**:

1. **In the first phase**, which is a **preliminary phase**, the **attributes** undergo a **processing** step through an *embedding layer*. The embedding layer is constructed using the Continuous Bag of Attributes (CBOA) dense embedding technique. This technique enables the projection of the **attributes** into a *lower-dimensional space*. By

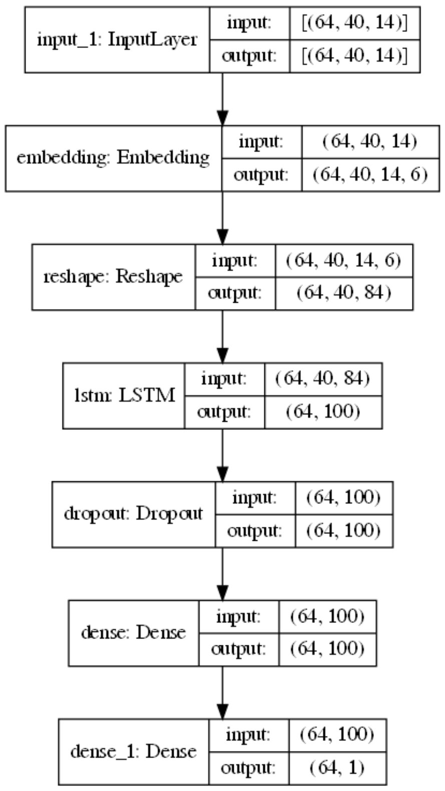


Figure 3.1: The Model Architecture

doing so, the attributes are **quantified** and represented as **real-valued vectors** that capture their *semantic relationships* with **other attributes**.

2. The **second phase** frames the **model architecture**.
   The model compounds a **LSTM layer** and two **Dense layers** on top. The **LSTM** is fed with **sequences** of **vectorized attributes** and extracts **temporal patterns**. The two **Dense layers**, instead, implement the actual **classifier**, which determines whether a transaction is fraudulent or not.

The model architecture shown in figure 3.1.

CBOA and time series models like **LSTMs** belong to the **NLP methodology**. In this context, sequences of transactions can be seen as analogous to *words in sentences*. The goal of the model is to capture a sense of coherence within the sequence, as well as to understand the flow and coherence of speech by evaluating words in context. As the model processes transaction after transaction, it gradually becomes better at assessing the coherence and compatibility of each subsequent transaction with the previous ones, similar to understanding the coherence of words in a sentence.

To start, let us delve into the definition of NLP.

3.2 NLP

Natural Language Processing or **NLP** is a field of Artificial Intelligence that allows machines to read, understand and derive meaning from *human languages*[23]. However, before machines can understand human language, it is crucial to transform *words* in these languages into a format that machines can process effectively. This is achieved through the use of **word embeddings**, which encode words as numerical vectors.

Word embeddings

Word embeddings are techniques that map a word (or phrase) from its original high-dimensional input space (the body of all words) to a lower-dimensional numerical vector space[24]. They represent words using continuous vectors. These vectors are obtained through *shallow neural networks*. *A shallow neural network*, as opposite of a *deep neural network*, is a network with a single hidden layer. This hidden layer is the word embedding we are looking for. The two main shallow models, shown in figure 3.2, used now-a-days in NLP are:

- **SkipGram.** The distributed representation of the input word is used to predict the context

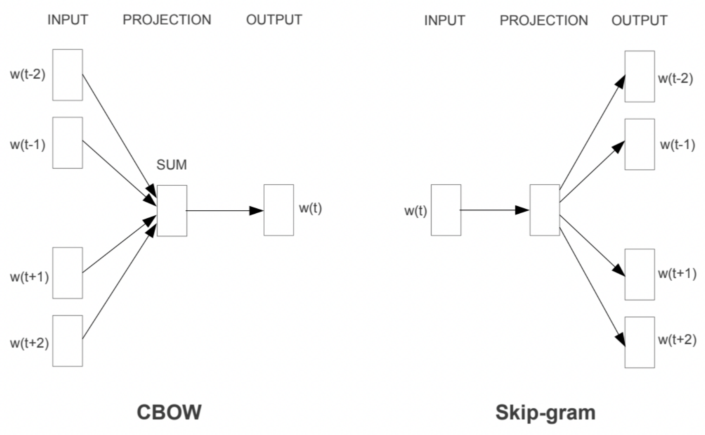


Figure 3.2: Graphical representation of the CBOW model and Skip-gram model, w_t represents word at time step t, source [25]

- **CBOW.**[1] The distributed representations of context are combined to predict the word in the middle.

In the two definitions above, which quote the original paper[25], the expression *distributed representation of the input word* is used. It underlines that each word is represented by its context, that is, by the other words that frequently appear along with our target word. This enables us to capture the semantics of words.

To make the concept clearer, it can be helpful to quickly go through the two models at hand.
In a preliminary stage, all the words in the dictionary, that is, all the words that can appear in our sentences, are encoded using **one-hot encoding**.

One-hot encoding is the simplest way to represent words in a numerical format. A vector that is as long as the number of words in the dictionary is assigned to each word. This vector is composed only of zeros and a single one. The position of the one within the vector varies for each word. To illustrate, let's consider a dictionary containing four words. Here is an example of how one-hot encoding represents these words:

- The 1000
- Red 0100
- Fox 0010
- Jumps 0001

In each vector, only one element is set to one, indicating the presence of a specific word, while all other elements are zero. The use of one-hot encoding is limited due to the "Curse of Dimensionality." This term refers to the increased computational burden associated with larger dictionaries, as the length of the encoding becomes longer. As a result, the network's training process becomes more demanding.

In a second stage, these words will be fed as input to a shallow network. The **Skip-gram model**, given a word, called center word, as input, will output a probability distribution over all the words in the dictionary. The higher the probability cast to a word, the higher the probability to find it in the context of input words. Context words are defined through a window-size, a hyper-parameter of the model that specifies the number of words before and after the center word. For example, if we have the sentence: *"The red fox jumps over the fence"* we will take

$$w_t : \text{"fox"}$$

as center word, and we set a window-size of 4, the context words will be

$$w_{t-2} : \text{"The"}, w_{t-1} : \text{"red"}, w_{t+1} : \text{"jumps"}, w_{t+2} : \text{"over"}.$$

[1]Continuous Bag of Words

In contrast, the CBOW model takes the context words as input and predicts the center word. This is the model we will utilize to encode our transactions. Now, let's delve into the training process of the CBOW model.

The training input will be composed of all the pairs (*target, context*). The example above, for instance, would produce the pairs: (*fox, the*), (*fox, red*), (*fox, jumps*), (*fox, over*). The objective function to optimize is:

$$\text{minimize} - \log \hat{P}(w_t | w_{t-n/2}..w_{t-1}w_{t+1}...w_{t+n/2})$$

So we will have as input the context words, which will be projected in the hidden layer, then the average sum will represent our target word. The interesting thing about this kind of word embedding is that they can encapsulate meaning. This means similar words will be close in the projection space, forming clusters. We will have clusters of countries, for example, or of cities. Moreover, basic operations are possible on them. For instance:

$$\text{France - Paris + Poland = Warsaw}$$

and

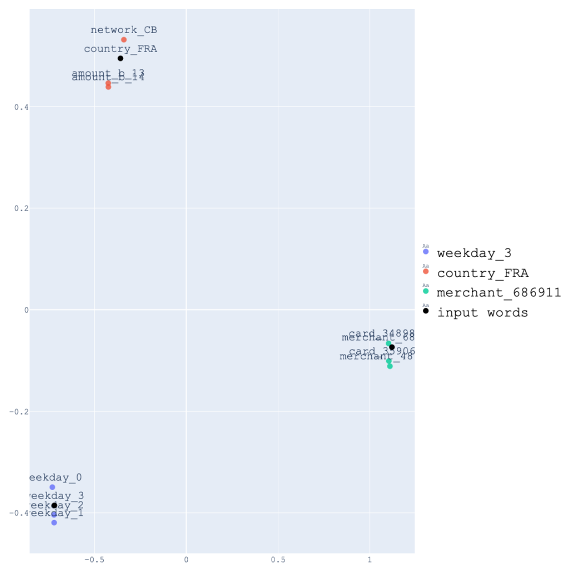
$$\text{King - Man + Woman = Queen}$$

As mentioned, we will apply the **CBOW** model to our transactions. This new model is called **CBOA**, which stands for **Continuous Bag Of Attributes**. This new name is due to the fact that we are no longer dealing with words, but with attributes of transactions. The **CBOA** algorithm projects the **attributes** in a *lower dimensional space*. This projection **quantifies** the attributes and expresses them through **real-valued vectors**. We end up with a **dictionary** that pair the attributes to **6-dimentional vectors**.

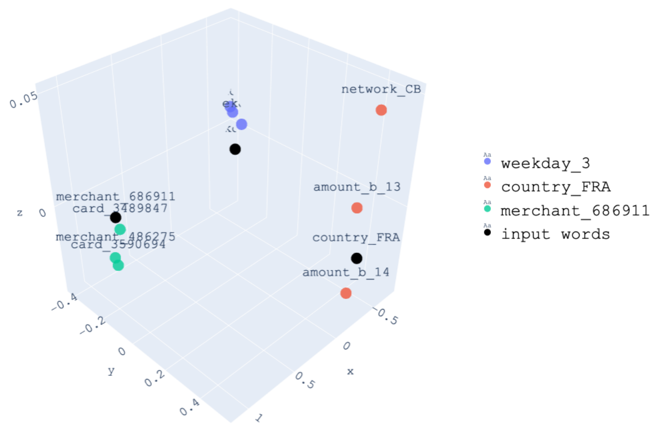
To visualize the attributes in their new subspace, we use common **dimensionality reduction techniques** such as **Principal Component Analysis** (PCA) and **T-distributed Stochastic Neighbor Embedding** (t-SNE). With either method, the **distance between words** expresses the **similarity between them**.

Let's begin with the concept of **PCA**.
PCA is a linear dimension reduction technique that aims to map high-dimensional data onto a lower-dimensional space while maximizing the variance of the data.

In Figure 3.3, we can observe the representation of words. The black words represent the input words, while the colored words represent the three most similar words to the input words. For instance, we notice that a word like "merchant" is grouped together with other similar merchants, indicating their semantic similarity. Similarly, the neighborhood of a word like "country" (e.g., France) consists of attributes related to amounts and networks, highlighting their contextual relevance.



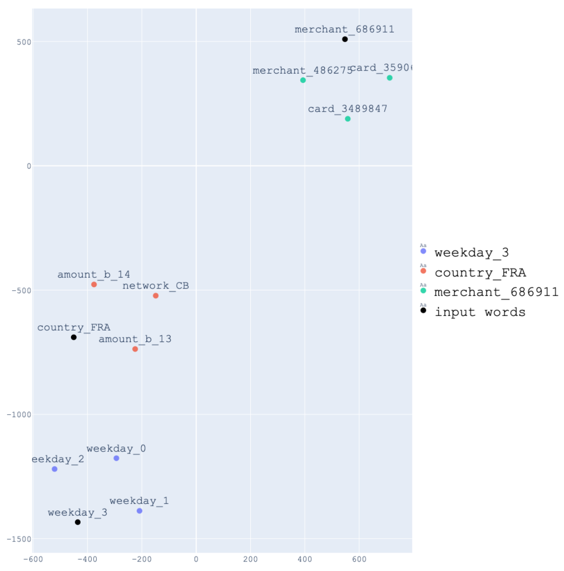
(a) Vectors visualized with PCA in 2D



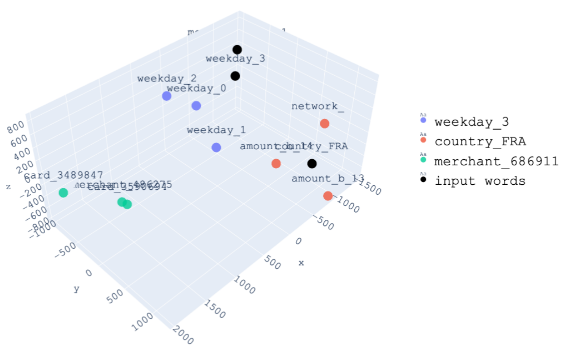
(b) Vectors visualized with PCA in 3D

Figure 3.3: Vectors visualized with PCA

Let’s now inspect the data using the **t-SNE** technique.
With **t-SNE**, the algorithm calculates the similarity in both high dimensional space and low dimensional space. Next, the **similarity difference in both spaces is minimized** using an optimization method, for example gradient descend method. Also in this case, in figure 3.4, black words are the input word, while the colored ones are the three most similar words to the input words. We obtain similar results.



(a) Vectors visualized with T-SNE in 2D



(b) Vectors visualized with T-SNE in 3D

Figure 3.4: Vectors visualized with T-SNE

Recurrency and Memory

After obtaining a meaningful representation of words and attributes using Continuous Bag-of-Attributes (CBOA), the next step is to choose a network that can effectively handle sequential data. This is necessary, because the meaning and context of words in a sentence or the patterns in bank transaction histories depend on the information from previous elements.

