UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica
Dipartimento di Eccellenza
Corso di Laurea Magistrale in Informatica

TESI DI LAUREA

A SCORE FOR FRAUD DETECTION LEVERAGING MACHINE LEARNING

RELATORI

Prof. **Michele Nappi**

Prof. **Fabio Narducci**

Università degli Studi di Salerno

CANDIDATO

Salvatore Froncillo

Matricola: 0522500858

Anno Accademico 2020-2021

Abstract

With the increasing diffusion of digital currency, it is of fundamental importance to put in place mechanisms capable of combating fraud. Although banking institutions have teams of experts capable of evaluating the genuineness of a transaction, this is clearly a problem that requires the support of a specific IT infrastructure. In fact, although it is possible to accept or reject a transaction based on rules established manually by experts, systems of this type are not scalable when the complexity of the parameters to be considered increases significantly. In building an automated Fraud Detection system, Machine Learning can make a substantial contribution. Using these techniques, it is in fact possible to extract patterns characterizing fraudulent transactions from the data, even when they are so complex that they cannot be even identified by a team of experts. This information can then be exploited to classify future transactions as fraudulent or genuine. Over the years a variety of Machine Learning algorithms have been used to tackle the problem of Fraud Detection. The present work of thesis aims at using a combination of three algorithms to develop a score for the classification of the risk of a transaction. The algorithms used are XGboost, Random Forest and an Artificial Neural Network (ANN). The score consists in a value ranging from 0 to 3, based on how many algorithms define the transaction as fraud.

A Mamma e Papà, a Federica, a Luigi e Simone,
per l'amore ed il supporto che quotidianamente mi donate.
Grazie di cuore

Contents

Contents **2**

List of Figures **5**

List of Tables **7**

Introduction **8**

1 The Problem of digital frauds **11**

1.1 Introduction to the problem 11

2 Machine learning algorithms **15**

2.1 Supervised and unsupervised learning 15

2.2 Algorithms Overview 16

2.3 Chosen Algorithms 22

2.3.1 Random Forests 22

2.3.2 XGBoost 23

2.3.3 Artificial Neural Network 24

3 Solution proposed **29**

3.1 Development Environment 29

3.2 The Dataset 30

3.3 Principal Component Analysis 31

3.4 Management of Class Imbalance 31

3.4.1 Oversampling 31

3.4.2 SMOTE 32

3.4.3 Undersampling 32
3.5 Metrics 33

4 Implementation **39**
4.1 Premise 39
4.2 Preliminary operations 40
4.2.1 Libraries Import 40
4.2.2 Dataset Import 40
4.2.3 Subdivision of the dataset 41
4.3 Hybrid algorithm 42
4.3.1 Random Forests and XGBoost 42
4.3.2 Artificial Neural Network Function 42
4.3.3 Plot function 43
4.3.4 Statistics function 44
4.3.5 Feature importance Function 45
4.3.6 Score function 46
4.4 Hybrid Algorithm vs Single Algorithms 47

5 Conclusion and future developments **48**
5.1 Conclusion 48
5.2 Future developments 49

A Appendix **50**
A.1 Libraries Import 50
A.2 Dataset Import 51
A.3 Random Forests and XGBoost train 51
A.4 Random Forests and XGBoost load 51
A.5 Artificial Neural Network Function 51
A.6 Plot Function 52
A.7 Statistics function 53
A.8 Feature importance Function 54
A.9 Score function 55

Rigraziamenti **57**

Bibliography **59**

List of Figures

1.1 Percentage of use for the different types of payment, by number and total value [1] 12

1.2 Left Axis: Total volume of fraud in percentage per platform. Right Axis: Volume of fraud as a percentage of total transactions [2] . . 14

2.1 Logistic Curve. Source: wikipedia.it 16

2.2 Example of Decision Tree. Source: eloquentarduino.github.io . . . 17

2.3 Simplified example of Random Forest. In this specific case the predicted class will be C. Source: `https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6` 18

2.4 K-Nearest Neighbors. Source: `https://www.researchgate.net/figure/A-k-nearest-neighbor-KNN-classifier-KNN-is-explained-as-follows_fig1_318096864` 19

2.5 Neural network structure. Source: Wikipedia.it 19

2.6 A simple Bayesian Network with conditional probability tables. Source: Wikipedia.com 21

2.7 The hyperplanes needed to separate samples with two features are nothing more than straight lines [3] 22

2.8 Neural network structure. Source: Wikipedia.it 25

2.9 Artificial Neuron of McCulloch and Pitts. Source: researchgate.net 26

3.1 A simple example of confusion matrix 34

4.1 pie chart representing the distribution of classes. 41

4.2 Graph representing Model Loss in relation to the number of epochs 43
4.3 Graph representing Model Accuracy in relation to the number of epochs 44
4.4 Representation of features in order of importance for Random Forest Algorithm 45
4.5 Representation of features in order of importance for XGBoost Algorithm 46

List of Tables

4.1 Chosen Algorithms Results 47
4.2 Hybrid Score Results 47

Introduction

The issue of fraud is destined to take on ever greater importance, with the progressive abandonment of the use of cash in favor of more modern and traceable payment methods. However, these payment methods bring with them - in addition to undeniable benefits - also new risks. In fact, if the scams involving cash could be based mainly on the use of counterfeit coins or banknotes or on the material theft of the same, electronic payment methods present other problems, including the possible theft of sensitive data such as number of card and CVV code, or your account credentials. This data can then be used to conduct illegitimate transactions, with a very low probability that the seller will detect something suspicious, especially if the transactions take place online. Banks, both credit and debit card issuers, are understandably at the forefront of trying to avoid fraud involving such payment methods. Fraud can in fact affect not only the customer who is directly affected by it, but also the bank itself, if it is required to compensate the defrauded user. To this end, various *fraud prevention* systems are in place that can block illegitimate transactions before they can occur. An immediate example of these security systems is the card's PIN code, in most cases verified by the terminal when a purchase is made in a physical store. Other less obvious but equally important steps carried out by the terminal include checking the validity and balance of the card used. However, there are scenarios where preventing an illegal transaction is not possible. To effectively deal with such situations, *fraud detection* systems must also be in place. These systems include rules under which a transaction can be considered more or less suspicious. In addition, each banking institution will have a team of experienced investigators, able to manually verify the authenticity of transactions. However, given the amount of transactions that

are carried out every day and their complexity, a crucial role is played by the so-called *Data Driven Models*, automated systems that often use Machine Learning to identify transactions with suspicious characteristics. The present thesis work is concerned precisely in this phase of the development of a *Fraud Detection System*. In fact, we propose to identify a strategy in the design of a *hybrid algorithm*, starting from the preprocessing of a transaction history, passing through the choice of Machine Learning algorithms to be used, up to the verification of the results thus obtained. In doing so, after examining the choices made in previous studies on the subject, we have opted to use $XGBOOST$, a relatively recent algorithm that is providing excellent results in a wide variety of scenarios from regression to classification, up to becoming the "to go" choice to achieve victory in Machine Learning competitions. The second algorithm chosen is *Random Forest*, one of the most used and most widespread algorithms for the fight against fraud. The last algorithm used is an *artificial neural network*, which gives a different vision of the problem and manages to capture different aspects compared to the previous algorithms.

- In **Chapter 1** we will introduce the problem of fraud concerning transactions, illustrating the functioning of a typical *Fraud Detection System* and the complications that characterize its implementation.
- In **Chapter 2** some background knowledge in the Machine Learning field necessary to understand the thesis work will be illustrated. We will show some choices made in other studies regarding fraud detection.
- In **Chapter 3** the architecture of the proposed system will be outlined, giving reasons for the choices made.
- In **Chapter 4** the salient points of the implementation of the system will be shown and we will proceed with the analysis of the results obtained by the system we propose.

- In **Chapter 5** we summarize the study conducted, highlighting the main results obtained and outlining the way for possible future developments.

Chapter 1

The Problem of digital frauds

In this chapter the problem of digital fraud will be introduced, highlighting the characteristics of the creation of a Fraud Detection System.

1.1 Introduction to the problem

A 2017 report from the European Central Bank highlights some interesting aspects about the payment methods preferred by European users in the previous year. Cash is still the most used payment method in Europe, accounting for 79% of transactions and 59% of total spending in 2016 [1]. Nonetheless, most citizens say they prefer credit cards as a payment method. This apparent discrepancy may arise from the fact that cash is used more frequently for low-value transactions, such as the daily purchase of food items, as well as frequenting businesses such as bars, restaurants and tobacconists. However, these low-value transactions constitute 2/3 of the total expenditure on the territory. However, it is clear that the users interviewed tend to remember more transactions of a higher amount, typically completed by credit card.

Regarding the subdivision of the use of the different payment methods in relation to users, it is interesting to note that it is possible to find substantial variations depending on the country of origin. Cash is the most widely used payment method in Southern Europe, as well as in Germany, Austria and Slovenia, where it was used for 80% of transactions. On the contrary, in countries such as

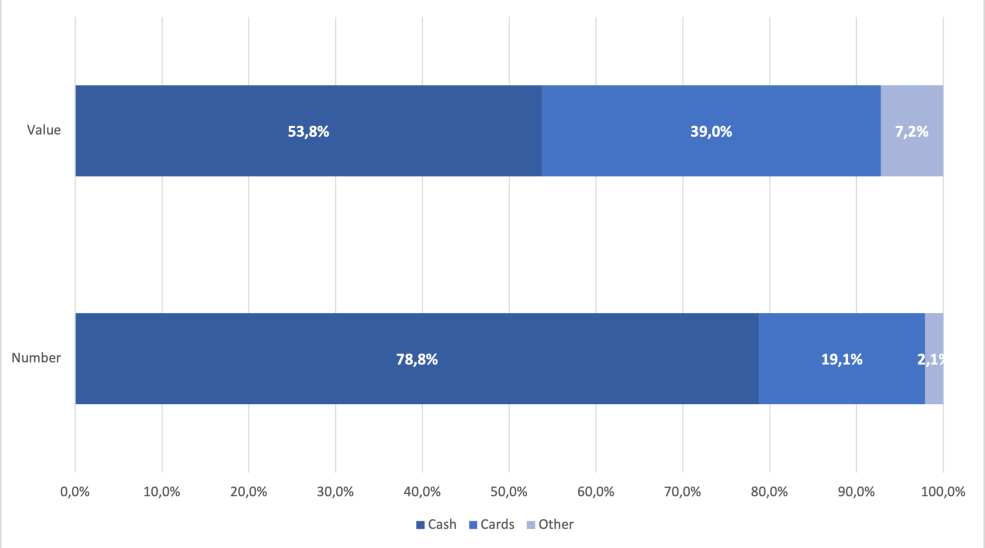

Figure 1.1: Percentage of use for the different types of payment, by number and total value [1]

the Netherlands, Estonia and Finland, the use of cash constitutes only 45-54% of total transactions. Estonia, France and Finland also have a relatively low impact of cash on the total amount of transactions, corresponding to only 33% of spending in 2016. However, while acknowledging the preponderance of cash use in Europe, it is important to note that for most consumers the speed of the transaction is a crucial aspect in the choice of the payment method. With this in mind, it is necessary to take into account the increase in the spread of POS and cards with contactless support, as well as the increasingly widespread use of systems such as Apple Pay and Google Pay, which allow you to pay by smartphone and smartwatch. The immediacy of using these devices, which we always have with us, has the potential to significantly decrease the use of cash payment even for small amounts in the coming years. This is even more true if we consider that 81% of payments made amount to less than €25, a threshold below which most cards do not require a PIN if used in contactless mode. The hypothesis on the increase of contactless transactions in the near future finds further support in the data issued by Mastercard in May 2020 [4], in which it is highlighted that 78% of transactions carried out on the Mastercard circuit in Europe now take place via contactless technology, with 42% of users say their use of cash has decreased due