As stated by Colah[26] in his blog post, "as you read this essay, you understand each word based on your understanding of previous words." This sequential nature applies to both language processing and analyzing transaction histories. To predict the next word in a sentence or determine whether a subsequent transaction will be a fraud, machines need to consider the temporal aspect of the data.

To face this task, Recurrent Neural Networks (RNNs) is one of the first network introduced. RNNs are designed to capture the relationship between inputs and outputs at each time-step, similar to a standalone feed-forward neural network. However, in RNNs, each time-step network is connected to the following one, creating a recurrent structure, as depicted in Figure 3.5. This recurrent connectivity enables the network to maintain memory of previous inputs and learn long-term dependencies.

By utilizing RNNs, our model can effectively model the sequential nature of the data, whether in sentences or transaction histories. This enables the system to make predictions and decisions based on the temporal patterns and dependencies present in the data.

After obtaining a meaningful representation of words and attributes using Continuous Bag-of-Attributes (CBOA), the next step is to choose a network that can effectively handle sequential data. This is necessary, because the meaning and context of words in a sentence or the patterns in bank transaction histories depend on the information from previous elements.

This way, the output is defined as:

$$\hat{y}_t = f(x_t, h_{t-1})$$

It is then dependent on the current input and hidden state of the network at the previous time-steps. While the hidden state of such a network at time t takes as input the current input and activation from hidden units at time $t-1$, defining the relationship:

$$h_t = f_w(x_t, h_{t-1})$$

It is important to notice that the function

$$f_w(...)$$

is **parametrized**, and that the **same function**, along with **its weights**, is used for all time-steps. This notion of **state** introduced by h_t is what makes **RNNs** remember previous inputs and their outputs. In figure 3.6, we can observe the internal state of a **RNN** with f_w modeled as the tanh function.

It appears intuitive that now, when performing back propagation during training with **RNNs**, we back propagate the gradients **not only through the network**, as with classical feed-forward networks, **but also through different time-steps**. This is what is called *Back Propagation Through Time*. Due to these numerous multiplications of weights matrices, **RNNs** are prone to the **vanishing gradient** and **exploding gradient** problem. This means that during training, the gradient tends to disappear or boost, leading to an inability of the network to retain information through longer sequences.

Long Short Time Memory network was conceived to deal with this problem. First introduced in 1997 by Sepp Hochreiter and Jürgen Schmidhuber[27], this network presents a gated memory cell, able to control the information that flows inside and outside of them.

Four gates[28], with respective weight matrices W_f, W_i, W_c and W_o, regulate their behavior, as shown in figure 3.7:

1. **Forget Gate**. This gate decides which information in the cell is no longer relevant. The following equation rules it:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

   The $\mathrm{sigmoid}$ function calibrates, based on h_{t-1} and x_t, which degree of information we want to retain for each number of the previous state cell C_{t-1}. The $\mathrm{sigmoid}$ function and the tanh function will perform similar tasks also in the other gates.

2. **Input Gate**. This gate controls the input information of the cell.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

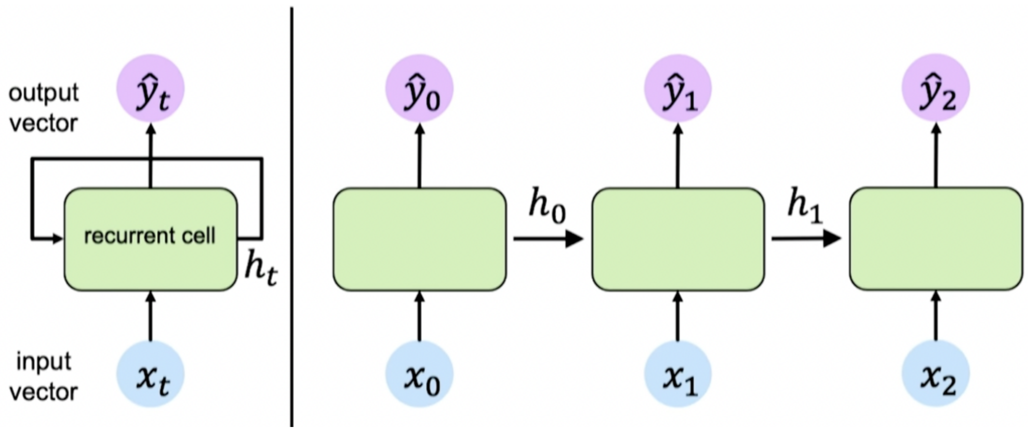


Figure 3.5: RNN recurrency step, source: *Introduction to Deep Learning, MIT*

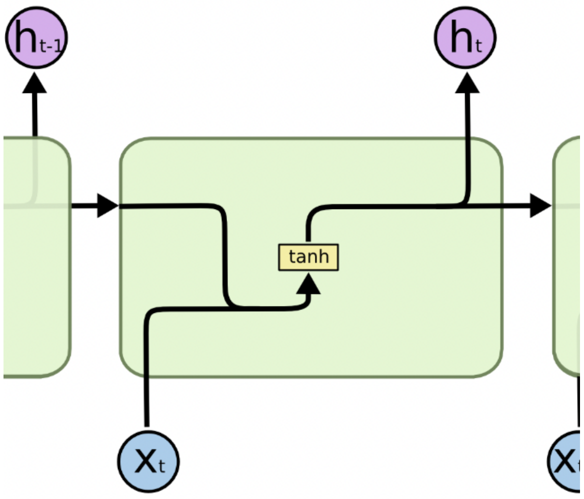


Figure 3.6: RNN internal state, source: *Understanding LSTM Networks, colah's blog*

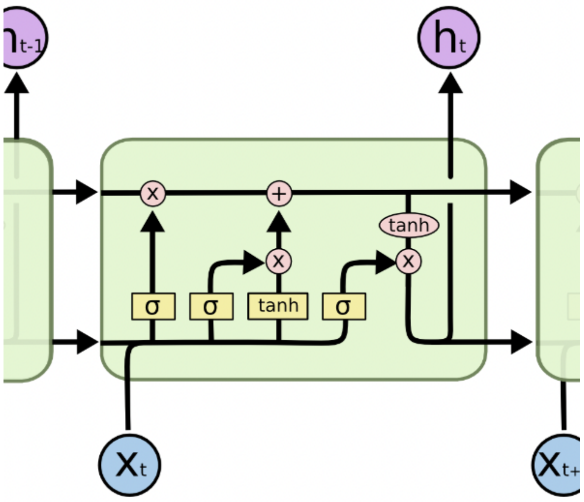


Figure 3.7: LSTM internal state, source: *Understanding LSTM Networks, colah's blog*

3. **Input Modulation Gate**. This gate proposes candidates substitutions for the cell memory.

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

4. **Output Gate**. This gate controls the output information of the cell.

$$o_t = \tanh(W_o \cdot [h_{t-1}, x_t] + b_o)$$

Then, the cell state C_t is updated as follows:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t}$$

The forget gate layer f_t multiplies the previous cell state C_{t-1} to decide what to keep or not, and the result is added to the product of the input layer i_t and $\tilde{C_t}$, which modules what to add. Finally, the output of the cell is:

$$h_t = o_t * \sigma(C_{t-1})$$

which consists of the modulation of the output information performed by the output layer o_t multiplied by the sigmoid of the state cell.

In conclusion, the **LSTM** network allows us to correlate the current transaction to the previous ones, even in long sequences.

3.3 Comparative insight on GBT approach

As mentioned before, the current adopted model is a **GBT** enhanced with **feature engineering techniques** for time-series classification. Feature engineering relies on experts' knowledge to be properly design, though **human intervention** plays a crucial role here. On the other hand, our model relies utterly on *artificial intelligence* to comply with the classification.

It's of clear interest to **compare the performances** between both approaches.

In this section, we will provide an overview of the **second approach**. We are presenting only the *aspects* functional to the **comparison**. More insight on **GBTs** can be found, for instance, in the following papers: *Thoughts on Hypothesis Boosting, Kearns et al.*[29], *Arcing the edge, Breiman et al.*[30] and *Stochastic gradient boosting, Friedman et al.*[31].

3.4 Feature engineering

*"**Feature engineering** refers to the process of using domain knowledge to select and transform the most relevant variables from raw data when creating a predictive model using machine learning or statistical modeling."*[32]

When it comes to **time-series**, features engineering is particularly important when the model used don't make use of recurrency. Features engineering encapsulates **time** in **new attributes** in three different ways[33]:

1. **Date Time Features**. These are features that refer to the temporal domain of the *time step itself* for each observation.
   These features are typically used to extract periodical or cyclic patterns in the data. They usually consist of features such as *Month*, *Day*, *Hour* and flags like *Weekday* or *Business hours*.

2. **Lag Features**. These features consist in values taken from *prior time steps*.

   Lag features put in relation current time-steps with previous ones. In this way, it's possible to transform a forecast time-series problem into a supervised problem.

   We can easily extend the time frame we consider with the **sliding window method**: this way the model captures correlation between values registered at multiple time-steps. A **hyperparameter** of interest when dealing with the sliding window method is the **window size**, which determines how many prior time steps are to be evaluated.

3. **Window Features**. These features consist in **summary of values** over a fixed window of prior time steps. These features provide the classifier features based on **summary statistics** across the values included in the sliding window.

The prior approach mainly exploited this last kind of features: they computed *rolling sums, rolling averages and standard deviations* over various periods to enhance each data point. These features provide the classifier **values of reference** to decide whether a new data instance is anomalous.

3.5 GBT

Gradient Boosting is a technique used to build predictive models and classifiers. **Boosting** relies on the idea of constructing an algorithm which can convert a weak learner into a better one[2].

The first actual **application** of boosting is identified with the so-called **AdaBoost algorithm**[3]. AdaBoost algorithm starts with training a **decision tree** on a equally weighted set of the observations. At the end of the first iteration, the decision tree will have successfully classified some samples and misclassified others. The *misclassified observations* will be assigned *greater weights*. The second tree is therefore grown on this weighted data. The new model will be composed of the *two trees*. This procedure is repeated until a satisfying result is obtained. Eventually, predictions are made by *majority vote* of the weak learners' predictions, weighted by their individual accuracy.

Gradient Boosting Trees are a *generalization of AdaBoost*. They are obtained reassessing **AdaBoost** in a statistical framework[36]. This framework rephrases the problem as a *numerical optimization problem* with the aim of minimizing the objective of the model through a **gradient descent-like procedure** by **sequentially adding weak learners**.

The implementation adopted in this thesis is Light Gradient Boosted Machine (LightGBM), an open-source library.

[2]A weak hypothesis or weak learner is defined as one whose performance is at least slightly better than random chance. All the theory behind weak learners and more can be investigated here *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World, Arney et al.* [34]

[3]A decision-theoretic generalization of on-line learning and an application to boosting[35]

Chapter

4 Dataset

In this section, we will **present** and then **inspect** the provided dataset. As explained below, the original dataset has been modified in various ways: from *dropping attributes* useless to the learning process to *change the data format* to boost performances.
In this instance, we also provide an overview of the **derived features**, result of the *feature engineering* process, used along with the **GBT**, as hinted in the previous section. [2.2]
Finally, **the last subsection**, *Consideration about the dataset*, **pinpoints the motivations** regarding the chosen model.