to the SARS-CoV-2 pandemic. In particular, 17% of users said they no longer use cash under any circumstances. However, the rise in the adoption of contactless payments does not appear to be going to fade with COVID-19, given that 72% of users say the end of the pandemic will not push them to resume using cash. Although we want to limit ourselves to the data provided in the ECB report already mentioned above, by examining the sum spent with the various payment methods rather than the number of individual transactions, it is possible to note that already in 2016, payments by credit card made up 39% of the total amount. Therefore, the importance of making these payment methods as safe as possible is clear, in order to protect both the end customer and the bank that issued the card. The term *credit card fraud* is used to indicate situations in which a dishonest entity uses the credit card for its own needs without the knowledge of the legitimate owner. Credit card fraud can occur through the theft of the physical card, or with the theft of sensitive data such as card number, CVV code, type of card and other information of this type [5]. In a report by the European Central Bank [2], published in 2019, we can find specific information on the trend of expenses made by credit card also in relation to fraud. In 2018, 4.84 trillion euros were spent by credit card in the SEPA area, and of these 1.80 billion were resulting from fraudulent transactions. This marked an increase in the amount of fraud compared to 2017 of 0.002 percentage points, bringing it to 0.037%. This means that, for every € 100 of total transactions, 3.7 cents of euro are to be attributed to illegal transactions. The trend of fraud therefore seems to have increased again, after having seen a decrease between 2015 and 2017. In this period, the highest percentage of fraudulent transactions was found in 2015, when it reached 0.042%, while the lowest it reached 0.035% in 2017. Of the fraudulent transactions, 79% is represented by CNP (Card Not Present) transactions, up from 75%. These are mainly payments made via the internet. One of the reasons why CNP fraud is so widespread is that for those who process the payment, when the data entered is correct, it is difficult to check that it is really the cardholder who is requesting the transaction in question. It is particularly interesting to note that, between 2017 and 2018, the amount in euro of fraudulent transactions increased by 13%, com-

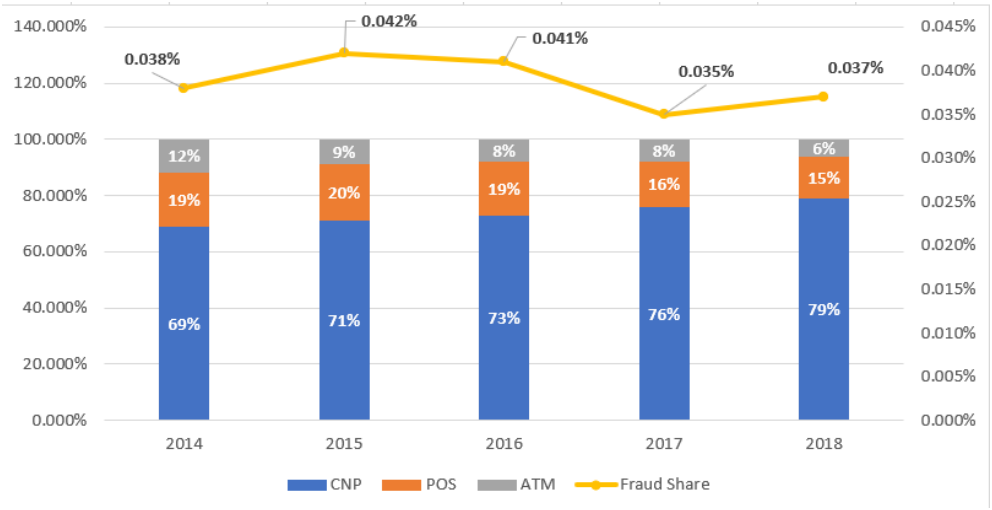

Figure 1.2: Left Axis: Total volume of fraud in percentage per platform. Right Axis: Volume of fraud as a percentage of total transactions [2]

pared with an increase of 6.5% in total transactions. This means that fraud has seen double growth in the number of transactions, which makes this particularly worrying. Banking institutions are clearly very active in trying to prevent fraud. An example of such efforts is represented by the tokenization of credit cards. Tokenization is a security measure that consists of replacing all sensitive data with unique identifiers called tokens. This can be useful, for example, during an online payment, or to use 'virtual' cards stored on our smartphones [2]. The generation of such tokens adds an additional degree of security, as it is possible to verify that the token is used only in the context for which it was generated (e.g., online payment on a particular app, or POS payment via NFC chip). Since a token, even if it were to be leaked in a data breach, does not constitute useful information once its task has been completed, banking institutions are able to use this system also to protect themselves and customers from part of the risks deriving from the loss of sensitive information. The use of tokenization alone is not able to eliminate all fraud, since it cannot cover all cases of use of credit cards but can only be adopted in specific cases such as those mentioned above. For this reason, every banking institution has systems to prevent and detect fraud attempts.

Chapter 2

Machine learning algorithms

This chapter will provide information on Machine Learning algorithms. We will focus in particular on the choices made regarding the Machine Learning algorithms chosen for our thesis work.

2.1 Supervised and unsupervised learning

A clarification to be made, before getting to the heart of the choices made in the literature, is that one typically chooses to approach Fraud Detection as a **binary classification problem**, i.e. each individual transaction is examined and characterized as fraud or as genuine, without an intermediate possibility. It is also **supervised learning** as, for each transaction that makes up the dataset, a label will be available to indicate its class. This was the approach used in almost all of the literature examined. An alternative solution [6], but much less used, is to create an "average" for the various transaction features, and proceed in **outlier detection** mode [7]. This means going to report as possible fraud all transactions that deviate, based on pre-established metrics, from what is considered normal.

2.2 Algorithms Overview

In order to better understand what is illustrated in section 2.3, a brief description of the Machine Learning algorithms most used in the literature for classification problems will be reported below.

Logistic Regression (LR) It is a widely used technique for binary classification. It takes its name from the *logistical function* that forms its foundation. It is a *sigmoid*, i.e. S-shaped function (see figure 2.1). This type of function is used to map any real number in the range between 0 and 1, without however assuming one of these two values in any case [8].

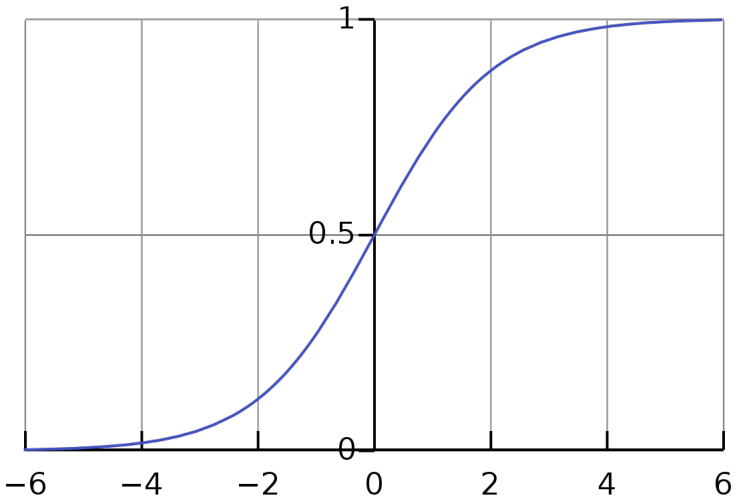

Figure 2.1: Logistic Curve. Source: wikipedia.it

Decision Trees (DT) Decision trees are a prediction model that creates a mapping between observations and their possible consequences [6]. Basically, *questions* are extracted from the data to be answered with one or more of the sample features. The answer to these questions will guide the model along the tree until it reaches a classification. An interesting aspect of the decision trees is that it is possible to visualize them, understanding what is the criteria used to arrive at the classificationn [9]. A very simple example of a decision tree is shown in figure 2.2.

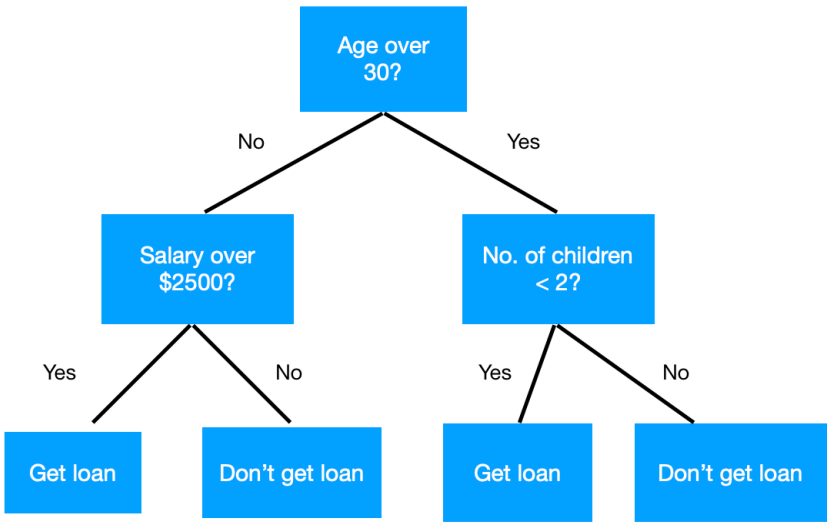

Figure 2.2: Example of Decision Tree. Source: eloquentarduino.github.io

Random Forests (RF) Random Forests are an *ensemble* method, that is, they are composed of other learning algorithms, in the specific case of Decision Trees. They operate by constructing multiple Decision Trees during the training, and then giving an *average* of the classifications established by them as output. This can be thought of as a voting system (see figure 2.3). Taking for example a random forest made up of 100 decision trees, if 30 trees considered a sample to be a fraud, and the remaining 70 considered it genuine, the random forest will output the label "genuine transaction" for the specific sample considered.

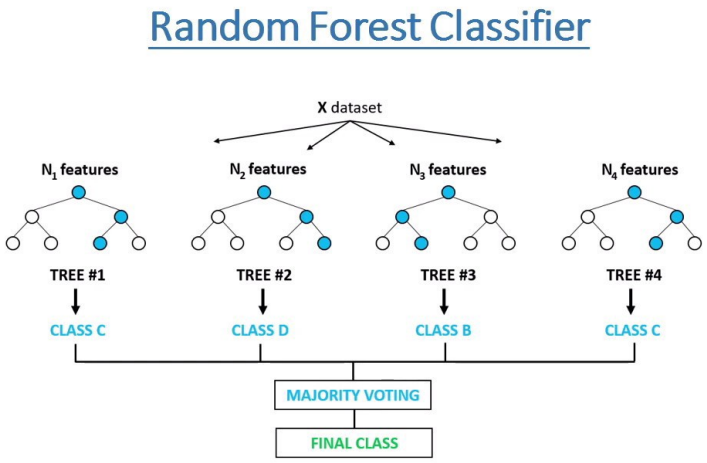

Figure 2.3: Simplified example of Random Forest. In this specific case the predicted class will be C. Source: `https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6`

XGBoost XGBoost implements machine learning algorithms under the *Gradient Boosting* framework. It provides a parallel tree boosting (also known as $GBDT$, GBM) that solve many data science problems in a fast and accurate way.

K-Nearest Neighbors (KNN) K-Nearest Neighbors assumes that similar elements exist close to each other [10]. This means that if we have an unlabeled sample, it is reasonable to assume that it is of the same class as the nearest K samples (see figure 2.4). Of course, the concept of *distance* can be conceived in different ways depending on the type of problem we are trying to solve, but the

classic *Euclidean distance*, that is, the straight line connecting the two points, is also frequently used.

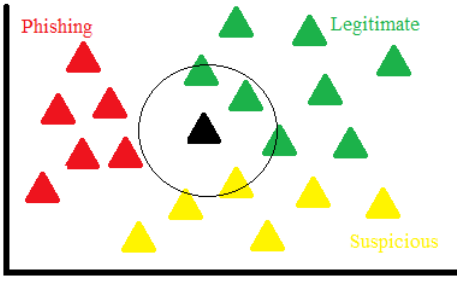

Figure 2.4: K-Nearest Neighbors. Source: `https://www.researchgate.net/figure/A-k-nearest-neighbor-KNN-classifier-KNN-is-explained-as-follows_fig1_318096864`

Neural Networks (NN) This is a technique that tries to mimic the functionality of the human brain using a set of interconnected nodes. They have the advantages of being adaptive and generating robust models [6].

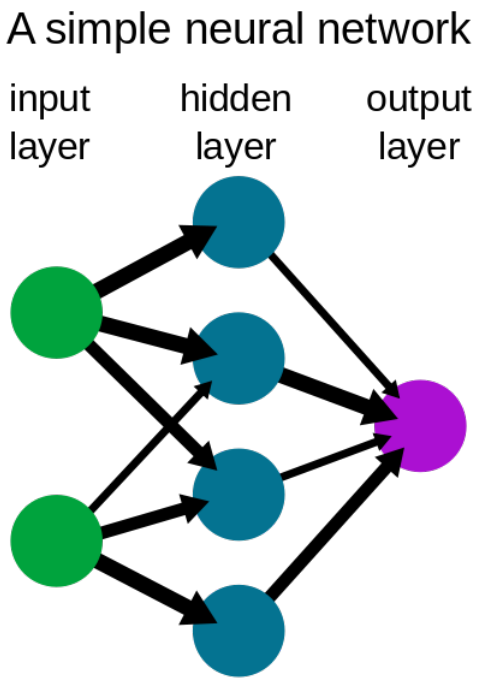

Figure 2.5: Neural network structure. Source: Wikipedia.it

Long Short-Term Memory (LSTM) Long Short-Term Memory (LSTM) Long Short-Term Memory Networks are a type of Recurrent Neural Network (RNN). The recurrent neural networks are distinguished from the traditional ones, called *feed-forward*, for the possibility of using internal states to *remember* contextual information relating to previous iterations, for an amount of time that can vary depending on the input data and their weights [11]. In particular, LSTM solves some characteristic problems of RNN, including *vanishing gradient* and *exploding gradient* [12]. These two problems occur, respectively, when the influence of a certain input on the hidden state decays or increases exponentially [13].

Multilayer Perceptron (MLP) Multilayer Perceptron (MLP) Perceptron multilayer is a class of artificial *feed-forward* neural networks consisting of at least three levels of nodes: an input level, a *hidden level* and an output level. Each node (except the input node) is a *neuron* that uses a non-linear activation function. MLP uses a mechanism called *backpropagation*. This mechanism allows, in essence, to recalibrate the weights assigned to the neurons that make up the network based on the error rate of the previous iteration. This *-tuning* makes it possible to achieve relatively low error rates, making the model reliable and generalized [14]. Multilayer perceptrons are sometimes referred to as *vanilla neural networks*, especially when they have a single hidden level [15].

Naive Bayes(NB) A Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem (2.1). Using Bayes' theorem, one can calculate the probability that event A will occur, knowing that event B has occurred. It is called *naive* because it is based on the assumption that the features are independent, i.e. that the presence of specific feature does not affect the others [16].

$$P(A \mid B) = \frac{P(A \mid B)P(A)}{P(B)} \tag{2.1}$$

Bayesian Belief Networks (BBN) Bayesian Belief Networks are a type of probabilistic chart model. This means, in short, that they are a way of representing a probabilistic model with a graph structure. Each node in the graph represents one or more random variables, while the links express probabilistic relationships between these variables [17]. A Bayesian Belief Network captures the joint probability of all events represented in the model [18]. A simple example of a BBN is shown in Figure 2.6.

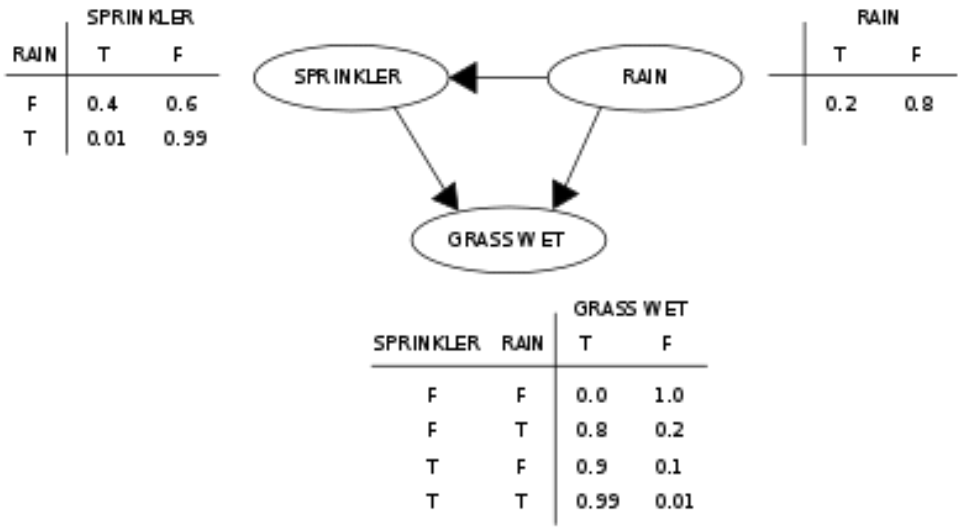

Figure 2.6: A simple Bayesian Network with conditional probability tables. Source: Wikipedia.com

Support Vector Machines Support Vector Machines are widely used for classification. The goal of this algorithm is to find a hyperplane in the N-dimensional space (where N is the number of features considered) such that the samples of the different classes are correctly separated. Intuitively, the goal is to build a hyperplane such as to have the maximum amount of separation between the samples of the two different classes and the hyperplane itself [3]. The number of dimensions of the hyperplane will depend on the number of features. If there were only 2 features, the hyperplane would be nothing more than a straight line (see Figure 2.7). If there are three features, it would be a plane instead. Clearly, having N features it becomes difficult to imagine a representation for the resulting hyperplane.

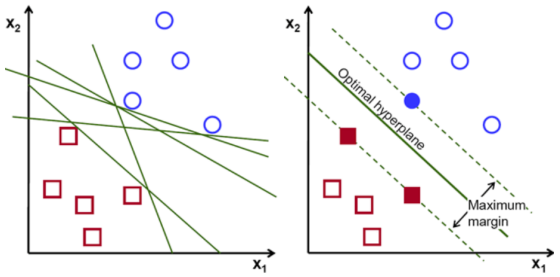

Figure 2.7: The hyperplanes needed to separate samples with two features are nothing more than straight lines [3]

2.3 Chosen Algorithms

In this section we will discuss more in depth about the 3 algorithms we used in our *hybrid model*.

2.3.1 Random Forests

Random forest algorithm was our first choice for his popularity in the past publications on implementation of Fraud Detection System. An overview of the most used methods to extract information from data in order to make a classification was presented in 2010 by Ngai et al [6]. The authors took into consideration 49 research articles published between 1997 and 2008, classifying the objectives they set themselves and the techniques used to achieve them. From this survey it was clear that classification is the most frequent application for data mining in the financial field, being the goal set for 61.2% of the studies examined. As regards the most used techniques, a prevalence of logistic models was found (in 21.3% of the studies) followed by neural networks at 13.3%, and by Bayesian networks and decision trees, both at 6.7%. However, in the years following those considered in the survey mentioned [6], other techniques that began to gain great traction in their ability to classify with excellent performance were random forests and support vector machines. A comparison between the latter two methods with the more traditional logistic regression was conducted by Bhattacharyya et al. in 2011 [19]. This study highlighted how random forests present a superior overall performance. The same result was also reported by Whitrow et al , who in their study showed that random forests have better results, as well as compared to Support Vector

Machines, Logistic Regression, and K-Nearest Neighbors [20]. Varmedja et al. [5] in turn conducted a comparison between various machine learning algorithms, including Logistic Regression, Random Forest and Multilayer Perceptron, in order to identify among them the most suitable for Digital Fraud Detection. Using metrics such as recall, accuracy and precision, were able to establish that once again it is the Random Forests that give the best results in the classification of transactions. Another interesting comparison was made by Jurgovsky et al. in 2017 [21]. This study notes that, although the models using Random Forests and Long Short-Term Memory have comparable results from the point of view of the number of detected frauds, the two models seem to detect fraud of different types, agreeing on 50.8% of true positives. It is also highlighted that, in any case, the Random Forests have some advantages including the lower tendency to overfitting and the greater ease in interpreting the reasons behind the classification of the samples. **Random Forests therefore seem to be the most suitable classification algorithm for Digital Fraud Detection.**

2.3.2 XGBoost

The second chosen algorithm is XGBOOST, acronym indicating **X**treme**G**radient**BOOST**ed Trees. It is a relatively recent machine learning algorithm, which became known in 2016 thanks to the paper by Chen, Guestrin called *XGBOOST: A scalable tree boosting system* [22]. It is an *ensemble* method, like random forests, and like random forests, it uses decision trees as an algorithm within it. However, the way these trees are built is fundamentally different. Random forests build deep, specialized trees, which work very well with a subset of the data, and then combine the predictions of individual forest trees. XGBOOST works differently. IT builds *weak learner* decision trees that are short and simple. To do this the model proceeds by building the trees up to a predetermined maximum length, and then simplifying them backwards with a process called *tree pruning*. The trees are built iteratively, and each tree will try to improve on the attributes that may have caused a misclassification in a previous iteration. XGBOOST is widely used for both regression and classification problems,

and has become a very popular choice in various applications, as well as being frequently the winning algorithm in Machine Learning competitions. XGBOOST offers several advantages over many other Machine Learning algorithms:

- Use regularization to avoid overfitting
- Handles missing values automatically
- Works in parallel on multiple threads / cpu / machines
- Allows you to save the model and then resume training it later

2.3.3 Artificial Neural Network

The third and last algorithm chosen for our hybrid algorithm is the Artificial Neural network. An artificial neural network is a mathematical model composed of artificial "neurons" based on biological neural networks, used to attempt to solve engineering problems of artificial intelligence [23]. It can be implemented both by software programs and by dedicated hardware. In most cases an artificial neural network is an adaptive system that changes its structure based on external or internal information flowing through the network during the learning phase. In practical terms, neural networks are nonlinear structures of statistical data organized as modeling tools. They can be used to simulate complex relationships between inputs and outputs that other analytic functions cannot represent. An artificial neural network receives external signals on a layer of input nodes (processing units), each of which is connected with numerous internal nodes, organized in several layers. Each node processes the received signals and transmits the result to subsequent nodes. In general, neural networks are made up of three layers (which, however, can involve thousands of neurons and tens of thousands of connections):

- the **input layer** (I - Input): it is the one that has the task of receiving and processing the input signals, adapting them to the demands of the neurons in the network;

- the **hidden layer** (H - Hidden): it is the one that is in charge of the actual processing process (and can also be structured with more column-levels of neurons);
- the **output layer** (O - Output): here the results of the processing of the H layer are collected and adapted to the requests of the next level-block of the neural network.