4.1 Original Dataset

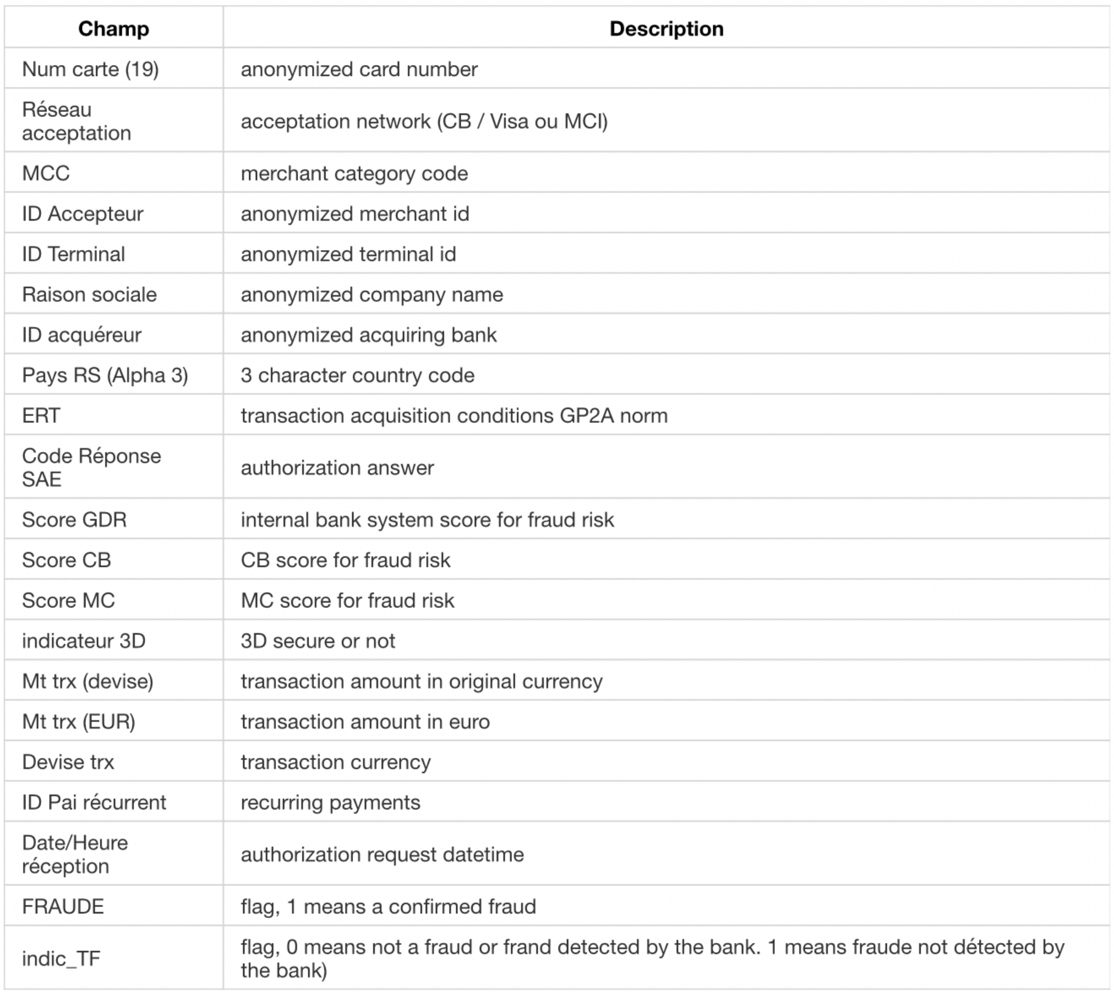
The full **original dataset** is shown in figure 4.1. It comprises **100 millions transactions** over **5 months** of online payments, labeled with frauds. We have been provided with eight **.csv** files of **10 millions transactions each** from this dataset. Two **.csv** files are kept apart for *testing purposes.*

Only *some* of these **fields** are useful for the machine learning process. Therefore, under the advice of **bank experts**, fields confirmed useless have been removed, as well as the **NaN** and the **outliers** values.

4.2 Optimized Dataset

Loading the original data under Pandas *DataFrame* takes more than **5 minutes** on a *Core i5 3.8 GHz* computer. The DataFrame size exceed **19.5 GB RAM.** To provide a more compact, memory efficient DataFrame, to simplify feature engineering and accelerate machine learning, the following **transformations and optimizations** have been made:

- Transform every datetime from text to `datetime64`[ns]



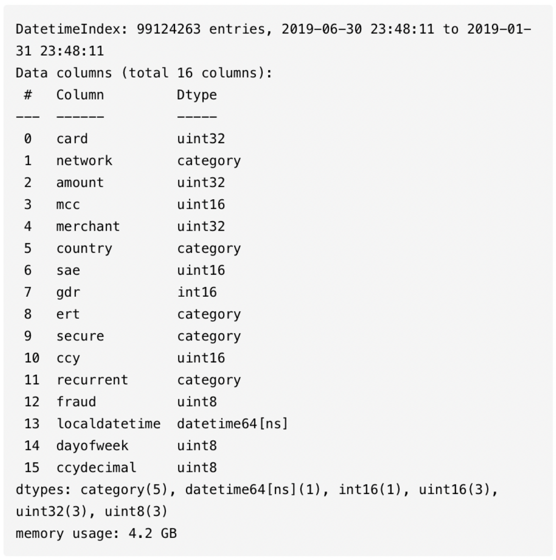
Champ	Description
Num carte (19)	anonymized card number
Réseau acceptation	acceptation network (CB / Visa ou MCI)
MCC	merchant category code
ID Accepteur	anonymized merchant id
ID Terminal	anonymized terminal id
Raison sociale	anonymized company name
ID acquéreur	anonymized acquiring bank
Pays RS (Alpha 3)	3 character country code
ERT	transaction acquisition conditions GP2A norm
Code Réponse SAE	authorization answer
Score GDR	internal bank system score for fraud risk
Score CB	CB score for fraud risk
Score MC	MC score for fraud risk
indicateur 3D	3D secure or not
Mt trx (devise)	transaction amount in original currency
Mt trx (EUR)	transaction amount in euro
Devise trx	transaction currency
ID Pai récurrent	recurring payments
Date/Heure réception	authorization request datetime
FRAUDE	flag, 1 means a confirmed fraud
indic_TF	flag, 0 means not a fraud or frand detected by the bank. 1 means fraude not détected by the bank)

Figure 4.1: Original Dataset Fields

- Set reception `datetime64` as `DateTimeIndex` [1]
- Transform `['Reseau acceptation','Pays RS (Alpha 3)','ERT','indicateur 3D','ID Pai recurrent']` as pandas *Categorical* objects. This enables to encode them in a **more compact** way with **numerical internal values** of needed size only[2] while keeping the text values intact for making easier features engineering
- drop `['Mois','Type Trx','ID Terminal','Raison sociale','ID acquereur','Score CB','Score MC', 'trx_id','indic_TF']`, thanks to bank experts advise
- drop rows with transactions that accounts for amounts over **one million of euros**

[1] make it ready for rolling windows features computing

[2] `int8` for instance



```
DatetimeIndex: 99124263 entries, 2019-06-30 23:48:11 to 2019-01-
31 23:48:11
Data columns (total 16 columns):
 #   Column         Dtype
---  ------         -----
 0   card           uint32
 1   network        category
 2   amount         uint32
 3   mcc            uint16
 4   merchant       uint32
 5   country        category
 6   sae            uint16
 7   gdr            int16
 8   ert            category
 9   secure         category
 10  ccy            uint16
 11  recurrent      category
 12  fraud          uint8
 13  localdatetime  datetime64[ns]
 14  dayofweek      uint8
 15  ccydecimal     uint8
dtypes: category(5), datetime64[ns](1), int16(1), uint16(3),
uint32(3), uint8(3)
memory usage: 4.2 GB
```

Figure 4.2: Optimized Dataset

or any amount marked as **NaN** .[3]

- add dayofweek column
- extract the *decimal part* of the transaction amount in currency, multiply it *by 100* and store *its integer value* to ccydecimal feature. Finally drop Mt trx [4]
- convert every *integer features* into their *necessary type* from uint8 to uint32
- for Score GDR , transform NaN into **-1** and convert it to int16
- multiply Mt trx (EUR) by 100 and convert it into uint32

After these optimizations, the dataframe size dropped ***from 19.5 GB to 4.2 GB***. The new dataset is shown in figure 4.2. Moreover, **Very small values** in the dataset are assimilated **to zero**, while **very large values** are peaked **to the maximum** of the 32-bits integer encoding used in order **to save space** while preserving a **good precision**. This last property is particularly desired when, to return the data to its original values, we divide them by 1000.

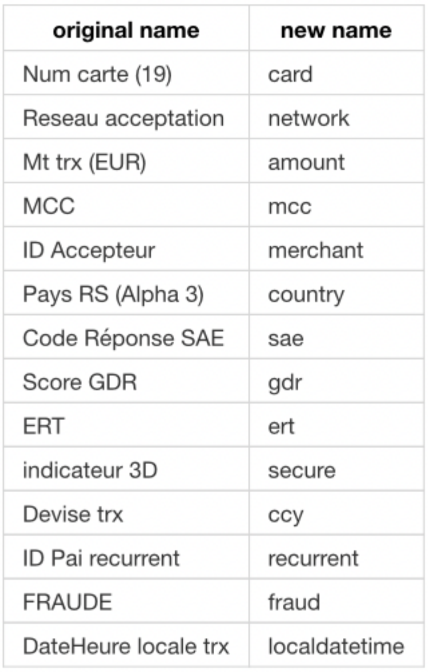
Finally, columns are renamed according to the table 4.3.

Parquet Storage

Parquet is selected as data format storage. Parquet[37] is a data format storage that uses columnar memory format. It is designed to optimize data storage and retrieval.

[3]Their respective proportion of frauds is **20** and **40 times lower** than the global proportion, meaning NaN amounts is **not** a value **related to an increased presence of fraud**

[4]devise



original name	new name
Num carte (19)	card
Reseau acceptation	network
Mt trx (EUR)	amount
MCC	mcc
ID Accepteur	merchant
Pays RS (Alpha 3)	country
Code Réponse SAE	sae
Score GDR	gdr
ERT	ert
indicateur 3D	secure
Devise trx	ccy
ID Pai recurrent	recurrent
FRAUDE	fraud
DateHeure locale trx	localdatetime

Figure 4.3: Dataset with renamed columns

This was done after testing against other formats: **.csv**, **SQLite** and **Feather.**

While the original **.csv** file took up to **5 minutes** to be loaded, the **SQLite** file took **more than 15 minutes**[5], finally the **parquet file** took about **1 minute**. The **Feather** file, on the other hand, has *read/write* performances in the same range of efficiency, but the files are *several times bigger*.
The **Parquet** data format enables:

- fast writing and loading
- natively compressed file[6]
- preserved types including *Categorical* and *DateTimeIndex*
- loading only column selection
- Dataset file written with *pyarrow.*

Moreover, the **parquet optimized dataset** took around **24 seconds** for loading.

[5]stopped before the end during testing!
[6]**1.7 GB** for the optimized dataset

4.3 Derived Features

For each **card**, **merchant** and **pair (card,merchant)**, the following **derived features** are computed:

- **mean** and **standard deviation** (std) of the amount spent. They are converted **to integers** as they are. Given their ranges, there is **no need** of multiplying them by a factor to preserve information.
- **cumulative standard deviation** (nbstd) of the amount spent. It indicates how much of **std** is represented by the current amount, compared to the **std** calculated over previous amounts. When the value of the first samples of each group of attributes is either **NaN** (*std=0, mean=0*) or **np.inf** (*std=0, mean>0*), it is replaced with **-1**. Moreover, the **nbstd** is multiplied **by 1000** before being converted **to integer**. This preserves 91.27% of all data from the entire historical dataset, keeping it completely intact.
- **rolling sum of the amount spent**
- **count** of the total transactions over **one month**, **one week** and over **the whole history**
- **average number of hours** *-in the range 1 to 24 over a day period-* and **average minutes** between **two consecutive transactions**
- **average delay** in seconds after the *first to last*, *second to last* and *third to last* transaction, and for each of them also the **standard deviation** and the **cumulative standard deviation**
- **average period** in seconds between transactions, and its **standard deviation** over **the whole history**. This corresponds to the statistics on the delay of the first to last transaction.
- **difference of amounts** between two consecutive movements for the *first to last*, *second to last* and *third to last* transactions
- **difference of amounts** between short and long periods. For example, *1 h vs 1 week....*
- categories of **countries**. Fields such as "France\Entrager", "UE\NON UE" .... are added

Moreover:

- **different merchants** associated to **a card** over a period of **1 hour**
- **categories of merchants**. For example: *fund, transfer, company...*

- **merchants** appeared recently
- **merchants'** activities. For example: *several small transactions, ...*

4.4 Some consideration about the data

Data Format

We can point out **four main characteristics** based on the **format** of the data:

- **Temporal nature of data.**

  The data is constituted by **time-series**: each transaction is part of a sequence of transactions that develops through time.

- **Mixed nature of the attributes.**

  Either **categorical** and **continuous attributes represent a transaction.**

- **Contextual nature of the attributes.**

  It's interesting to notice that some attributes can be identified as so-called **contextual attributes**. These type of attributes determine the *context* of an instance.

  Some other attributes are, on the contrary, **behavioral attributes**. **Behavioral attributes** define the *non-contextual characteristics* of an instance.[7]

- **Availability of labels**.

  The availability of labels is important in the choice of the approach to use: *supervised*, *unsupervised* or *semi-supervised*.

Let's further inspect our data by extracting some statistics.