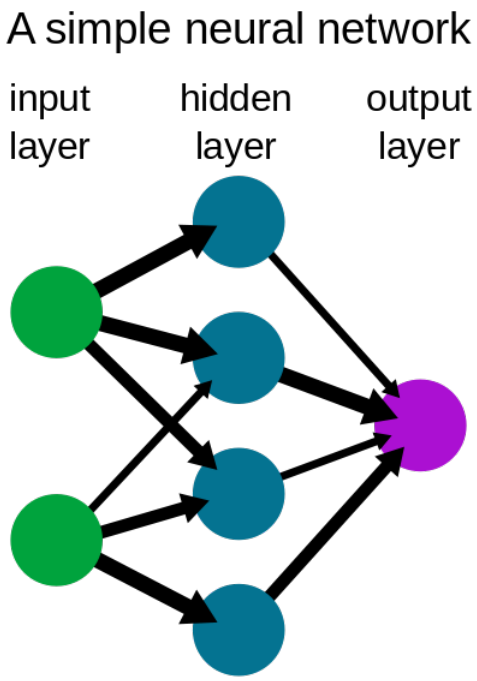

Figure 2.8: Neural network structure. Source: Wikipedia.it

The first theoretical model of a rudimentary artificial neuron sees the light in 1943. To propose it is a couple of scientists, McCulloch and Pitts. The two describe an apparatus capable of receiving n binary input data in each of its elements, followed by a single output data for each [24]. This machine is able to work on elementary Boolean functions, and only on those (Figure 2.9).

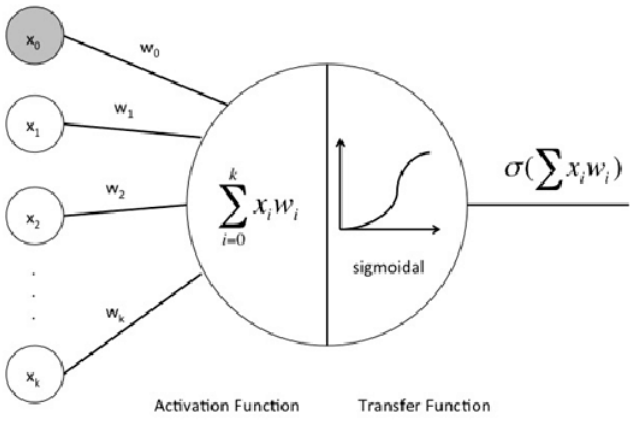

Figure 2.9: Artificial Neuron of McCulloch and Pitts. Source: researchgate.net

In 1949 D. O. Hebb hypothesized the possibility of instructing machines with a learning that emulates that at the basis of human intelligence. In 1958 the first neural network was proposed by Rosenblatt: *Perceptron*. Rosenblatt's Perceptron has a layer of input nodes (artificial neurons) and an output node. Synaptic weights (a weight indicates the strength of a connection between two nodes) are dynamic, allowing the machine to learn, in a roughly similar, though much more basic, way to that of biological neural networks. The model is feedforward: the impulses propagate in a single direction, forward. Its scope is very limited. It consists in recognizing shapes, classifying them into two separate groups, and in calculating simple functions. The next step is the multilayer Perceptron (MLP). Inside, between the input and output nodes there is a hidden layer, where the processing of information from the input layer takes place, which is then sent to the output node. It is a non-linear feedforward network: the connections to and from each single node are multiple. Thanks to this architecture, the MLP can compute any function. Werbos, in 1974, describes in his doctoral thesis how to set up the learning of an MLP. His work is then resumed and perfected by Rumelhart, Hinton and Williams, who in 1986 elaborated the famous Error Back-Propagation, with which we enter the present, being still used today. EBP allows you to perfect the machine learning of a neural network in subsequent stages. It is implemented by changing the weights of the connections between nodes that do not produce the optimal output, until the latter is obtained. No less important,

in this sense, is Hebb's previous work on reciprocal connections between neurons. Hebb postulates that their weight must increase only in case of convergence between the two pre- and postsynaptic values, hence the "Law of Hebb":

"If a neuron A is close enough to a neuron B to contribute repeatedly or permanently upon its excitation, then a process of growth or metabolic change takes place in both neurons such that the effectiveness of A in exciting B is increased" [25]

With MLPs and EPBs, machine learning now finds some fields of practical application. We are in the 90s. Meanwhile, neural networks with feedback architectures are also implemented (Hopfield networks, named after the physicist who proposed the model in 1982). In such architectures, information between nodes travels in any direction: forward, backward and between nodes of the same layer. The field of applications is further expanded. Also in these years the neural network proposed by Elman in 1990 was designed. This is also a recurring network model (bidirectional), but with the variant that a group of nodes is added to the classic MLP structure with the purpose of storing the information of the previous configuration of network values. Thanks to this modification, the Elman network proves advantageous in the calculation of time sequences. In 1992, however, Kohonen designed a type of neural network with both feedforward and feedback architecture. Its peculiar feature is the ability to modify the configuration (map) of its nodes based on the weight they assume as inputs are provided. Knots with similar weights approach; those with very different weights move away. The Kohonen network is also known as the Self-Organizing Maps (SOM) network. The evolution of neural networks continues with the recent technology adopted by IBM, through which a neural network based on phase change materials has been developed. The hardware of this architecture uses alloys such as ammonium germanium telluride, which have the interesting property of assuming two different states: crystalline (homogeneous spatial configuration) and amorphous (poorly defined spatial configuration). In phase-change neurons, electrical impulses are capable of causing crystallization of the material, ultimately triggering its firing (activation). Well,

this is analogous to what happens in nerve cells. For now, IBM's artificial neuron allows information to be written to it but does not store it stably, but it is certain that its functioning is as similar to the emulation of a human brain.

Chapter 3

Solution proposed

This chapter discusses the decisions made for the design of a Fraud Detection System. The choices made regarding the preprocessing of the dataset as well as the selection and training of the Machine Learning model to be used will be examined. The algorithms chosen for the classification are XGBOOST, Random Forest and Artificial Neural Network. The implementation of the proposed system will then be detailed in Chapter 4.

3.1 Development Environment

The thesis work was developed using the Python language, a very popular choice in Data Science and Machine Learning. This language provides a large number of libraries that make it extremely powerful and flexible, while maintaining great ease of use. In particular, the PyCharm IDE was used, in which it was possible to use the Anaconda platform [26]. Very popular libraries and packages have been used and particularly suitable for our purposes, including:

NumPy Open source project that aims to allow numerical computation in Python. It adds support for large arrays and multidimensional arrays, as well as a large collection of mathematical functions that allow you to operate efficiently on these data structures [27].

Scikit-learn Open source machine learning library [28]. Contains classification, regression and clustering algorithms. It also contains all the key metrics that can be used to train and evaluate models. It is designed to work with other libraries such as NumPy and SciPy.

Pandas Pandas Open source library for data analysis and data manipulation [29]. It offers data structures and the ability to manipulate numerical tables and time series. It is mainly used for data analysis, allowing you to import datasets from files of different formats including JSON and CSV. It allows operations on data such as the union and modification of datasets, as well as the cleaning of the same.

Keras Open source library for machine learning and neural networks, written in Python [30]. It is designed as an interface at a higher level of abstraction than other similar lower level libraries, and supports the TensorFlow, Microsoft Cognitive Toolkit (CNTK) and Theano libraries as a back-end. Designed to allow rapid prototyping of deep neural networks, it focuses on ease of use, modularity and extensibility.

MatPlotLib MatPlotLib Library suitable for creating static, animated and interactive graphics [31].

3.2 The Dataset

Before talking about the dataset, it must be premised that this thesis project is actively in use in the company for which I currently work. For this reason, I will have to omit some information about the dataset and be very generic. The dataset is made up of more than 30 features relating to transactions and customers. A 4-month fraud segment was considered for the thesis work. The dataset is made up of a number of transactions in the range of thousands.

3.3 Principal Component Analysis

Principal Component Analysis (PCA) is one method used to reduce the dimensionality of large datasets, transforming a set of variables into a smaller variable that retains most of the information contained in the initial set of variables [32]. This process makes it easier to explore, analyze, and visualize data, as well as make it easier and faster for machine learning algorithms to learn from it. A fundamental step in the PCA is standardization, which consists in bringing the range of values within a predetermined range. The first operation we perform is to remove the duplicate rows from the dataset. All transactions that have exactly the same values for all features are then removed. In our case study, some tests I conducted verified that this method degraded the performance, so it was not taken into consideration for the implementation of the hybrid algorithm.

3.4 Management of Class Imbalance

Class imbalance can be substantially tackled with two diametrically opposed approaches, *oversampling* and *undersampling*. Clearly, each of these approaches can then be implemented with different techniques, but which share the same basic ideas. Before examining the differences between oversampling and undersampling, we highlight that the goal of both approaches is to create a dataset that is more balanced between the majority and minority class.

3.4.1 Oversampling

Oversampling basically consists in going to **replicate** champions of the minority class in some way, so as to bring them closer in number to the champions of the majority class. A typical, although very trivial, way of oversampling is to duplicate samples of the minority class as many times as necessary to balance the dataset. This oversampling methodology, although in the case of very unbalanced datasets it can avoid that the minority class is considered as simple *noise* in the data, it can however lead to a lower generalization and therefore to worse

performance of the model in real scenarios.

3.4.2 SMOTE

A potentially more effective and widely used oversampling technique is the Synthetic Minority Oversampling Technique, or **SMOTE**. This technique consists in creating *synthetic* samples belonging to the minority class, which are however in some way similar to real samples [33]. The generation of synthetic samples in SMOTE takes place as follows:

- A sample from the minority class is chosen
- Its k-neighbors are identified (typically $k = 5$)
- You choose n of these k samples (depending on how much you need to balance the dataset)
- The difference between the feature vector of the selected sample and the closest sample is calculated
- The difference obtained is multiplied by a random number between 0 and 1, and the result obtained is added to the vector of the features in question.

The final effect will be that the models that will train on these augmented datasets will be more generalized, and more effective in real scenarios. What makes SMOTE superior to simple oversampling is that we are actually going to add *new information*, rather than replicating what is already in the dataset.

3.4.3 Undersampling

Undersampling consists in *ignoring* a portion of the samples of the majority class. In several cases this can improve the performance of the model. However, it is intuitive to understand how, if it is necessary to eliminate many samples to obtain balance in the dataset, an important loss of information potentially useful for training the model is inevitable.

In our study we have adopted the undersampling method.

3.5 Metrics

The quality of the classifications produced is of fundamental importance. In our scenario, the "fraud" will be the positive class, as it is the one, we are particularly interested in distinguishing. From a formal point of view, we will assign label $'1'$ to transactions identified as frauds, while those deemed genuine we will assign label $'0'$. When the classifier predicts the class for a transaction, we can run into four possible scenarios:

1) True Positive (TP) A *true positive* occurs when a sample of the positive class (in our case, fraud transactions) is correctly classified. In the context of Fraud Detection, a true positive is a fraud that has been correctly identified by the model.

2) True Negative (TN) A *true negative* occurs when a sample of the negative class (in our case, genuine transactions) is correctly classified. In the context of Fraud Detection, a true negative is a genuine transaction that the model has recognized as such.

3) False Positive (FP) A *false positive* occurs when a sample of the negative class (in our case, genuine transactions) is not correctly classified. In the context of Fraud Detection, a false positive represents a "false alarm", that is, a transaction that is marked as fraud although it is not. It is a scenario that is very important to avoid, for reasons that will be explored later.