Data Statistics

The statistical analysis was conducted on the first 2 parquet files of data provided to us, for a total of **20 million of transactions.**

Two main considerations with respect to the statistics obtained by our inspection are to be done:

1. Only a small amount of the total payments is fraudulent:
   - 0.43 % of the total payments is fraudulent
   - 0.53 % of the total cards is fraudulent
   - frauds affect 3.54% of all merchants

[7]In other words, they are the properties of data that don't depend on *environmental factors.*

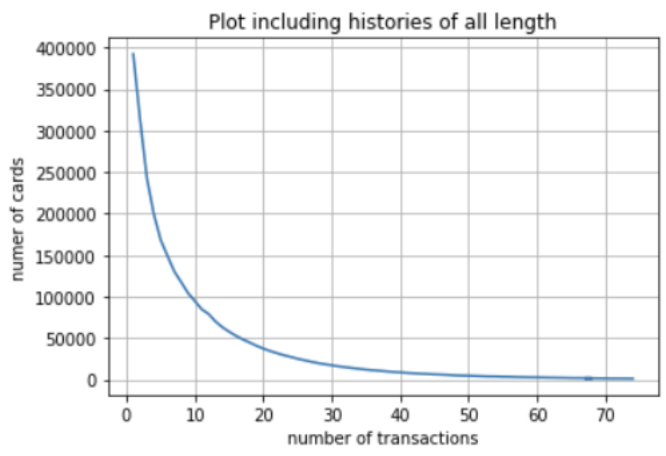


Figure 4.4: Graph with the number of cards in function of their histories' length, representing cards with histories up to 100 transactions

2. Each card has a **varying number** of transactions associated with it, and this number wanders in a **considerable range**:
   - We have an **average number** of card entries, from the first transaction to the last, of 7.78 entries. Some cards have only one transaction, while the maximum number of card entries accounts to 5693.
     This is visually shown in the bar graph 4.4 below.
   - the **95th percentile** number of card entries in the dataframe is 25 and the **median** is 4. This underlines that most cards have no more than 25 transactions, and that most of them have only 4.

These considerations naturally lead us to the type of model to implement and what experiments to perform.
Indeed, when deciding what solution you want to model, the starting point is always to look at the available data and see what is feasible and what is not.

The Model Architecture

The *architecture of the model* is determined by the **data format**:

- **Time-Series Data and Availability of Labels**
  With such data, we want to use a supervised algorithm that works well on *sequence-based* tasks with *long-term dependencies.* **LSTMs** are well established networks in the field that meet such criteria.
- **Contextual Attributes**.

  Attributes come within a *context*, and we want to exploit this information to *boost* classification. Moreover, LSTMs networks need **real values** as **input**. To either extract *contextual information* and provide *continuous and real-valued* attributes, we use a dense embedding technique, such as **CBOA**.

Elements of Complexity

The **statistical analysis** of the dataset, on the other hand, pinpoints some *elements of complexity* that deserve *particular attention*:

- **Highly Imbalanced Classes**

  We have a *vast majority* of samples in the dataset, which belongs to the class of **non-fraudulent transactions**. Datasets of this kind are more *delicate* to deal with, because they make it hard to understand whether the model captured the **underlying mechanism** that separates the classes or is simply **overfitting** the majority class. This issue is known as **Accuracy Paradox**: the model will achieve **high accuracy**, while actually performing poorly.

- **Time-Series of Varying Length**

  Time-series of *varying length* add complexity to our problem for **two reasons**:

  1. *Firstly*, as we've seen while inspecting the data, most cards have a **very short history**: some of them don't even account for more than a *single transaction*. We need to *discard* such instances, because they cannot be considered for *time-series classification*.
     But *how long* a time-series should be for the **LSTM layer** to extract **significant pattern** from it?
     Different experiments with this *hyper-parameter* have been undertaken and discussed in the *experiment section* [6.3].

  2. *Secondly*, since we are provided with histories of *variable length*, we want to use **customized window-sizes** to choose how many transactions to include in the computation, which length varies accordingly. This results in **manually computing** batches and epochs.

Chapter

5 Technical Implementation

The model was built using a **step-by-step approach.**
A step-by-step approach consists of building **several prototypes** of the model, each of them of **increasing complexity**.
This way, the main task gets chunked down in sequential, smaller ones, and we can **debug** one task before **integrating** the following one.

All prototypes are produced using **tensorflow** and **numpy** libraries.
Tensorflow is an open source software library that provides modules to support and facilitate the development of machine learning algorithms.
Numpy, on the other hand, is an open source software library that facilitates the manipulation and operation within matrices.
In the code, the **numpy** library appears as **np**.

synthetic0.py

The first prototype we produced is **synthetic0.py**.
In this prototype, we want to test the performance of the **LSTM network** on an **imbalanced dataset**.
However, in this initial stage, we use a **Gated Recurrent Unit**, or **GRU**.
They are a simpler version of **LSTM** networks, where the forget and update gate are combined. The hidden state and the cell state are also combined, and other minor changes occur[26]. Indeed, the dataset synthetically produced is small and empirical evidence[38] justifies the use of **GRUs** on fewer training data, as they train faster and perform better than **LSTMs**.
However, **LSTMs** can remember **longer sequences** than **GRUs** and outperform them in tasks that require to model long-distance relations.
For this reason, later on, the **GRU** layer will be substituted by a **LSTM** layer.

The *synthetic dataset* generated is an **oversimplified** version of the real one, and with a **slighter disparity**[1] between classes.
The dataset is represented only by **increasing series of numbers**, with target **1**, and **decreasing series of numbers**, with target **0**.
The increasing series are flagged as fraudulent, and the decreasing series as non-fraudulent. Moreover, each time series is of **fixed length**.

The following **snippet of code** shows the **database generation**.

```
def generate():
    #the fuction np.random.randn takes as parameters the dimensions of
        the array and return samples from the standard normal
        distribution.

    #nb_users indicates the number of users
    #ratio_fraud is the ratio between fraudolent and non fraudolent
        transactions

    d1 = np.random.randn(int(nb_users*ratio_fraud), ts_length)
    d1.sort(axis=1)
    d2 = np.random.randn(nb_users - int(nb_users*ratio_fraud), ts_length)
    d2.sort(axis=1)

    d2 = np.fliplr(d2)
    y1 = np.ones((int(nb_users*ratio_fraud)))
    y2 = np.zeros((nb_users - int(nb_users*ratio_fraud)))

    ds = np.concatenate((d1, d2), axis=0)
    y = np.concatenate((y1, y2), axis=0)
    perm = np.arange(ds.shape[0])
    np.random.shuffle(perm)
    ds = ds[perm]
    y = y[perm]
    return ds, y
```

In the following snippet, we can observe the **model**.
It is composed of the **GRU layer**, a **dropout layer** and **two dense layers** on top.
We recall here that the **GRU layer** is meant to catch **temporal patterns**, while the **dense layers** serves as **classifier**.
The **dropout layer** is included to help prevent **overfitting.** Noticeably, the number of units of the GRU layer is a fine tuned hyper-parameter.

```
    gru = keras.layers.GRU(units=100, activation='relu',
        return_sequences=False)(input_layer)
    dropout = Dropout(0.5)(gru)
    dense1 = Dense(100, activation='relu')(dropout)
    dense2 = Dense(1, activation='sigmoid')(dense1)
```

[1] 70% instead of 99.6%

synthetic1.py

In the second prototype, we incorporate time-series of **variable length**.
LSTM accepts only input of fixed size, so we perform a sort of *fake padding*[2] on the dataset produced before: for each sequence, all values from a random point on get substituted with **-1**.
A **mask layer**, which masks a sequence by using a mask value to skip time steps, allows the **LSTM** to ignore the values flagged as **-1**.
This way, we simulate a dataset of sequences of **different lengths**.

These are the lines of code that do that:

```
for i in range(len(ds)):
    j = random.randrange((len(ds[i])*2)//10, len(ds[i]))
    for k in range(j, len(ds[i])):
        ds[i, k] = -1.
```

The **input layer** is shaped as before

```
input_shape = (ts_length, 1)
```

Moreover, the **LSTM** substitutes the **GRU**, since the data produced are now more complex than before.

```
mask = keras.layers.Masking(mask_value=-1.0, dtype=float)(input_layer)
lstm = keras.layers.LSTM(units=max_window_size, return_sequences=False,
    dropout=0.5)(mask)
```

synthetic2.py

In **synthetic2.py** we integrate **rolling windows** on cards histories.
A **rolling window** is composed of a **window** that moves along the time axis. The **window** has a fixed size and contains many consecutive transactions.
The window moves, for each user history, step by step along the data, one transaction at the time, as shown in figure 5.1. The window size depends on the length of the card history. The rolling window method allows us to obtain smaller sequences from the original card history. This procedure provides more insight into the data at hand and contributes to balancing the dataset. The so obtained sequences are finally **padded** to make their length homogeneous and compliant to the longest one.
The **input layer** is now shaped like this:

```
input_shape = (max_window_size, 1)
```

[2]we call this *fake padding* because the actual procedure of padding consists in making a sequence longer, introducing placer values that will be ignored by the model.

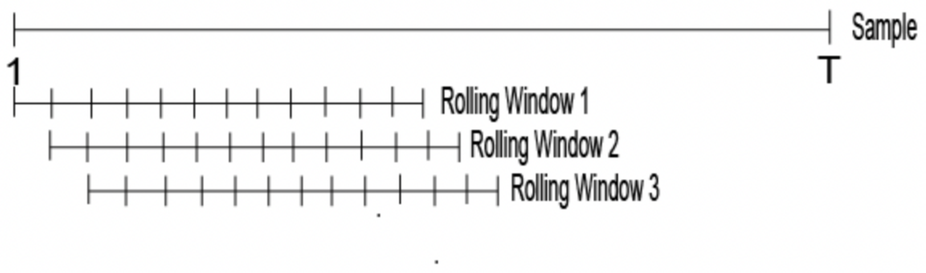


Figure 5.1: Rolling window example, source: *Mathworks*

We then concatenate all the new generated sequences one after the other to obtain an epoch. The **generator() function** handles these steps.

```
def generator(df, train=True):
users = df['user'].unique()
max_window_size, steps_per_epoch = compute_steps_per_epoch(df, train)
def generator(df, train, users, max_window_size):
    ts = []
    y = []
    count = 0
    for _ in range(epochs):
        for user in users:
            df_user = df[df['user'] == user]
             #customized window_size
            window_size = max(10, len(df_user) // 10)
            if train:
                #start training dataset
                start = 0
                 #end training dataset
                end = round((len(df_user) - window_size)*train_ratio)
            else:
                #start validation dataset
                start = round((len(df_user) - window_size)*train_ratio)
                 #end validation dataset
                end = len(df_user) - window_size
            for i in range(start, end):
                ts.append(list(df_user['value'].to_numpy()[i:i+window_size]))
                y.append(df_user['fraud'].to_numpy().astype(float)[i+window_size-1])
                count += 1
                if count == batch_size:
                    ts = pad_sequences(ts, padding='post', value=-1.0,
                        maxlen=max_window_size, dtype=float) #padding
                    yield(np.array(ts), np.array(y))
                    ts = []
                    y = []
                    count = 0
    return generator(df, train, users, max_window_size)
```

We also generate a dataset **more similar** to the real one:

```
        dt = np.dtype(dtype=[('user', int), ('timestamp', int), ('value',
            float), ('fraud', int)])
```

Testing the model on a small dataset of 10000 synthetic transactions with a ratio of 80% legitimate transactions and 20% fraudulent transactions led to the encouraging results.

synthetic3.py

We established that the **LSTM-based model** works well with the **imbalanced** dataset, now we want to make it work **efficiently**. In **synthetic3.py**, differently from before when we waited to have a whole epoch ready before outputting it, we pop out a **subset of time-series** as soon as we produce it.[3] This way, we can **distribute** the computation on **multiple GPUs** through the **mirrored strategy technique**.

```
     mirrored_strategy = tf.distribute.MirroredStrategy()
     with mirrored_strategy.scope():
```

The two following lines ensure the **auto-sharding** functionality is off.
Autosharding a dataset over a set of workers means each worker is assigned to a subset of the entire dataset. Since we feed the workers as soon as the instances are ready to go, we disable this option to ensure the compiler does not raise errors.

```
     options = tf.data.Options()
     options.experimental_distribute.auto_shard_policy =
         tf.data.experimental.AutoShardPolicy.OFF
```

synthetic4.py

In **synthetic4.py**, we incorporate the **real dataset**.
In first place, we drop cards and merchants not associated to fraudulent transactions to slightly balance the dataset and focus the learning process on fraudulent sequences:

```
    # Keep only some of the data
    if config.data.raw.keep:
        df = df[:config.data.raw.keep]

    # add column to track non-fraudulent users/vendors
    # merchantss and cards implied in fraud
    merchant_fraud = list(set(df.loc[df['fraud'] == 1]['merchant']))
```

[3]through the yield operator

```
    card_fraud = list(set(df.loc[df['fraud'] == 1]['card']))

    # Shall we keep only fraudulents?

    if config.data.fraudulent:
        df['fraudulent'] = 0
        df.loc[(df['card'].isin(card_fraud)) |
            (df['merchant'].isin(merchant_fraud)), 'fraudulent'] = 1
        print(f'Dropping non-fraudulent cards/merchants')
        # df.drop(df[df.fraudulent == 0].index)
        df = df[df.fraudulent == 1]
        df.drop('fraudulent', inplace=True, axis='columns')
```

Then, we make **continuous attributes categorical**. We drop **daytime** and substitute it with its categorical counterparts:

```
    df['datetime'] = pd.to_datetime(df['localdatetime'],
        infer_datetime_format=True, unit='ns')
    df['hour'] = df.datetime.dt.hour
    df['day'] = df.datetime.dt.day
    df['month'] = df.datetime.dt.month
    df['weekday'] = df.datetime.dt.weekday
    df.drop('datetime', inplace=True, axis='columns')
```

The only other **continuous attribute** is **amount**. To make it categorical, we **bucketize** it through the following **cutlog function**:

```
    # Bucketize continuous values
    # Only amount actually
    cut_log(df, 'amount')
    print('==== Amount ====')
    print(df['amount_b'].unique)
    print('================')
```

```
def cut_log(df, col):
    '''
    Compute a log scale for the values in col and bucketize them in col_b.
    '''
    n_bins = math.ceil(math.log10(ceil_power_of_10(df[col].max())))
    bins = [-1.0] + list(np.logspace(-1, n_bins, base=10,
        num=(n_bins+1)*3+1))
    labels = [0] + list(range(1, (n_bins+1)*3+1))
    if verbose:
        print(f'n_bins = {n_bins}')
        print(f'bins = {bins}')
        print(f'labels = {labels}')
    df[col+'_b'] = pd.cut(df[col], bins=bins, labels=labels)
    return df
```

Finally the **CBOW algorithm** can be applied[4].
CBOW algorithm is provided by the **Word2Vec** facility available on the **gensim library**. It takes as input a dictionary of words (in our case, **enumerable attributes**) and it outputs **vectors**.
This way it gives a **mathematical meaning** to the concept of **similarity**, capturing either the **syntactical** and **semantic** sense of words.