4) False Negative (FN) A *false negative* occurs when a sample of the positive class (in our case, fraud transactions) is not correctly classified. In the context of Fraud Detection, this corresponds to a fraud that goes unnoticed, so it is by far the most serious scenario we can face.

Clearly therefore, True Positive and True Negative represent correct classifications, while False Positive and False Negative represent classification errors.

Confusion Matrix An immediate way to depict the effectiveness of the classifier is to view the results on a *confusion matrix*, of which we can see an example in Figure 3.1. The rows of the confusion matrix represent the predictions of the model, while the columns represent the real membership class of the samples. We will therefore have on the main diagonal the correctly classified samples, while on the secondary diagonal those for which an error has been made. In the specific case of the example shown, the color of the box will provide a further way of immediately identifying the number of samples for each category, with darker colors symbolizing a higher number of samples.

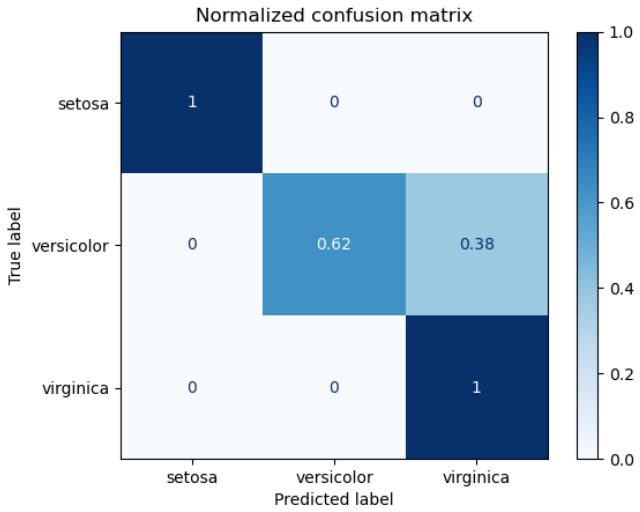

Figure 3.1: A simple example of confusion matrix

Metrics Having introduced these concepts, we can present some units of measurement for the goodness of the model.

Specificity is a measure of how effective the model is in avoiding false alarms, that is, in reporting as fraud only the samples that really are. Specificity is defined as

$$Specificity = \frac{TN}{FP + TN} \tag{3.1}$$

Another case of a metric that would not do for us is the *precision*, that is the portion of true positives identified on all the samples identified as positive. Precision is defined as

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

A much more sensible metric might be *Sensitivity*, or *recall*. Recall is the number of positives correctly identified out of the total number of positive samples present in the dataset. Recall is defined as

$$Sensitivity = \frac{TP}{TP + FN} \tag{3.3}$$

The importance of Recall in Fraud Detection therefore appears clear, as it is intrinsically a type of problem in which the highest cost is associated with false negatives, that is, fraud that is not correctly categorized by the system. In some

contexts, false negatives and false positives can have very different costs. A striking example is that of medical diagnoses for diseases that can have serious health implications. In this case, it is preferable to have extra "False Positives", to be examined more carefully with further checks, rather than having a false negative that could cause a serious health risk to go unnoticed. In the same way, while implementing a Fraud Detection system we consider false negative a worse scenario than false positive [34]. A false negative becomes, in most cases, a direct economic loss for the customer (if the fraud is not detected), or for the merchant (if it is a case in which it is required to compensate the customer). This may prompt us to be very cautious about classifying a transaction as genuine, accepting an increase in false positives as a trade-off. However, we must avoid making the mistake of considering false positives as a negligible problem. In fact, whenever a transaction is reported as suspicious by the algorithm, manual intervention by investigators would be required. These would have the task of verifying, based on their experience and possibly after a telephone contact with the customer, the genuineness of this transaction. One disadvantage is the need to interact with the customer, potentially leaving a genuine transaction pending and causing inconvenience [21]. A second disadvantage that has already been mentioned above, and which is even more significant, is that expert teams have limited time and resources. For reasons of economy and efficiency, a bank or merchant will hire a team of investigators of limited size, and for reasons of time, each investigator will be able to verify only a small percentage of the alerts received. This means that false positives are, at best, a waste of investigators' time and resources. But it can also mean an increase in undetected fraud, as some reports corresponding to real fraud could get out of control due to the amount of false positives to check. To mitigate the problem, one could think of notifying investigators only of the n most suspicious transactions, where n is the number of transactions that the investigating team can verify in a day [35]. For this reason, some studies cite the accuracy obtained on the n transactions considered most risky as a classifier evaluation metric [36]. In the specific case of the aforementioned study, $n = 100$ was set, as according to the banking partner of the study this is a reasonable number of transactions to

be verified in one day for investigators.

We are interested in reducing false positives as much as possible, as they can waste time and resources for investigators. We therefore need a metric that allows us to take into account both false positives and false negatives, since ideally, we would like to minimize both. From this point of view, we can consider the *F1-Score* an excellent metric, being the harmonic mean of Precision and Recall

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.4}$$

However, Precision, Recall and F1-Score share the same problem: they just consider the *positive* class, in our case fraud. In fact, they use only three of the four values contained in a confusion matrix: TP, FP, FN, ignoring TN, ie the true negatives. Whether TN takes a value of 1, 100 or infinity, the three metrics mentioned will not vary. This means that we will have an asymmetry in evaluating the two different classes. In fact, if we were to consider fraud as a negative class and genuine transactions as a positive class, the same model would risk giving very different results [37].

A metric that instead, as we wish, takes into account both classes is the **Matthew Correlation Coefficient (MCC)**. This metric treats the real and predicted class as two variables, and calculates their *correlation coefficient*. The higher the correlation between the two variables, the more valid the prediction can be [38].

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.5}$$

As you can guess from the formula 3.5, the MCC will always assume values between -1 and 1. In particular:

- $MCC = 1$: $FP = FN = 0$, we are in the presence of a perfect positive correlation
- $MCC = 0$: The classifier behaves like a random choice
- $MCC = -1$: $TP = TN = 0$, we are in the presence of a perfect negative correlation, which means that the classifier is always wrong. Naturally in this case it is sufficient to invert the classifications to obtain MCC = 1

For the reasons illustrated and taking advantage of the fact that this metric is (as well as the others mentioned) present in scikit-learn, **the Matthew Correlation Coefficient and F1-score** will be our reference metrics. This means that we will use them as metrics to be maximized in the training of the hybrid algorithm.

Chapter 4

Implementation

This chapter will illustrate the implementation of the Fraud Detection System under study of this thesis. The choices illustrated in Chapter 3 will be implemented in code and, where necessary, some details will be deepened.

4.1 Premise

Before starting this chapter, it must be said that, as previously announced, this thesis project is protected by company secrecy. For this reason, it will not be possible to explain the results in the form of numbers, but they will always be converted as a percentage of the total transactions.

Furthermore, it is important to specify that the following operations have been tried:

- scaling the dataset
- add weighing parameters for the minority class
- hyperparameter tuning operations

However, due to the inherent nature of the dataset, some of these operations, which typically bring improvements, have worsened overall performance, so they have been not taken in consideration for further discussions.

4.2 Preliminary operations

The first operations to be carried out, which we will not focus on particularly, include the import of the main libraries used for the implementation of the system and the reading of the dataset. Part of the libraries used are described in section 3.1.

4.2.1 Libraries Import

The first operation is the libraries import.
This section refers to code at Appendix A.1.

4.2.2 Dataset Import

We start by importing the dataset and save in the variable $x_complete$ the contents of the columns from 0 to K, with K equal to the number of features. Then the information contained between column 1 to column $k-1$ is saved in x, as column zero contains the transaction identifier and is not needed for model training. Finally, the contents of column k are saved in y, containing the Boolean value that indicates whether the transaction is genuine or fraud. For our dataset we decided to use the undersampling process. At the beginning our dataset contained all the transaction during 4-month period. We kept all the frauds and we took a random sample of the genuine transaction. In the graph contained in the figure 4.1 it is possible to see the distribution in percentage between genuine transaction and fraud.
This section refers to code at Appendix A.2.

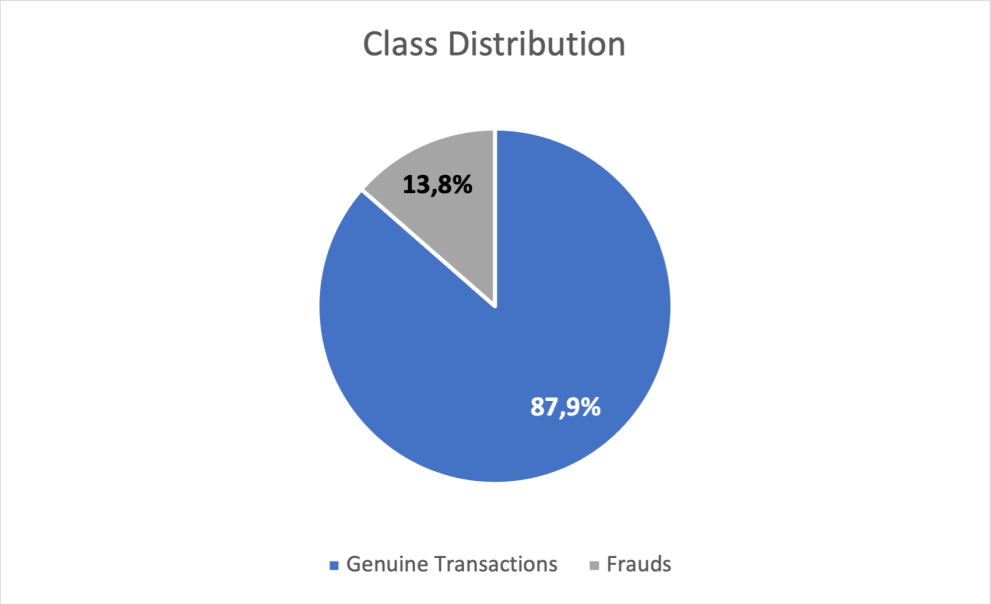

Figure 4.1: pie chart representing the distribution of classes.