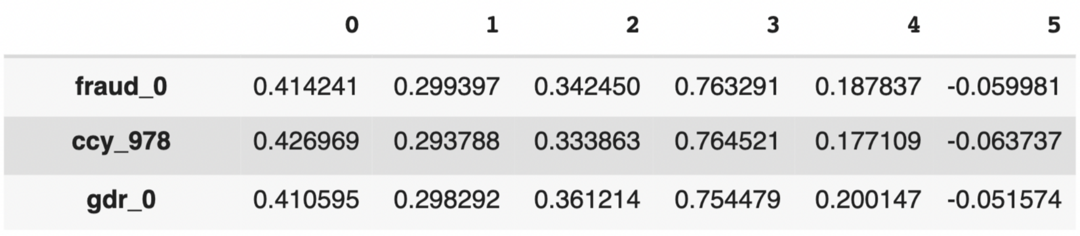
```
cboa_columns = [ 'card', 'network', 'amount_b', 'mcc', 'merchant',
    'country', 'sae', 'gdr', 'ert', 'secure', 'ccy', 'fraud', 'hour',
    'day', 'month', 'weekday']

tokenized_columns = ['t_' + c for c in cboa_columns if c not in ['card',
    'fraud']]

# Tokenize values
for c in cboa_columns:
df['t_'+c] = c + '_' + df[c].astype('str')

# Limit to the training set
df1 = df[df.train == 1]

# data_list = df1.loc[:, tokenized_columns].values.tolist()

data_list = df1[tokenized_columns+['t_fraud']].values.tolist()
# with open('c:/temp/data_list.txt', 'w') as f:
#     for item in data_list:
#         f.write("%s\n" % item)
cbow = Word2Vec(data_list, min_count=1,
                window=len(tokenized_columns),
                vector_size=config.data.cbow.vector_size,
                workers=config.data.cbow.workers,
                epochs=config.data.cbow.epochs,
                callbacks=[epoch_logger])
weights = [None] * (len(cbow.wv.key_to_index.keys())+1)
weights[0] = [-1.]*config.data.cbow.vector_size
token_index = {}
for k, w in enumerate(cbow.wv.key_to_index.keys()):
    weights[k+1] = cbow.wv.get_vector(w, norm=True).tolist()
    token_index[w] = k+1

# We may have a problem here if we compute cboa only on
# the train set: some values in the test set may not have
# been seen at all, hence .fillna(0)
for tok_c in tokenized_columns+['t_fraud']:
    df[tok_c] = df[tok_c].map(token_index).astype('Int64').fillna(0)
# print('DTypes after:')
# for c in df.columns:
#     print(f'{c}\t\t\t{df[c].dtype}')
```

[4]CBOA or CBOW are used indistinctly. CBOA stands for Continuous Bag of Attributes while CBOW stands for Continuous Bag of Word

We end up with a **dictionary** that pairs the attributes to **6-dimentional vectors**. The dimension of the vectors is an hyper-parameter chosen upon tuning.

The dictionary is shown in figure 5.2. The model is now assembled. In the following section, we show the experiments performed on it.

Figure 5.2: attributes encoded with CBOA



	0	1	2	3	4	5
fraud_0	0.414241	0.299397	0.342450	0.763291	0.187837	-0.059981
ccy_978	0.426969	0.293788	0.333863	0.764521	0.177109	-0.063737
gdr_0	0.410595	0.298292	0.361214	0.754479	0.200147	-0.051574

CHAPTER 6

Experiments

In **this section**, we will present the **experiments effectuated**.
The **first subsection** gives an overview of the **computational resources** we exploited to perform the **experiments**.
The **second subsection** is a short guide on how to reproduce them.
A **third subsection** explains in detail the **experiments** effectuated and the **rationale behind them**.

6.1 Moulon Mésocentre

Mésocentre is a project that *"rest upon the decision of two institutions, CentraleSupelec and ENS Paris-Saclay, to pool their computing resources within the Paris-Saclay University, host them at IDRIS and set up a common support team."*[1]
Mésocentre gives access to **Ruche** R, a *Lenovo Super Computer* and **Fusion** F, *HPE supercomputer*.

- **General information**:
  - OS distribution : CentOS 7.9.2009
  - Network technology : OPA network 100 Gbit/s
  - Storage technology : Spectrum Scale GPFS parallel file system (380 Tio of usable space, IOs rate : 9 GB/s)
  - Visualization solution : Enginframe / Nice DCV (30 licences)
  - Nvidia driver version : 470.57.02
- **Hardware Detail**

[1]reference to the Mésocentre website

- **User resources**
    * Two login nodes R
    * A Spectrum Scale GPFS parallel file system (380 Tio of usable space, IOs rate : 9 GB/s) R
    * Intel OPA network 100 Gbit/s R
- **Parallel distributed memory computing**
    * 192 compute nodes comprising 2 Intel Xeon Gold 6230 20 cores @ 2.1 GHz (Cascade Lake) and 192 GB of RAM R
- **Parallel shared memory and sequential computing**
    * 14 compute nodes comprising 4 Intel Xeon Gold 6230 20 cores @ 2.1 GHz (Cascade Lake) with 1.5 TB of RAM and 480 GB of SSD disk R
    * One server comprising 4 Intel Xeon CPU 6148 80 cores @ 2.4 GHz (Skylake) and 1.5 TB of RAM F
    * 3 servers comprising 4 Intel Xeon CPU 6148 80 cores @ 2.4 GHz (Skylake) and 768 GB of RAM F
    * 16 compute nodes comprising 2 Intel Xeon CPU 6148 40 cores @ 2.40 GHz (Skylake) and 192 GB of RAM F
- **GPGPU computing**
    * 10 GPGPU nodes comprising 2 Intel Xeon Gold 6230 20 cores @ 2.1 GHz (Cascade Lake), 768 GB of RAM and 4 NVIDIA Tesla V100 PCIe graphic cards with 32 GB of GRAM R
    * 5 GPGPU nodes comprising 2 Intel Xeon Gold 6230 20 cores @ 2.1 GHz (Ice Lake), 1024 GB of RAM and 4 NVIDIA Tesla A100 NVLink graphic cards with 40 GB of GRAM R
    * 4 GPGPU nodes comprising 2 Intel Xeon CPU 6126 24 cores @ 2.60 GHz (Skylake), 192 GB of RAM and 2 Nvidia Tesla P100 graphic cards with 12 GB of GRAM F
- **Visualization**
    * 3 visualization nodes comprising 2 Intel Xeon Gold 6230 20 cores @ 2.1GHz (Cascade Lake), 256 GB of RAM and 1 Nvidia Quadro RTX 5000 graphic card R

6.2 Quick How-To

Every user has availability of a **ruche-quota** that is accessible through an **ssh connection.**

```
Last login: Wed Feb 16 12:25:56 on ttys000
(base) ludole@MacBook-Air-di-Ludovica-2 ~ % ssh
    lermalu@ruche.mesocentre.universite-paris-saclay.fr
```

```
Last login: Wed Feb 16 02:26:25 2022 from 187.198.28.93.rev.sfr.net
            ______        __
 ,-.       /      \      /  \   ,--.                  Bonjour !
 `-'      /        \     \__/  /    \
   ,-----(          )-----.    \    /       the Mesocenter team wishes you
  /       \        /        _   `--'      a great time computing on ruche.
 /         \______/        | |
 \          _ __ _   _  ___| |_    ___       We appreciate your feedback.
  \        | '__| | | |/ __| '_ \ / _ \
   )-----( | |  | |_| | (__| | | |  __/      ----------------------------
  /        |_|   \__,_|\___|_| |_|\___|
 /          ______
 \         /      \         /        Support : ruche_support@groupes.renater.fr
  \       /        \       /
   `-----(          )-----'     Website : http://mesocentre.centralesupelec.fr/
          \        /
           \______/       https://mesocentre.pages.centralesupelec.fr/user_doc/

[lermalu@ruche01 ~]$
```

Each user has access to **two separate directories** for their own usage:

1. the users directory:

   ```
   /home/username
   ```

   where they can store their **binaries** and **data**.

2. the working directory:

   ```
   /workdir/username
   ```

   where they can store their **source files**.

To perform **the experiments** we navigate through the **work directory** up to **our project**.

```
[lermalu@ruche01 ~]$ cd /workdir/lermalu/ts-fraud/
[lermalu@ruche01 ts-fraud]$ ls
batch                                          meso-prepare_output.log
fraud_data                                   OpenKETF2
data-meso                                     README.md
fraud_data                                    req.txt
meso-gru-a100-1294213_error.log               Results.xlsx
meso-gru-a100-1294213_output.log              synthetic
meso-gru-a100-1294238_error.log               TODO.md
meso-gru-a100-1294238_output.log              training-local.png
meso-gru-a100_error.log                       training-meso-a100.png
meso-gru-a100_output.log                      ts-fraud-dist.py
meso-prepare_error.log
[lermalu@ruche02 ts-fraud]$
```

Among these files we are particularly interested in the **batch**, **fraud_data** and **conf directories**.

Navigating through the **batch directory**, we find several **scripts**. Each of them allocates the necessary **resources** and launches a **program** that will run within an **anaconda**

environment.
Here's for example, the **content** of the **meso-gru.sh**:

```
#!/bin/sh
#SBATCH -job-name=meso-gru-a100-%j
#SBATCH -output=meso-gru-a100-%j_output.log
#SBATCH -error=meso-gru-a100-%.log
#SBATCH -partition=gpua100
#SBATCH -nodes=1
#SBATCH -ntasks=1
#SBATCH -cpus-per-task=10
#SBATCH -gres=gpu:4
#SBATCH -time=24:00:00
# #SBATCH -qos=qos_gpu-t4
# #SBATCH -A zkh@gpu
module purge
module load cudnn/8.1.0.77-11.2-linux-x64/intel-20.0.2
module load miniconda3/4.7.12.1/intel-19.0.3.199
.
    "/gpfs/softs/spack/opt/spack/linux-centos7-cascadelake/intel-19.0.3.199/miniconda3-
cd
```

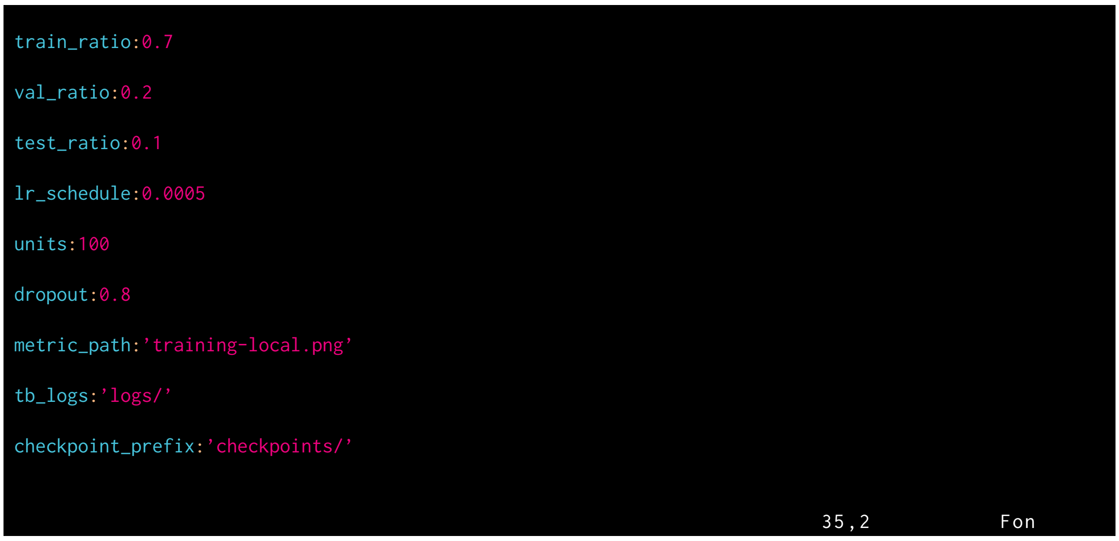
For our experiments we will launch the **meso-prepare.sh**, the **meso-gru.sh** and the **jz-lgbm.sh**.