4.2.3 Subdivision of the dataset

The next important step was the subdivision of the dataset into a *Training Set* and a *Testing Set*. This operation is necessary to ensure that the model carries out training and testing on two different portions of the dataset. If this were not the case, testing the model on the same data set it trained on would lead to extremely misleading results and absolutely not representative of the performance we might expect when, in real situations, the model will receive new samples that it had not previously encountered. We opted for an 80 : 20 split between training set and testing set. To complete this task I used a function already implemented in Scikit-learn library, called *train_ test_ split*:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state =
    ↪ 2, stratify=y)
```

4.3 Hybrid algorithm

As already mentioned above, the central objective of this thesis project is the development of a score capable of giving a criticality to transactions ranging from 0 to 3, where 0 is considered a genuine transaction, and 3 is considered a transaction with a high risk of fraud. This score is nothing more than the sum of the considerations of the 3 algorithms already detailed in section 2.3. So, if no algorithm considers the transaction fraud the resulting score will be 0, if an algorithm considers the transaction fraud the score will be 1 and so on. The use of multiple algorithms is very useful, especially in being able to prioritize the transactions that need the most attention in the shortest possible time. Below we will explain how the algorithms were implemented individually and how they were then combined to create a single score.

4.3.1 Random Forests and XGBoost

We implement the Random Forest and XGBoost algorithms and train the models with the dataset resulting from the *train_ test_ split* function. Finally, we will use the *dump* function to save the 2 trained algorithms. This section refers to the code at Appendix A.3

You can later use the saved algorithms with the commands reported at Appendix A.4

4.3.2 Artificial Neural Network Function

At Appendix A.5 you can find the implementation of the function for the neural network. It takes as input the same dataset as the previous algorithms, and a threshold. We then print the information regarding the results and metrics. Finally, the *plot_ accuracy_ loss* function is called.

4.3.3 Plot function

The *plot_ accuracy_ loss* function help to visualize the progress of the training. It will plot the model accuracy and the model loss.
You can find the code referring to this function at Appendix A.6.

Below we can find the results of the function:

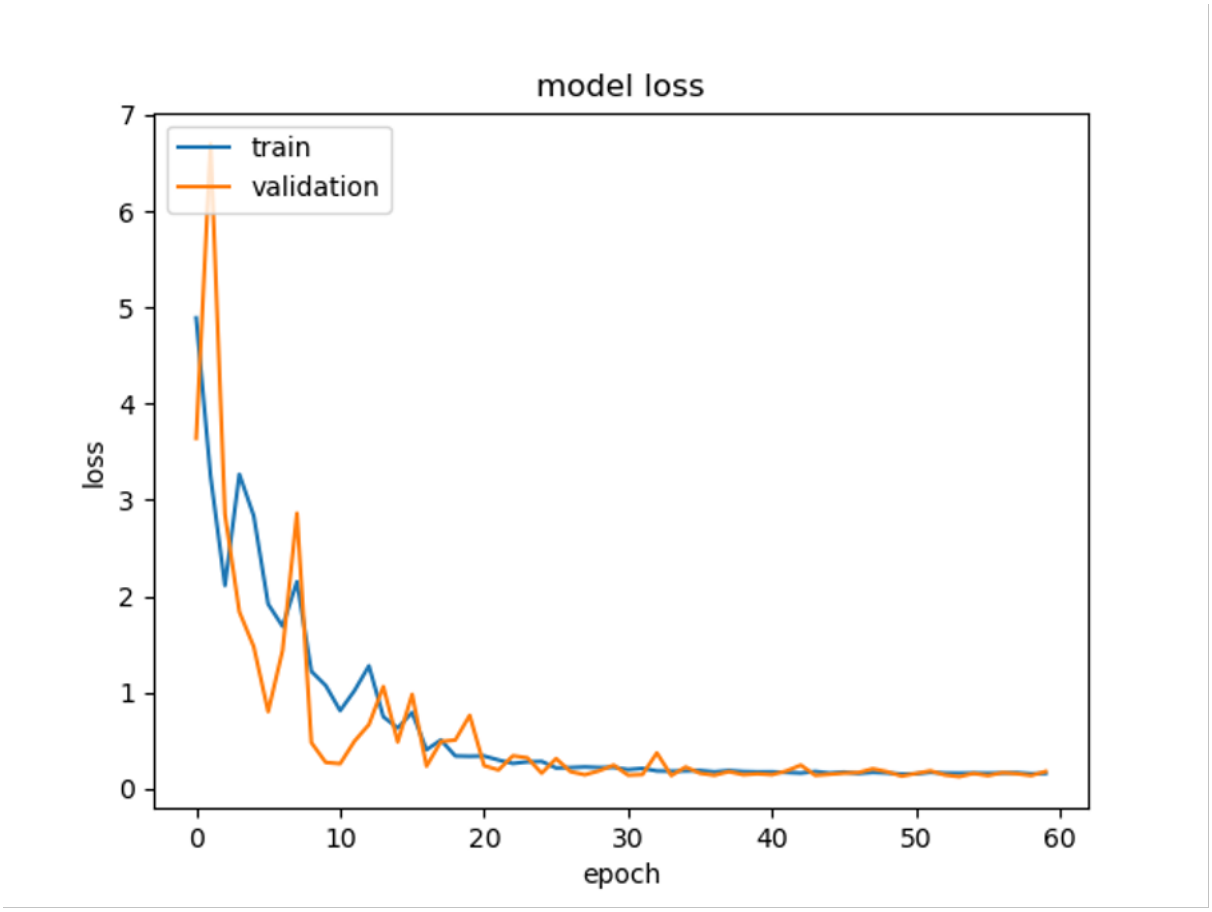

Figure 4.2: Graph representing Model Loss in relation to the number of epochs

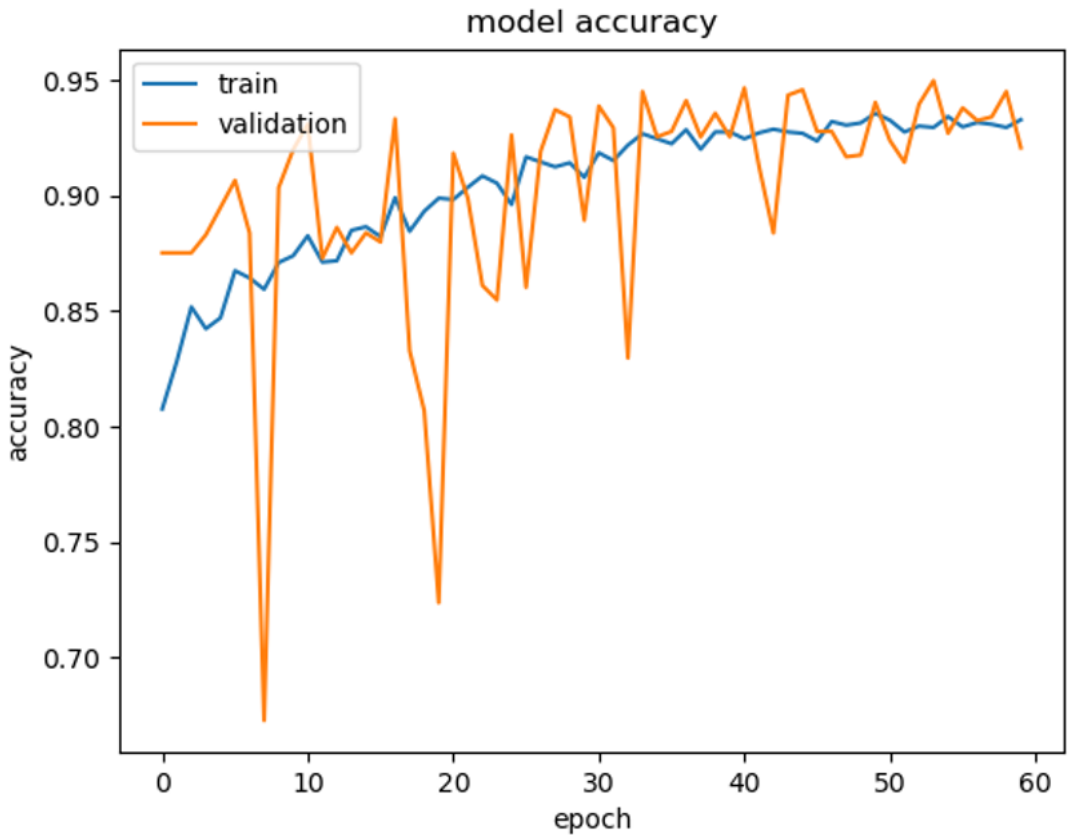

Figure 4.3: Graph representing Model Accuracy in relation to the number of epochs

4.3.4 Statistics function

Unlike what we did in the case of the neural network, for the random forest and for xgboost algorithms we have created a common function that calculates and prints the statistics and metrics. Finally, the plot_feature_importance function is called, which will be explained in the next subsection 4.3.5.

The implementation of this function is at Appendix A.7.

4.3.5 Feature importance Function

This function is called from *stats_and_featureplot*, and it is very useful to let us visualize the importance of each feature and how much the model takes them in consideration for the classification.
The implementation of this function is at Appendix A.8.

Below we will show the 2 different graphs for Random Forest and XGBoost, renaming the feature with a progressive number.

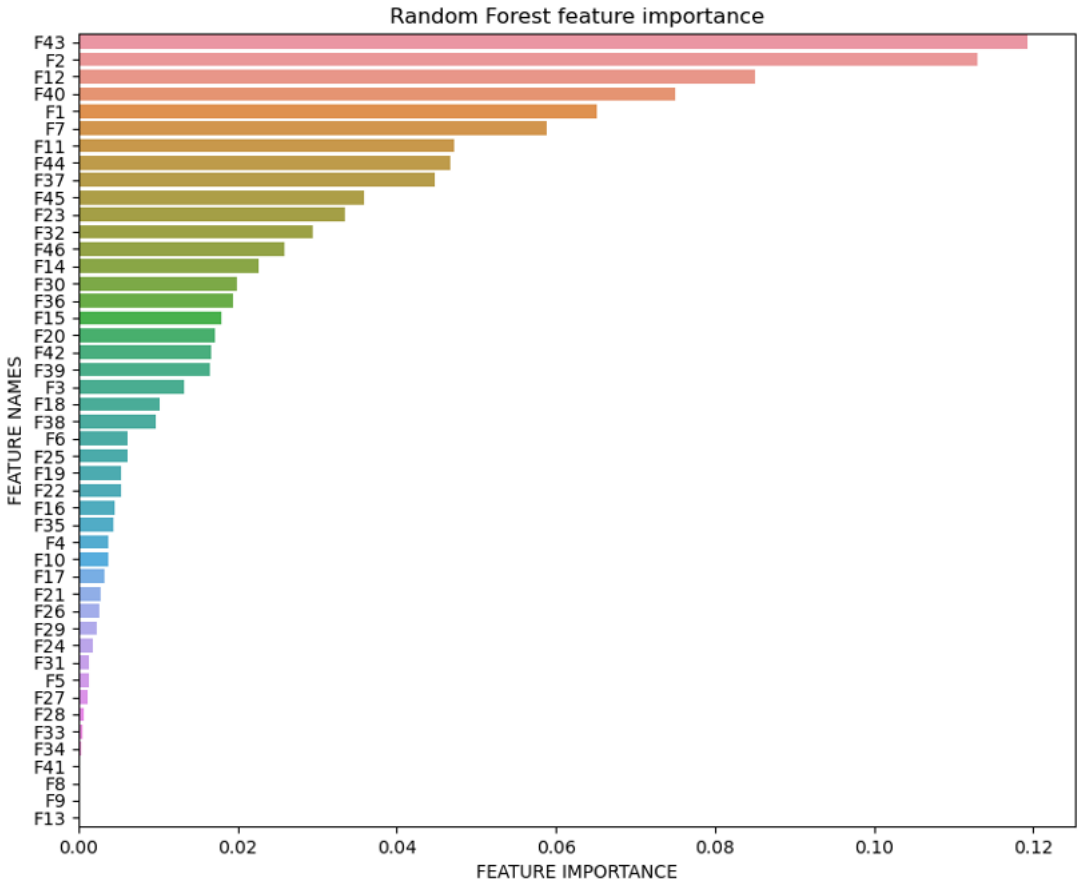

Figure 4.4: Representation of features in order of importance for Random Forest Algorithm

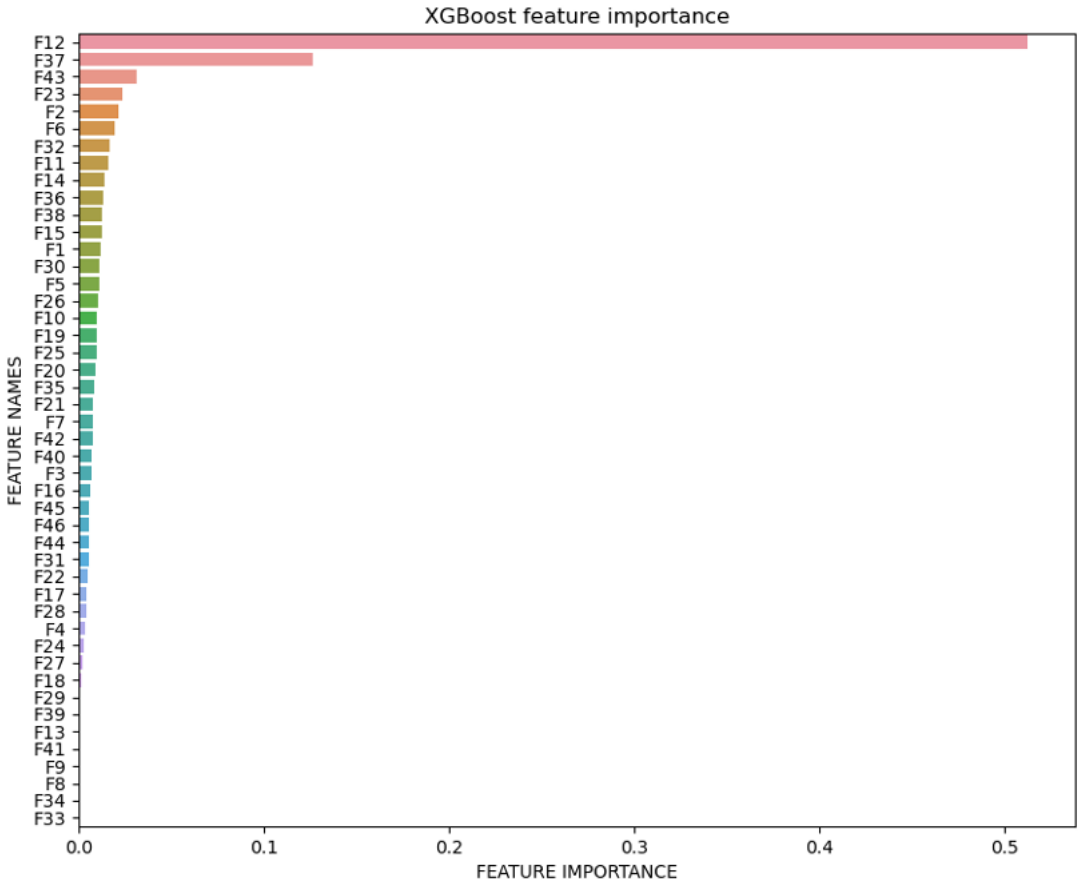

Figure 4.5: Representation of features in order of importance for XGBoost Algorithm

4.3.6 Score function

The last function we will discuss is the Score function. We create a dataframe with 4 Boolean columns, 3 of them regarding the classification of the 3 algorithms, and the last one indicate if it is a real fraud or not. The fifth column is the sum of the 3 algorithms decision, and it will be our score. In the last part of the function we will print all the stats regarding the HighScore (with 3 points), MediumScore (2 points), and LowScore (1 point). At the end we will save the dataframe in a csv file named ScoreFile.

The implementation of the function is at Appendix A.9.

4.4 Hybrid Algorithm vs Single Algorithms

In the last section of this chapter we will discuss the differences of results that exist in the case we use the algorithms individually and in the case of the hybrid algorithms we developed.

In the tables below it is possible to find the results of the individual algorithms and for the hybrid algorithm. The numbers have been multiplied for a random coefficient to preserve the secrecy.

	Neural Network	Random Forest	XGBoost
Total Frauds in Test Set	193	193	193
True Positive	109	167	174
False Positive	19	16	13
Missed Frauds	84	26	19
F1 - Score	0.82	0.93	0.95
MCC	0.66	0.87	0.90

Table 4.1: Chosen Algorithms Results

	Low Score	Medium Score	High Score
True Positive	14	62	104
False Positive	11	8	7
Total Alerts triggered	25	70	111

Table 4.2: Hybrid Score Results

As we can see from table 4.2 with the 3 score we achieved to classify correctly 180 frauds on 193 present in test set. The F1 - score for hybrid model is 0.95 and MCC is 0.89. In the chapter 5 we will discuss the results of this study.

Chapter 5

Conclusion and future developments

5.1 Conclusion

Before concluding, it must be said that this score is applied in a second level of skimming. First level is a proprietary system that automatically approves and rejects transactions.
This score was developed, trained, and tested considering only the transactions that were allowed by the previous security level. For this reason, the score looks for non-trivial fraud, which would have been a threat to the company. The results are:

- On alerts tagged with High score, the 94% are fraud and 6% are false positive. In our test this score takes 81% of total test frauds in the dataset.
- On alerts tagged with Medium score, the 71% are fraud and 29% are false positive. In our test this score takes 9% of total test frauds in the dataset.
- On alerts tagged with Low score, the 9% are fraud and 81% are false positive. In our test this score takes 7% of total test frauds in the dataset.
- 3% of total frauds in the dataset has not been found.

We can therefore say that our score starts with a high level of accuracy and progressively decreases. The number of false positives on the other hand are inversely proportional to accuracy.

5.2 Future developments

In the near future it will be important to add more features and try to eliminate features with zero impact on the models. It will also be interesting to find out if customizing features based on the algorithm can lead to performance improvements. The aforementioned developments will be needed to further improve accuracy, but above all to try to reduce false positives, which require valuable time from risk team members.

Appendix A

Appendix

A.1 Libraries Import