The configuration of the **hyper-parameters** for the **experiments** can be set through the **config files** in the **conf directory**.

```
[lermalu@ruche02 conf]$ ls
config-jz-small.yaml  config-jz.yaml  config-meso.yaml  config.yaml
[lermalu@ruche02 conf]$
```

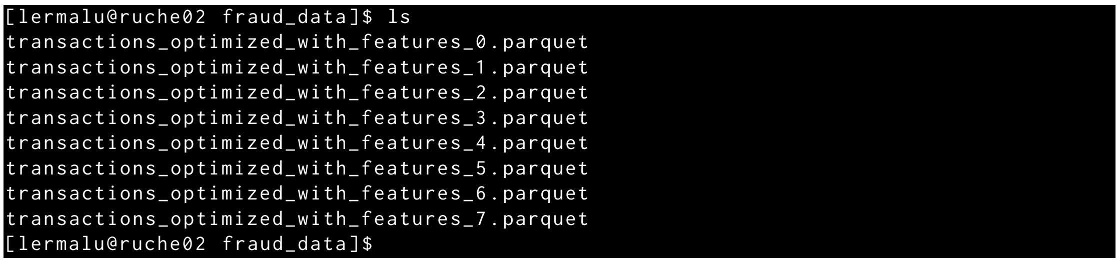
In particular, for the experiments with the **LSTM model**, we set them through the **config.yaml file**:

```
data:
  path:'data-meso/'
  prepared: 'fraud_data.parquet'
  fraudulent:1
   raw:
    path:'../../shared/XXXX/fraud_data/'.
    min:2
    max:5
    keep:38000000
  rfm:
    r_path:'recency/'
    f_path:'frequency/'
    m_path:'monetary/'
  cbow:
    vector_size:6
    epochs:100
    path:'cboa.json'
    workers:80
gru:
 window_size:12
 window_size_max:40
 epochs:30
 seq_len:18
```



```
train_ratio:0.7

val_ratio:0.2

test_ratio:0.1

lr_schedule:0.0005

units:100

dropout:0.8

metric_path:'training-local.png'

tb_logs:'logs/'

checkpoint_prefix:'checkpoints/'

                                                                  35,2          Fon
```

Finally, the **fraud_data** directory contains the **data files** in a **parquet format**.



```
[lermalu@ruche02 fraud_data]$ ls
transactions_optimized_with_features_0.parquet
transactions_optimized_with_features_1.parquet
transactions_optimized_with_features_2.parquet
transactions_optimized_with_features_3.parquet
transactions_optimized_with_features_4.parquet
transactions_optimized_with_features_5.parquet
transactions_optimized_with_features_6.parquet
transactions_optimized_with_features_7.parquet
[lermalu@ruche02 fraud_data]$
```

6.3 Experiments

We performed experiments either with the **LSTM model** and with the **LGBM model**. The aim is to *compare the results* and to deduce which one *performs better*.
As we already anticipated, the **most critical hyper-parameter** to fine-tune is the **window_size**. We tried different values of it to understand how many transactions the **LSTM model** needs in order to get significant temporal features.
In the following sections, we show **how to reproduce** the experiments and **how they work**.

How to run

- **check the parameters in your conf/*.yaml file of choice** or create a new one named **conf/my-config.yaml**

- **run python ts-fraud-dist.py –cfg conf/my-config.yaml prepare**. This will create the data frame and the embeddings for the slice of the dataset

- **run python ts-fraud-dist.py –cfg conf/my-config.yaml gru**. This will train and evaluate the model.
- **run python ts-fraud-dist.py –cfg conf/my-config.yaml lgbm**. This will train a **LGBM model** on the dataset with the engineered features, without any tweaking.
- **run python ts-fraud-dist.py –cfg conf/my-config.yaml lgbm-cboa**. This will train a **LGBM model** on the dataset with the CBOA features, without any tweaking.

How it works

For each experiment, we need to run either the **meso-prepare.sh** and the **meso.gru.sh** batches.

1. The **meso-prepare.sh** takes care of manipulating the data and prepare them for the **LSTM model**. During this step:
   - the original features are *slightly transformed*:
     - *amount* is bucketized
     - *datetime* is split into *hour, day, weekday, month*
   - cards and merchants that are never involved into fraud are dropped: this slightly helps in balancing the dataset
   - cards are assigned a **transaction counter** to keep transactions ordered by time, afterwards *localdatetime* is dropped
   - **window sizes** are computed per card[2], lower bounded by the hyper-parameter **window_size** and upper bounded by the hyper-parameter **window_size_max**
   - a *train value* is added by card in order to have a *train_ratio* time-series for training, a *val_ratio* for validation and a *test_ratio* for testing
   - all non training cards are used for evaluation
   - **CBOA embeddings** are computed on categorical encodings of the features and stored in a *JSON* file
   - the prepared dataframe is stored on disk
2. In the *second phase*, when **meso.gru.sh** is run:
   - the model is build, embeddings are loaded into the **Embedding layer** of the model
   - a **dataset** is built from a light generator returning only card numbers[3]
   - a **parallelized function** is applied to generate the actual timeseries from the card number
   - the model is trained on **multiple GPUs** with the **mirrored strategy**

[2] 1/10 of the number of transactions for the card
[3] it is an efficient and fast generator

Parameters

The followings are the *tunable parameters* of the **LSTM model**.

```
data:
  path: 'data-local/'             # Where to store the prepared files
  prepared: 'fraud_data.parquet' # Named of the prepared dataset
  fraudulent: 1                   # If we remove the cards/merchants never
      involved in fraud
  raw:
    path: '../../data-caps/'      # Where to find the original files
    min: 1                        # First parquet file to use
    max: 2                        # Last parquet file to use
    keep: 2000000                 # Number of transactions to keep
  rfm:
    r_path: 'recency/'
    f_path: 'frequency/'
    m_path: 'monetary/'
  cbow:
    vector_size: 6                # CBOA embeddings dimension
    epochs: 40                    # Number of epochs for computing the
        embeddings
    path: 'cboa.json'             # Filename for the embeddings
    workers: 12                   # Number of cores to use for computing the
        embeddings
gru:
  window_size: 12                 # Minimum window size for time series
  window_size_max: 40             # Maximum window size for time series
  epochs: 40                      # Number of epochs for training
  batch_size: 64
  seq_len: 18                     # Minimum number of transactions for a card
  train_ratio: 0.7
  val_ratio: 0.2
  test_ratio: 0.1
  lr_schedule: 0.0005
  units: 40                       # Number of units in the LSTM
  dropout: 0.8
  metric_path: 'training-local.png'  # Filename for the plot of the loss
      while traing
  tb_logs: 'logs/'                    # Tensorboard logs path
  checkpoint_prefix: 'checkpoints/'  # Checkpoints path
```

Experiments

As we anticipated, we have **two objectives**. The first one I is to compare different *window_sizes* and find out which one allows the **LSTM model** to perform the best. The term window size here is referring to the number of transactions used to train the LSTM model.[4] The second one II is to compare the **LSTM model** and the **LightGBM model**.

[4]the reader may confuse it with the widow size used to train the CBOA algorithm. However, that window size is fixed to the number of contextual attributes of the transaction.

In order to fulfill the objectives, we disposed **three categories** of experiments.

- The **first ones** are performed on cards with *short histories*, i.e cards related to **8 or more transactions**. The selection of this minimum value is based on an average of 7.76 transactions, ensuring that most of the dataset is accounted for. To include many cards, the window size for this experiment is 3, given the median is 4. The objective here is to assess the model's ability to learn despite the limited number of transactions.
- The **second ones** on cards with a *long histories*, i.e cards related to **18 or more transactions**. While it is true that the majority of cards have fewer than 25 transactions, it is crucial to recognize that longer card histories offer the LSTM more data to effectively learn patterns. To strike a balance between providing sufficient data for the LSTM to learn sequential patterns and including a reasonable number of cards in the analysis, a window size of 12 is selected for this experiment.
- The **third ones** on cards with a *longer histories*, i.e cards related to **40 or more transactions**. The objective of this experiment is to evaluate the performance of the model with extended card histories. Therefore, a window size of 32 is chosen to accommodate these lengthy sequences and understand how the model handles them.

For each experiment, we are checking it with I if it contributes to the first objective, and with II if it contributes to the second objective. The choice of the number of epochs for training the LSTM model is driven by the observation that the model can converge within a restricted number of epochs. Nevertheless, we extended the training duration to examine if any notable differences would arise in the model's predictions.

- **short history** experiments:
  1. **TS** classification **with CBOA encoding** - 25 epochs - window_size: 3 I II
  2. **TS** classification **with CBOA encoding** - 30 epochs - window_size: 3 I II
  3. **LGBM** classification **with augmented features** II
  4. **LGBM** classification **with CBOA encoding** II
- **long history** experiments:
  1. **TS** classification **with CBOA encoding** - 30 epochs - window_size: 12 I II
  2. **TS** classification **with CBOA encoding** - 80 epochs - window_size: 12 I II
  3. **LGBM** classification **with augmented features** II
  4. **LGBM** classification **with CBOA encoding** II
- **longer history** experiments:

1. **TS** classification **with CBOA encoding** - 30 epochs - window_size: 32 I

Results are exposed and discussed in the *Results* section [7].

CHAPTER

7 Results

In this section, the **results obtained** are presented.
The **first subsection** displays some **considerations** on the experimental process.
The **second subsection** presents the **performance metrics** used to evaluate the results.
Finally, the **third subsection** displays, for each experiment, the **results obtained**.

7.1 Consideration on the experimental process

- **About the Hyperparameters Setting of the Model.**

  We made different experiments, changing the geometry of the **LSTM model**. We empirically proved that, to perform optimally:

  1. the **number of units** of the **LSTM** should be equal to the **maximum window_size length**
  2. the **hidden state length** should match the **length of the sequences handled**: more units won't be used while fewer won't memorize
  3. Since some cards account to **thousands of transactions**, the *maximum_window_size* parameter **explodes**. With such values, the model takes an unfeasible amount of time to train, so we decided to cut the *maximum_window_size* down to a fixed value of **40**.

- **About the Vanishing Gradient.**

Outliers notably[1] badly affect the training process, resulting in either **vanishing** or **exploding gradient**. We:

1. set the parameter **global_clipnorm** to **0.5** in the *keras.optimizers.Adam*(...) options. It allows to cap **L2 norm** of the gradients at the specified value to *smooth* its values.
2. put a **higher weight** on the instances of the positive class when computing the loss function. This way we **counterbalance** the dataset imbalance.

- **About Convergence.**

  The learning rate needs to be set to a value around **0.001**. Values on the order of magnitude of **0.01** have been proved to be ***way too high***, and the LSTM model **doesn't converge**.

7.2 Performance Metrics

As we already mentioned, **accuracy** is not a helpful metric when evaluating imbalanced datasets, as it may be **misleading**. For instance, if we'd build a model that predicts "non-fraud" on all instances, it would score a **99.6%** accuracy, since fraud represents only the 0.4% of all instances. Metrics that can provide better insight include:

- **False negatives** and **false positives**: incorrectly classified samples.
- **True negatives** and **true positives**: correctly classified samples.
- **Precision**: percentage of predicted positives correctly classified. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
- **Recall**: percentage of actual positives that were correctly classified. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
- **AUC** refers to the Area Under the Curve of a Receiver Operating Characteristic curve (ROC-AUC). This metric is equal to the probability that a classifier will rank a random positive sample higher than a random negative sample

Among the metrics used to evaluate the performance of the models, the most important ones are **Recall** and **Precision**.

[1] Activations saturate at either tail of 0 or 1, and gradients are near zero in these regions.

Recall

Recall is the number of *true positives* divided by the *sum of true positives and false negatives*.

$$\mathrm{recall} = \frac{\mathrm{TruePositives}}{\mathrm{TruePositives} + \mathrm{FalseNegatives}}$$

A high recall indicates the model can successfully **identify relevant instances without mislabeling them as irrelevant**.

In the fraud detection task, classifying fraudulent transactions as non-fraudulent outcomes in a **loss of money** and **reliability** for the card-provider company.