```
import numpy as np
import pandas as pd
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import classification_report
```

A.2 Dataset Import

```
dataset = pd.read_csv('dataset.csv')
x_complete = dataset.iloc[:, 0:k].values
x = dataset.iloc[:, 1:(k-1)].values
y = dataset.iloc[:, k].values
```

A.3 Random Forests and XGBoost train

```
RF_model = RandomForestClassifier()
RF_model.fit(x_train, y_train)
RF_y_pred = RF_model.predict(x_test)
joblib.dump(RF_model, "RF_model.joblib")
XGB_model = XGBClassifier(use_label_encoder=False)
XGB_model.fit(x_train, y_train)
XGB_y_pred = XGB_model.predict(x_test)
joblib.dump(XGB_model, "XGB_model.joblib")
```

A.4 Random Forests and XGBoost load

```
\begin{lstlisting}[language=python]
RF_model = joblib.load("RF_model.joblib")
XGB_model = joblib.load("XGB_model.joblib")
```

A.5 Artificial Neural Network Function

```
def neural_network(x_train, y_train, x_test, y_test, threshold):
    NN_model = Sequential()
    NN_model.add(Dense(64, activation='relu', input_dim=k))
    NN_model.add(Dense(32, activation='relu'))
    NN_model.add(Dense(1, activation='sigmoid'))
    NN_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = NN_model.fit(x_train, y_train, batch_size=8, epochs=60, validation_split
        ↪ =0.2, shuffle=True)
```

```
    NN_model.save('NN_model.h5')
    global NN_y_pred_bool
    NN_y_pred = NN_model.predict(x_test)
    NN_y_pred_bool = (NN_y_pred > threshold)
    cm = confusion_matrix(y_test, NN_y_pred_bool)
    print(cm)
    print("Classification report\n%s\n" % (classification_report(y_test, NN_y_pred_bool)
        ↪ ))
    print("total frauds in test set ",
          "\n\nStats for Neural Network",
          ":\ncounter of alert ", cm[0][0],
          "\ncatched frauds ", cm[1][1],
          "\nfalse positive ", cm[1][0],
          "\nmissed frauds ", cm[0][1],
          "\nF1 - Score ", f1_score(y_test, NN_y_pred_bool, average='macro'),
          "\nMCC ", matthews_corrcoef(y_test, NN_y_pred_bool))
    plot_accuracy_loss(history)
```

A.6 Plot Function

```
def plot_accuracy_loss (history):
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
```

```
    plt.show()
```

A.7 Statistics function

```
def stats_and_featureplot(y_pred, y_test, model, model_name):
    counter = 0
    catched_fraud = 0
    false_positive = 0
    global total_fraud
    total_fraud = 0

    accuracy = accuracy_score(y_test, y_pred)
    print("\nAccuracy ", model_name, " : %.2f%%" % (accuracy * 100.0))
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    for i in range(len(y_test)):
        if y_test[i] == 1:
            total_fraud = total_fraud + 1

    for j in range(len(y_pred)):
        if y_pred[j] > 0:
            counter = counter + 1
            if y_test[j] == 1:
                catched_fraud = catched_fraud + 1
            else:
                false_positive = false_positive + 1
    missed_fraud = total_fraud - catched_fraud

    print("total frauds in test set ", total_fraud,
          "\n\nStats for ", model_name,
          ":\ncounter of alert ", counter,
          "\ncatched frauds ", catched_fraud,
          "\nfalse positive ", false_positive,
          "\nmissed frauds ", missed_fraud,
          "\nF1 - Score ", f1_score(y_test, y_pred, average='macro'),
          "\nMCC ", matthews_corrcoef(y_test, y_pred))
```

```
    title = model_name + '_feature_importance'
    feature_names = dataset.columns[1:k]
    plot_feature_importance(model.feature_importances_, feature_names, title)
```

A.8 Feature importance Function

```
def plot_feature_importance(importance, names, title):
    #Create arrays from feature importance and feature names
    feature_importance = np.array(importance)
    feature_names = np.array(names)

    #Create a DataFrame using a Dictionary
    data={'feature_names':feature_names,'feature_importance':feature_importance}
    fi_df = pd.DataFrame(data)

    #Sort the DataFrame in order decreasing feature importance
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)

    #Define size of bar plot
    plt.figure(figsize=(10,8))
    #Plot Searborn bar chart
    sns.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])
    #Add chart labels
    plt.title(title)
    plt.xlabel('FEATURE_IMPORTANCE')
    plt.ylabel('FEATURE_NAMES')
    plt.show()
```

A.9 Score function

```
def score(XGB_y_pred, RF_y_pred, NN_y_pred_bool, y_test):
    # need to create another function for that
    score_file = pd.DataFrame(columns=['RF', 'XGB', 'NN', 'Score', 'Fraud'])
    highCounter = 0
    mediumCounter = 0
    lowCounter = 0
    highFraud = 0
    mediumFraud = 0
    lowFraud = 0
    highFP = 0
    mediumFP = 0
    lowFP = 0
    missedGeneral = 0
    for i in range(len(y_test)):
        if XGB_y_pred[i] == 1 and RF_y_pred[i] == 1 and NN_y_pred_bool[i] == 1:
            score = 3
            highCounter = highCounter + 1
            if y_test[i] == 1:
                highFraud = highFraud + 1
            else:
                highFP = highFP + 1
        elif ((XGB_y_pred[i] == 1 and RF_y_pred[i] == 1) or (RF_y_pred[i] == 1 and
            ↪ NN_y_pred_bool[i] == 1)
              or (XGB_y_pred[i] == 1 and NN_y_pred_bool[i] == 1)):
            score = 2
            mediumCounter = mediumCounter + 1
            if y_test[i] == 1:
                mediumFraud = mediumFraud + 1
            else:
                mediumFP = mediumFP + 1
        elif XGB_y_pred[i] == 1 or RF_y_pred[i] == 1 or NN_y_pred_bool[i] == 1:
            score = 1
            lowCounter = lowCounter + 1
            if y_test[i] == 1:
                lowFraud = lowFraud + 1
```

```
            else:
                lowFP = lowFP + 1
        else:
            score = 0

        missedGeneral = total_fraud - highFraud - mediumFraud - lowFraud
        score_file.loc[i] = [RF_y_pred[i], XGB_y_pred[i], NN_y_pred_bool[i], score,
            ↪ y_test[i]]

    print("\n\ntotal_frauds_in_test_set_", total_fraud,
          "\n\nStats_for_High_Score:",
          "\ncounter_of_High_Score_", highCounter,
          "\ncatched_frauds_", highFraud,
          "\nfalse_positive_", highFP,
          "\n\nStats_for_Medium_Score:",
          "\ncounter_of_alert_", mediumCounter,
          "\ncatched_frauds_", mediumFraud,
          "\nfalse_positive_", mediumFP,
          "\n\nStats_for_Low_Score:",
          "\ncounter_of_alert_", lowCounter,
          "\ncatched_frauds_", lowFraud,
          "\nfalse_positive_", lowFP,
          "\n\nmissed_frauds_in_general", missedGeneral)
    score_file.to_csv('ScoreFile.csv')
```

Ringraziamenti

A conclusione di questo percorso magistrale desidero ringraziare prima di tutti coloro che mi hanno guidato nel lavoro di tesi:

- Il professore Michele Nappi, per avermi accolto nel suo team.
  Ho iniziato questo percorso magistrale seguendo ed affrontando il suo corso come primo della mia lista, e chiudo il cerchio concludendo il percorso con Lei, il che rende tutto un po' più poetico. Grazie.

- Il professor Fabio Narducci, per avermi seguito assiduamente con la sua proverbiale serenità in grado di trasmette fiducia e sicurezza anche nei periodi tutt'altro che sereni.
  La mia stima nei suoi riguardi iniziò qualche anno prima del percorso magistrale e non può che crescere costantemente, sia per le sue enormi conoscenze, sia per il suo ancor più ricco lato umano. Grazie di tutto.

Un ringraziamento particolare va alla mia Famiglia:

- Ai miei Genitori: Con la vostra costante stima mi avete donato un'identità sociale che mi ha reso in grado di affrontare grandi sfide, puntando in alto senza remore.
  Non mi avete mai fatto sentire la minima mancanza, sia in termini morali che fisici, e questo fa di voi il modello da seguire quando giungerà il giorno di diventare genitore a mia volta.

- A Federica: Grazie per aver passato metà della tua vita al mio fianco. Il tuo supporto è stato fondamentale per la riuscita di questo percorso. Crescere insieme e gioire l'uno dei traguardi dell'altro è un'esperienza fantastica. Spero di passare il resto della mia vita con te. Ti amo.

- Ai miei Fratelli: la vostra presenza nella mia vita è fonte inesorabile di gioia per me. Vedervi crescere e maturare mi rende molto orgoglioso di essere il vostro fratello maggiore. Spero che le vicissitudini della vita non ci separino mai.

- A Stitch: Grazie per la felicità che mi regali ogni volta che torno a casa e mi fai le feste.

Ringrazio Luca per essere stato un ottimo alleato nelle sessioni d'esame, sono onorato di aver condiviso la "trincea" con te! Sono sicuro che avrai una carriera brillante.

And last but not least, I would like to thank Heidi immensely. Her contribution in terms of moral and technical support was of vital importance for the success of this project. I really appreciate your great help.
I feel honored to have a role model like you in my team and in my life.

Bibliography

[1] Henk Esselink and Lola Hernández. The use of cash by households in the euro area. *ECB Occasional Paper*, 1(201), 2017. 5, 11, 12

[2] Sixth report on card fraud. `https://www.ecb.europa.eu/pub/cardfraud/html/ecb.cardfraudreport202008~521edb602b.en.html`, 2019. Accessed: 23/11/2020. 5, 13, 14

[3] Support vector machine — introduction to machine learning algorithms. `https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47`, 2018. Accessed: 28/11/2020. 5, 21, 22

[4] Contactless continent. `https://newsroom.mastercard.com/eu/press-releases/contactless-continent/`, 2020. Accessed: 23/11/2020. 12

[5] Dejan Varmedja, Mirjana Karanovic, Srdjan Sladojevic, Marko Arsenovic, and Andras Anderla. Credit card fraud detection-machine learning methods. In *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–5. IEEE, 2019. 13, 23

[6] Eric WT Ngai, Yong Hu, Yiu Hing Wong, Yijun Chen, and Xin Sun. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision support systems*, 50(3):559–569, 2011. 15, 17, 19, 22

[7] Richard J Bolton, David J Hand, et al. Unsupervised profiling methods for fraud detection. *Credit scoring and credit control VII*, pages 235–255, 2001. 15

[8] Logistic regression for machine learning. https://machinelearningmastery.com/logistic-regression-for-machine-learning/, 2016. Accessed: 27/11/2020. 16

[9] A beginner's guide to decision tree classification. https://towardsdatascience.com/a-beginners-guide-to-decision-tree-classification-6d3209353ea, 2018. Accessed: 27/11/2020. 17

[10] Machine learning basics with the k-nearest neighbors algorithm. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761, 2018. Accessed: 27/11/2020. 18

[11] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. 20

[12] A gentle introduction to long short-term memory networks by the experts. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761, 2018. Accessed: 27/11/2020. 20

[13] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2008. 20

[14] How does back-propagation in artificial neural networks work? https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7, 2019. Accessed: 28/11/2020. 20

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009. 20

[16] Naive bayes classifier. https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c, 2018. Accessed: 28/11/2020. 20

[17] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. 21

[18] A gentle introduction to bayesian belief networks. https://machinelearningmastery.com/introduction-to-bayesian-belief-networks/, 2019. Accessed: 28/11/2020. 21

[19] Siddhartha Bhattacharyya, Sanjeev Jha, Kurian Tharakunnel, and J Christopher Westland. Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3):602–613, 2011. 22

[20] Christopher Whitrow, David J Hand, Piotr Juszczak, David Weston, and Niall M Adams. Transaction aggregation as a strategy for credit card fraud detection. *Data mining and knowledge discovery*, 18(1):30–55, 2009. 23

[21] Johannes Jurgovsky, Michael Granitzer, Konstantin Ziegler, Sylvie Calabretto, Pierre-Edouard Portier, Liyun He-Guelton, and Olivier Caelen. Sequence classification for credit-card fraud detection. *Expert Systems with Applications*, 100:234–245, 2018. 23, 36

[22] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. 23

[23] Wikipedia - neural network. https://it.wikipedia.org/wiki/Rete_neurale_artificiale. 24

[24] Walter Pitts Warren McCulloch. *A Logical Calculus of the Ideas Immanent in Nervous Activity*. springer, 1986. 25

[25] Richard Brown. Donald o. hebb and the organization of behavior: 17 years in the writing. *Molecular Brain*, 13, 12 2020. 27

[26] Anaconda - website. https://www.anaconda.com. 29

[27] Numpy - website. https://numpy.org. 29

[28] scikit-learn - website. https://scikit-learn.org. 30

[29] Pandas - website. https://pandas.pydata.org. 30

[30] Keras - website. https://keras.io/. 30

[31] Matplotlib - website. https://matplotlib.org. 30

[32] Principal component analysis. https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60. 31

[33] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002. 32

[34] Ugo Fiore, Alfredo De Santis, Francesca Perla, Paolo Zanetti, and Francesco Palmieri. Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. *Information Sciences*, 479:448–455, 2019. 36

[35] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection: a realistic modeling and a novel learning strategy. *IEEE transactions on neural networks and learning systems*, 29(8):3784–3797, 2017. 36

[36] 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. *Information Sciences*, 2019. 36

[37] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):6, 2020. 37

[38] Matthews correlation coefficient is the best classification metric you've never heard of. https://towardsdatascience.com/the-best-classification-metric-youve-never-heard-of-the-matthews-correlation-coefficient-3bf50a2f3e9a, 2019. Accessed: 01/12/2020. 37

[39] Kuldeep Randhawa, Chu Kiong Loo, Manjeevan Seera, Chee Peng Lim, and Asoke K Nandi. Credit card fraud detection using adaboost and majority voting. *IEEE access*, 6:14277–14284, 2018.

[40] Sanghamitra Bandyopadhyay, Sankar K Pal, and CA Murthy. Simulated annealing based pattern classification. *Information Sciences*, 109(1-4):165–184, 1998.

[41] Gianni D'Angelo, Salvatore Rampone, and Francesco Palmieri. Developing a trust model for pervasive computing based on apriori association rules learning and bayesian classification. *Soft Computing*, 21(21):6297–6315, 2017.

[42] Gianni D'Angelo, Francesco Palmieri, and Salvatore Rampone. Detecting unfair recommendations in trust-based pervasive environments. *Information Sciences*, 486:31–51, 2019.

[43] C Victoria Priscilla and D Padma Prabha. Influence of optimizing xgboost to handle class imbalance in credit card fraud detection. In *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 1309–1315. IEEE, 2020.

[44] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[45] Nestor Hernandez, Mizanur Rahman, Ruben Recabarren, and Bogdan Carbunar. Fraud de-anonymization for fun and profit. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 115–130, 2018.

[46] Cuizhu Meng, Li Zhou, and Bisong Liu. A case study in credit fraud detection with SMOTE and XGBoost. *Journal of Physics: Conference Series*, 1601:052016, aug 2020.

[47] Fraud the facts 2020 the definitive overview of payment industry fraud. `https://www.ukfinance.org.uk/system/files/Fraud-The-Facts-2020-FINAL-ONLINE-11-June.pdf`. Accessed: 23/11/2020.

[48] Feature selector: Simple feature selection in python. `https://github.com/WillKoehrsen/feature-selector`, 2018. Accessed: 28/11/2020.

[49] Feature importance and feature selection with xgboost in python. `https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/`, 2016. Accessed: 03/12/2020.

[50] Smarter parameter sweeps (or why grid search is plain stupid). `https://medium.com/rants-on-machine-learning/smarter-parameter-sweeps-or-why-grid-search-is-plain-stupid-c17d97a0e881`, 2015. Accessed: 27/11/2020.

[51] Hyperparameter tuning using simulated annealing. `https://santhoshhari.github.io/simulated_annealing/`. Accessed: 27/11/2020.

[52] Simulated annealing. `https://en.wikipedia.org/wiki/Simulated_annealing`. Accessed: 01/12/2020.

[53] Shi-hua Zhan, Sannuo Lin, Ze-jun Zhang, and Yiwen Zhong. List-based simulated annealing algorithm for traveling salesman problem. *Computational Intelligence and Neuroscience*, 2016:1–12, 03 2016.

[54] Oversampling and undersampling. `https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf`, 2020. Accessed: 28/11/2020.

[55] A gentle introduction to xgboost for applied machine learning. `https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/`, 2016. Accessed: 01/12/2020.

[56] A step by step explaination of principal component analysis. https://builtin.com/data-science/step-step-explanation-principal-component-analysis, 2019. Accessed: 30/11/2020.

[57] Multi-layer perceptron (mlp). https://xzz201920.medium.com/multi-layer-perceptron-mlp-4e5c020fd28a, 2020. Accessed: 28/11/2020.

[58] Jupiter - website. https://jupiter.org.

[59] Kaggle - credit card transactions dataset. https://www.kaggle.com/mlg-ulb/creditcardfraud.

[60] Imbalanced learn - page on pypi website. https://pypi.org/project/imbalanced-learn/.

[61] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2):58, 2004.

[62] Nathalie Japkowicz. The class imbalance problem: Significance and strategies. In *Proc. of the Int'l Conf. on Artificial Intelligence*, volume 56. Citeseer, 2000.