Precision

Precision is the rate of *true positives* divided by the *sum of true positives and false positives*.

$$\mathrm{recall} = \frac{\mathrm{TruePositives}}{\mathrm{TruePositives} + \mathrm{FalsePositives}}$$

A high precision indicates the model is able to successfully return **relevant instances with limited irrelevant results**[39].

In the fraud detection task, classifying non-fraudulent transactions as fraudulent means for the client to have **their money frozen and the service suspended** for an undefined amount of time.

Our major interest relies on **recall**.

7.3 Results

Here below, the results obtained are presented in **tables**.
Such tables include both train and test results.[2] and a *Support* column that indicates the number of instances of the dataset used to train and test the model. Moreover, the experiments carried out with the LGBM report only the results obtained in testing, since the training results were very good and therefore not of particular interest.

The experiments are grouped by category.

- **Short history** experiments:

  This first three experiments test the efficacy of the models on a dataset made of cards with **histories** made of **8 or more** transactions.

[2]These last are highlighted in green

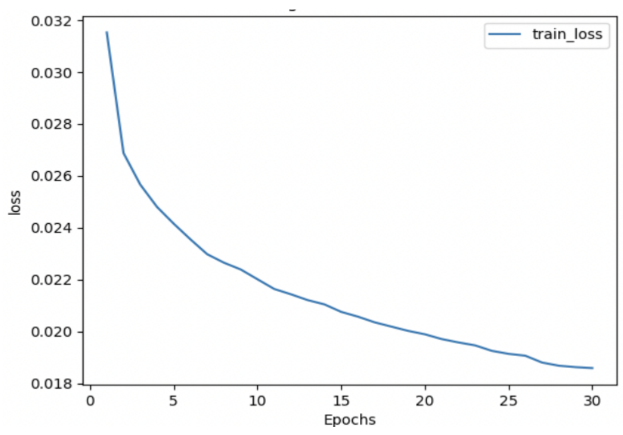


Figure 7.1: LSTM with CBOA encoding and window_size 8 loss function on 30 epochs

1. **Time-series classification with LSTM**
   The first experiment tests the **LSTM** model.
   As we explained in the **Model section**[3], the history of each card is cut down in shorter sequences. These sequences are as long as indicated either in the *window_size* parameter or in the *max_window_size* parameter, depending on which one of the two provides the biggest value. The sequences are then inputted to the model. We set a *window_size* of length 3, which means the shortest sequences in input to the model will be constituted of only 3 transactions.
   As seen in the **Dataset** section[4], most of our data is constituted by card with short histories. This means most sequences in input to the **LSTM** will be short. The results obtained, which are shown in table 7.1, are **not sufficient.** The experiment run on *30 epochs* presents a better trade off between the two metrics, but scores are still unsatisfactory.
   The loss function is shown in figure 7.1. It converges smoothly.
   This is not surprising, since we are considering only very short sequences of transactions for the computation.

LSTM with CBOA encoding										
short history									*window_size: 3*	
	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
25 epochs										
train	0,0204	22692	9873	16976884	77527	0,9949	0,6968	0,2264	0,9001	
test	0,0253	3156	3000	7279014	36942	0,9945	0,5127	0,0787	0,8656	7322112
30 epochs										
train	0,0130	55544	7837	16978920	44675	0,9969	0,8764	0,5542	0,9388	
test	0,03138	5756	17795	7264219	34342	0,9928	0,2445	0,1435	0,8112	7322112

Table 7.1: Time-series classification with CBOA encoding and LSTM on short history cards

2. **Time-series classification with LGBM.**

   The following **Light Gradient Boosting Machine** is trained on the same support dataset as the **LSTM model** to compare their results. We trained the **LGBM** either on the **raw dataset**, with the aid of the **augmented features**[3], or on the **CBOA encoded dataset.**
   The table 7.2 displays the results obtained by the **LGBM** with augmented features. The table 7.3 displays the results obtained by the LGBM trained on the instances encoded with CBOA. In either case, the results are similar to the ones obtained before and similarly poor.

- **Long history** experiments:

  These experiments test the **models** on a dataset made of cards with **histories** that counts **18 or more** transactions.

  1. **Time-series classification with LSTM.**
     The *window_size parameter* is set to 12, so we will have fewer, longer sequences. To counterbalance this, we increase the support dataset.
     The results displayed in table 7.4 show an **improvement** in recall. In particular, the experiment run during *30 epochs* shows a **significant improvement in recall**, and the results seem to hint to use *long histories* in order for the **LSTM** to extract more significant features.
     The loss function, shown in figure 7.2, hardly converge to values lower than 0.05.

[3]Discussed in the **Dataset section**, in the *Derived features subsection*[4]

lgbm classification with augmented features	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
train						0,9949				
test		5843	7841	7274411	34263	0,9942	0,4270	0,1457		7322358

Table 7.2: Time-series classification with augmented features and LGBM on short history cards

lgbm classification with CBOA econding	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
train						0,9945				
test		3297	4166	7278086	36809	0,9944	0,4418	0.0822		7322358

Table 7.3: Time-series with CBOA encoding and LGBM on short history cards

TS with CBOA encoding										
long history									*window_size: 12*	
	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
30 epochs										
train	0,0229	24640	9135	9264781	48004	0,9939	0,7295	0,8293	0,9296	9346560
test	0,1414	9790	12900	3966309	20473	0,9917	0,4315	0,3235	0,9077	4009472
80 epochs										
train	0,0068	60243	5672	9268242	12403	0,9981	0,9139	0,3392	0,9850	9346560
test	0,0499	8009	14469	3964740	22254	0,9908	0,3563	0,2646	0,7464	4009472

Table 7.4: Time-series classification with CBOA encoding and LSTM on long history cards

lgbm classification with augmented features	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
train						0,9937				
test		5843	6112	3973264	24424	0,4887	0,1930	0,1457		4009643

Table 7.5: Time-series classification with LGBM and augmented features on long history cards

LGBM classification with CBOA econding	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
train						0,9930				
test		2854	2984	3976392	27413	0,9924	0,4889	0,0943		4009643

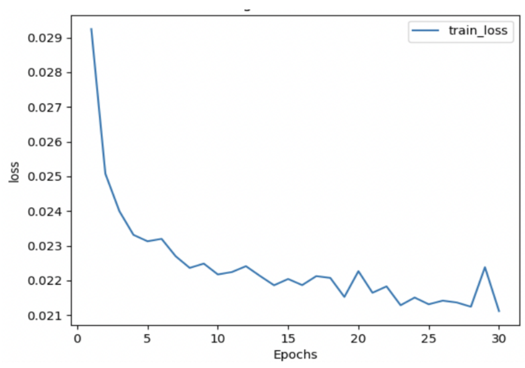
Table 7.6: Time-series classification with LGBM and CBOA encoding on long history cards

2. **Time-series classification with LGBM.**

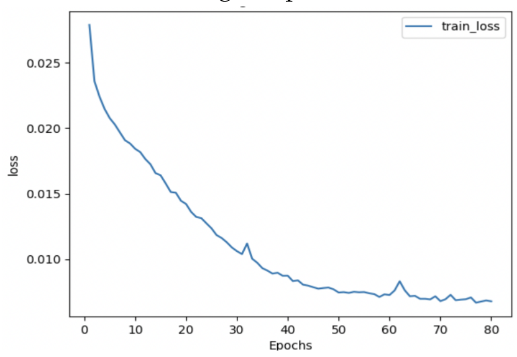
   We now train and test the **LGBM**, either with derived features, which results shown in table 7.5 and CBOA encoding, which results shown in table 7.6. As before, we use the same support dataset to obtain comparable results. The experiments run with the **LGBM** on *long histories* don't show much improvement with respect to the ones run on the *short histories*.

- ***Longer history*** experiments:

  This last experiment tested the performance of the **LSTM model** on cards related to **histories** of **40 or more** transactions. The evaluation in this experiment is specifically centered around the LSTM model. The main interest lies in understanding how longer history affects the model's performance, as described in the experiments section[6.3]. The **window_size** is set to **32**. It is to notice that most of the dataset is made of cards related to short histories, so that the support dataset for



(a) Loss function of LSTM with CBOA and window_size 12 during 30 epochs



(b) Loss function of LSTM with CBOA and window_size 12 during 80 epochs

Figure 7.2

this experiment is less populated than before. The results displayed in table 7.7 show that there is **not a significant improvement** on the performance of the model.

TS with CBOA encoding										
longer history									*window_size: 32*	
	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
30 epochs										
train	0.0203	31174	9487	9264429	41470	0.9945	0.7667	0.4291	0.9368	377200
test	0.0330	8994	12590	3966619	21269	0.9916	0.4167	0.2972	0.8777	161200

Table 7.7: Time-series classification with LSTM and CBOA encoding on longer history cards

Chapter

8 Discussion

In this section, we **compare** in a more straight forward way the results obtained in the previous section.
In the **first subsection**, we make some general considerations.
In the **second subsection**, we discuss the **main questions** we advanced in the previous sections[1]:

1. Which model is better in performing the **fraud detection task**: **Light Gradient Boosting Trees** or the **Time-Series Classification** implemented by the **LSTM-based model**?

2. Which value of the *window_size* allows the **LSTM-based model** to perform the best?

8.1 General Considerations on the Results Obtained

Unfortunately, the first consideration we need to make is that the **results obtained** are not sufficient enough to be used in production.
Indeed, the best result on the optimization of the loss function is the one obtained with the **LSTM model trained for 30 epochs on long histories** and scores around **0,0499**. Although, we were looking for results **1000 times lower!**

We made **some efforts** to improve this result.
At first, we checked the **consistency of the dataset**: we looked out for time-series classified twice and flagged as fraudulent and non-fraudulent, but **only three** of them presented this issue.
We then tried to apport some **changes to the geometry of the model**: we added a **third**

[1]More info about them in the last paragraph of the *experiments section* [6.3]

dense layer and used **two LSTM** layers instead of one.
Unfortunately, *no significant changes* were reported.
Eventually, to check if there was some bug in the code, we included targets in the training dataset to check whether the **LSTM-based model** could correctly classify all instances. The loss function in this case converged to zero. Therefore, the problem shall simply consist of the fact that ***this kind of model has a hard time learning this kind of task***.

8.2 Comparison LGBM - LSTM

The two tables 8.1, 8.2 provide a **direct comparison** between the **Light Gradient Boosting Trees** and the **Time-Series Classification** implemented by the **LSTM-based model**, either on short histories and on long histories.
In the first case, we cannot see significant improvements in the use of the **LSTM-based model** with respect to the use of the **Light Gradient Boosting Trees**.
On long histories, the **LSTM-based model** proves to ***perform remarkably better*** than its counterpart.
Indeed, we observe an improvement in **recall**, that scores around **0.3**.

Even if there are no similar studies to take as reference to determine whether this is a satisfying result, we can conclude the LSTM-based model is superior to the LGBM, solving the task of fraud detection on a dataset constituted of cards with long history.

The performance of the **LSTM-based model** on **short histories** is not excellent: the **Recall** score is comparable to the one achieved by **LGBM**, so we cannot state that a model is superior to the other.

Table 8.1: Comparison TSc with CBOA encoding and LGBM - short history

TSc with CBOA encoding *short history*										
	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
30 epochs										
train	0,0204	22692	9873	16976884	77527	0,9949	0,6968	0,2264	0,9001	
test	0,0253	3156	3000	7279014	36942	0,9945	0,5127	0,0787	0,8656	
80 epochs										
train	0,0130	55544	7837	16978920	44675	0,9969	0,8764	0,5542	0,9388	7322112
test	0,03138	5756	17795	7264219	34342	0,9928	0,2445	0,1435	0,8112	7322112
LGBM classification with augmented features										
train						0,9949				
test		5843	7841	7274411	34263	0,9942	0,4270	0,1457		7322358
LGBM classification with CBOA econding										
train						0,9945				
test		3297	4166	7278086	36809	0,9944	0,4418	0.0822		7322358

Table 8.2: Comparison TSc with CBOA encoding and LGBM - long history

TS with CBOA encoding *long history*										
	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
30 epochs										
train	0,0229	24640	9135	9264781	48004	0,9939	0,7295	0,8293	0,9296	9346560
test	0,1414	9790	12900	3966309	20473	0,9917	0,4315	0,3235	0,9077	4009472
80 epochs										
train	0,0068	60243	5672	9268242	12403	0,9981	0,9139	0,3392	0,9850	9346560
test	0,0499	8009	14469	3964740	22254	0,9908	0,3563	0,2646	0,7464	4009472
LGBM classification with augmented features										
train						0,9937				
test		5843	6112	3973264	24424	0,4887	0,1930	0,1457		4009643
LGBM classification with CBOA econding										
train						0,9930				
test		2854	2984	3976392	27413	0,9924	0,4889	0,0943		4009643

In both models, generalization is not good: the test phase shows Precision and Recall that are quite low. However, this is expected given the unbalanced dataset: the fraud examples are rare, so it is difficult to learn to identify them. Moreover, we have no idea of *"what makes a fraud"*, so if there are characteristics in the data to identify them. Additionally, the training time consumed by the **LSTM** model is huge compared to the one of the **LGBM** one, casting doubts on whether such slightly better results are worth it.

Differently from before, the **LSTM model** achieves considerably **better results** than the **LGBM** ones.
Indeed, the **LSTM model** trained on **30 epochs** scores **as twice as** the value on **recall** of the **LGBMs**.
This result is enough to state that **LSTM model outperforms LGBM model.**

8.3 Comparison within different window_sizes

The table 8.1 provides an immediate comparison between the performances achieved by the **LSTM model** set with different values of **window_sizes**. The three models are trained on different datasets:

1. **the first one**, with a **window_size set to 3**, is trained on a database made of cards with an history made of 8 or more transactions.

2. **The second one**, with a **window_size set to 12**, is trained on a database made of cards with an history made of 18 or more transactions.

3. **The last one**, with a **window_size set to 32**, is trained on a database made of cards with an history made of 40 or more transactions.

The three models are trained during **30 epochs**.
The best result is achieved by the second model.
Although, it must be considered that the majority of the cards are related to short histories. Indeed, when selecting cards with longer and longer histories, we end up with much fewer populated databases.

Table 8.3: Comparison window_sizes for TSc performance

TSc with CBOA encoding										
short history									*window_size: 3*	
	Loss	TP	FP	TN	FN	ACC	PREC	RECALL	AUC	Support
30 epochs										
train	0,0130	55544	7837	16978920	44675	0,9969	0,8764	0,5542	0,9388	7322112
test	0,03138	5756	17795	7264219	34342	0,9928	0,2445	0,1435	0,8112	7322112
long history									*window_size: 12*	
30 epochs										
train	0,0229	24640	9135	9264781	48004	0,9939	0,7295	0,8293	0,9296	9346560
test	0,1414	9790	12900	3966309	20473	0,9917	0,4315	0,3235	0,9077	4009472
longer history									*window_size: 32*	
30 epochs										
train	0.0203	31174	9487	9264429	41470	0.9945	0.7667	0.4291	0.9368	377200
test	0.0330	8994	12590	3966619	21269	0.9916	0.4167	0.2972	0.8777	161200

Chapter

9 Conclusion

To conclude, we can state that the **the LSTM-based model is superior to the LGBM, solving the task of fraud detection on a dataset constituted of cards with long history and window_size equal to 12, with an improvement of 0.1778 points on Recall**.
However, as pinpointed all along this report, our model presents various deficiencies.
The **main issues** are:

- **Sub-optimal performances.**
  In first instance, we can affirm that, even though the model we proposed performs better than the LGBM one, ***it still doesn't achieve the desired results***.
  In the *Experiments* section[6.3], we pinpointed that longer sequences may help the LSTM extract more significant features to correctly classify the following instance.
  However, very long sequences result in an infeasible training time.
  A possible approach to solve this issue may be to turn to the **Transformer architecture**. Transformers are models which use either CNN and RNN to implement the attention mechanism in a parallelized way.[1] A more significant insight on this topic can be found in the paper *Attention is all you need, Vaswani et al.*[40].

- **Limitations due to the nature of the model.**
  The proposed model:

  1. **fails at classifying transactions of unseen cards**.
     Our model is based on a **LSTM network** which takes as input time-series and uses it to classify the following instance.
     If we provide such a network only one transaction, ***it just won't produce a reasonable prediction***. This issue applies not only to cards related to only one

[1]a clear explanation about the limitation of the LSTM and how transformers overcome them can be found here How Transformers Work

transaction, but also to those that account for only a few.
Notably, in our experiments, we exclude such cards from the training and testing procedures.
To fix this issue, we can turn to the aid of **unsupervised learning techniques**. Such techniques aim to **profiling the distribution** of legitimate transactions and classify as fraudulent any outliers.
Moreover, unsupervised approaches are complementary to supervised ones: supervised techniques learn from past fraudulent behaviors, while unsupervised techniques target the detection of new types of fraud. Some example in literature of the combined use of the two approach can be found in papers like: *Combining unsupervised and supervised learning in credit card fraud detection, Carcillo et al.*[41], *An application of supervised and unsupervised learning approaches to telecommunications fraud detection, Hilas et al.*[42] and *Combining supervised and unsupervised learning for zero-day malware detection, Comar et al.*[43].

2. **Periodical retraining**.
Card-owners behavior is determined by a ***dynamical process*** and changes continuously.
This phenomenon is called **Concept Drift**.
Our model, however, is not able to include this variability in its predictive capability: when the data it has been trained on becomes outdated, its performances drops and it is necessary to retrain it on new data.
A possible solution is to include our model in a **Continual Learning framework**.[2] Some deeper insights about Continual Learning applied to the fraud detection task can be found in papers like *Autonomous deep learning: Continual learning approach for dynamic environments, Ashfahani et al.*[44] and *Continual learning for anomaly detection with variational autoencoder, Wiewel et al.*[45].

[2]Continual Learning is built on the idea of learning continuously and adaptively about the external world and enabling the autonomous incremental development of ever more complex skills and knowledge. More about it here: https://medium.com/continual-ai/towards-adaptive-ai-with-continual-learning-f493fd0d698

Bibliography

[1] European Central Bank. *Payments statistics: 2020*. Last accessed 25 January 2021. 2020. URL: https://www.ecb.europa.eu/press/pr/stats/paysec/html/ecb.pis2020~5d0ea9dfa5.en.html.

[2] pwc. *PwC's Global Economic Crime and Fraud*. Last accessed 25 January 2021. 2020. URL: https://www.pwc.com/gx/en/services/forensics/economic-crime-survey.html.

[3] Zisheng Li, Jun-ichi Imai, and Masahide Kaneko. "Facial-component-based bag of words and phog descriptor for facial expression recognition." In: *2009 IEEE International Conference on Systems, Man and Cybernetics*. IEEE. 2009, pp. 1353–1358.

[4] Roberto Toldo, Umberto Castellani, and Andrea Fusiello. "A Bag of Words Approach for 3D Object Categorization." In: *Computer Vision/Computer Graphics CollaborationTechniques*. Ed. by André Gagalowicz and Wilfried Philips. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 116–127. ISBN: 978-3-642-01811-4.

[5] Jin Wang, Ping Liu, Mary F.H. She, Saeid Nahavandi, and Abbas Kouzani. "Bag-of-words representation for biomedical time series classification." In: *Biomedical Signal Processing and Control* 8.6 (2013), pp. 634–644. ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2013.06.004. URL: https://www.sciencedirect.com/science/article/pii/S174680941300089X.

[6] Khaled Gubran Al-Hashedi and Pritheega Magalingam. "Financial fraud detection applying data mining techniques: A comprehensive review from 2009 to 2019." In: *Computer Science Review* 40 (2021), p. 100402.

[7] Altyeb Altaher Taha and Sharaf Jameel Malebary. "An Intelligent Approach to Credit Card Fraud Detection Using an Optimized Light Gradient Boosting Machine." In: *IEEE Access* 8 (2020), pp. 25579–25587. DOI: 10.1109/ACCESS.2020.2971354.

[8] David G Whiting, James V Hansen, James B McDonald, Conan Albrecht, and W Steve Albrecht. "Machine learning methods for detecting patterns of management fraud." In: *Computational Intelligence* 28.4 (2012), pp. 505–527.

[9] Kunlin Yang and Wei Xu. "FraudMemory: Explainable Memory-Enhanced Sequential Neural Networks for Financial Fraud Detection." In: *Hawaii International Conference on System Sciences*. 2019.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[11] Ghosh and Reilly. "Credit card fraud detection with a neural-network." In: *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*. 1994. DOI: 10.1109/HICSS.1994.323314. URL: https://ieeexplore.ieee.org/abstract/document/323314/authors#authors.

[12] Raghavendra Patidar, Lokesh Sharma, et al. "Credit card fraud detection using neural network." In: *International Journal of Soft Computing and Engineering (IJSCE)* 1.32-38 (2011).

[13] Kang Fu, Dawei Cheng, Yi Tu, and Liqing Zhang. "Credit Card Fraud Detection Using Convolutional Neural Networks." In: *Neural Information Processing*. Ed. by Akira Hirose, Seiichi Ozawa, Kenji Doya, Kazushi Ikeda, Minho Lee, and Derong Liu. Cham: Springer International Publishing, 2016, pp. 483–490. ISBN: 978-3-319-46675-0.

[14] Aisha Abdallah n, Mohd Aizaini Maarof, and Anazida Zainal. "Fraud detection system: A survey." In: *Journal of Network and Computer Applications* (2016). DOI: http://dx.doi.org/10.1016/j.jnca.2016.04.007. URL: https://arxiv.org/abs/2009.11732 (visited on 02/08/2021).

[15] Bernhard Schölkopf, Robert C Williamson, Alex Smola, John Shawe-Taylor, and John Platt. "Support Vector Method for Novelty Detection." In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/8725fb777f25776ffa9076e44fcfd776-Paper.pdf.

[16] David M. J. Tax and Robert P. W. Duin. "Support Vector Data Description." In: *Machine Learning* 54 (2004), pp. 45–66.

[17] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. "Translating Embeddings for Modeling Multi-relational Data." In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

[18] Andreas Arning, Rakesh Agrawal, and Prabhakar Raghavan. "A Linear Method for Deviation Detection in Large Databases." In: *KDD*. Vol. 1141. 50. 1996, pp. 972–981.

[19] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. "Towards parameter-free data mining." In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 206–215.

[20] Caleb C Noble and Diane J Cook. "Graph-based anomaly detection." In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pp. 631–636.

[21] Lion Bergman and Yedid Hoshen. "Classification-Based Anomaly Detection for General Data." In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=H1lK_lBtvS.

[22] Chen Qiu, Timo Pfrommer, Marius Kloft, Stephan Mandt, and Maja Rudolph. *Neural Transformation Learning for Deep Anomaly Detection Beyond Images*. 2022. arXiv: 2103.16440 [cs.LG].

[23] Diego Lopez Yse. *Your Guide to Natural Language Processing (NLP)*. URL: https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1.

[24] Matteo Matteucci. *Word Embeddings*.

[25] Tomas Mikolov, Quoc Le, and Ilya Sutskever. "Exploiting Similarities among Languages for Machine Translation." In: (Sept. 2013).

[26] *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[27] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[28] Gang Chen. "A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation." In: (Oct. 2016).

[29] Michael Kearns. "Thoughts on Hypothesis Boosting." In: (1988).

[30] Leo Breiman. *Arcing the edge*. Tech. rep. Technical Report 486, Statistics Department, University of California at ..., 1997.

[31] Jerome H Friedman. "Stochastic gradient boosting." In: *Computational statistics & data analysis* 38.4 (2002), pp. 367–378.

[32] *What is feature engineering*. URL: https://www.displayr.com/what-is-feature-engineering.

[33] *Basic Feature Engineering With Time Series Data in Python*. URL: https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/.

[34] Chris Arney. "Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World." In: *Mathematics and Computer Education* 48.1 (2014), p. 126.

[35] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting." In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[36] Leo Breiman. "Prediction games and arcing algorithms." In: *Neural computation* 11.7 (1999), pp. 1493–1517.

[37] URL: https://parquet.apache.org.

[38] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling." English (US). In: *NIPS 2014 Workshop on Deep Learning, December 2014*. 2014.

[39] Mark Bentivegna. *Precision Vs. Recall — Evaluating Model Performance in Credit Card Fraud Detection*. URL: https://towardsdatascience.com/precision-vs-recall-evaluating-model-performance-in-credit-card-fraud-detection-bb24958b2723.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017).

[41] Fabrizio Carcillo, Yann-Aël Le Borgne, Olivier Caelen, Yacine Kessaci, Frédéric Oblé, and Gianluca Bontempi. "Combining unsupervised and supervised learning in credit card fraud detection." In: *Information sciences* 557 (2021), pp. 317–331.

[42] Constantinos S Hilas and Paris As Mastorocostas. "An application of supervised and unsupervised learning approaches to telecommunications fraud detection." In: *Knowledge-Based Systems* 21.7 (2008), pp. 721–726.

[43] Prakash Mandayam Comar, Lei Liu, Sabyasachi Saha, Pang-Ning Tan, and Antonio Nucci. "Combining supervised and unsupervised learning for zero-day malware detection." In: *2013 Proceedings IEEE INFOCOM*. IEEE. 2013, pp. 2022–2030.

[44] Andri Ashfahani and Mahardhika Pratama. "Autonomous deep learning: Continual learning approach for dynamic environments." In: *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM. 2019, pp. 666–674.

[45] Felix Wiewel and Bin Yang. "Continual learning for anomaly detection with variational autoencoder." In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 3837–3841.