ABSTRACT

Financial data is some of the most sensitive information stored on the Internet. With the advent of new attack methods in mobile technology, cardholder property is being invaded, making traditional fraud detection systems unable to provide the necessary security. For this reason, we have developed an advanced system that relies on new technologies to immediately detect unusual behavior and prevent inauthentic transactions.

Our system is based on machine learning classification algorithms, which are considered the best solution to the aforementioned problem, as they can find sophisticated fraud features that a human simply cannot detect.

There are many approaches to detect fraudulent transactions, but not many of them result in high accuracy due to high transaction class imbalance. Tree Boosting has proven to be a highly effective and widely used machine learning method for solving various regression and classification problems. In this work, a tree boosting method called Extreme Gradient Boosting Algorithm (XGBoost) is used to address credit card fraud detection and deal with data imbalance.

Two XGBoost models are built and their performances are evaluated. Very satisfactory results, related mainly to the accuracy of transaction classification, are obtained for both datasets (balanced/unbalanced).

DEDICATION

I dedicate this humble work to

My dear parents for their endless love, support and encouragement. Words would never suffice to express my gratitude for all they have provided for me.

My sister's encouraging words and push to persevere ring in my ears.

And my special friends who have supported me throughout the process; I will always appreciate all they have done.

ACKNOWLEDGMENTS

Praise to Allah, the Most Beneficent and the Most Merciful, for all his blessings and givings, for leading me during all these years of my study path and for giving me the strength and knowledge to realize this modest work.

My deepest gratitude goes to

My project Supervisor Dr.R. NAMANE for taking me under his supervision, for believing in my ability to make an idea come true, and for his understating and guidance throughout the duration of this project.

My parents who have always been there for me and for their devoted and unconditional support.

Not least of all, I owe so much to my whole family for their undying support, their unwavering belief that I can achieve so much.

Finally I would like to thank my friends and the teachers in the institute that gave me guidance through the last five years, May God bless them.

Table of Contents

List of Figures

List of Tables

GENERAL INTRODUCTION

Problem Statement

With the growth of internet and e-commerce, online payment becomes inevitable. Credit card or debit card is increasingly becoming the popular payment method for both online and normal purchases as it is a more convenient way of payment. With the explosive development of modern technology, data breaches have also increased significantly, leading to a rise in credit card fraud cases. Therefore, an efficient fraud detection method is essential to maintain the reliability of the payment system. In this project we have used Extreme Gradient Boosting (XGBoost) for fraud detection and took the fraud detection problem as a binary classification to predict the class of transactions (Class 0: No fraud, Class 1: Fraud).

Motivation

Generally, credit card fraud is a state crime, but in some cases, card fraud falls under federal jurisdiction. As defined by the Federal Bureau of Investigation, credit card fraud is the unauthorized use of a credit or debit card or similar payment tool, to fraudulently obtain money or property. Credit and debit card numbers can be stolen from unsecured websites or obtained as part of an identity theft scheme [6].
With a huge amount of loss, credit card fraud has become a very major risk for financial institutions, the prevention of which with traditional prevention techniques such as PINs, passwords, and identification systems has become insufficient in modern banking systems. Fortunately, the techniques of Machine Learning can play an important role in combating these types of fraud, as they tend to provide more accurate insights and make predictions when large amounts of data are fed into the system. Recently, it has proven to be valuable in making sense of huge amounts and variety of data and solving the critical and urgent needs of businesses, especially in the financial industry as it is one of the industries that tend to generate huge amounts of data related to daily transactions. Much of this is stored online, which increases the risk of a security breach

and results in fraud being a major problem for banking institutions, causing billions of dollars in losses each year. Therefore, using a robust machine learning model to detect and prevent fraud is very important.

Aim and Objectives

The aim of this work is to provide fraud detection model that will enable bank to detect fraudulent transactions. The objective is to focus on achieving:

- High accuracy,
- High fraud detection (True Positives) rate and low false alarms (False Positives),
- Good fitting model.
- Fast training rate.

Structure of Work

This project attempted to implement a Gradient Tree Boosting Algorithm known as XGBoost to solve the problem of detecting credit card fraud.

Chapter 1 outlines some previous work that has been done to solve this problem. It also gives an introduction to the various machine learning classification models and evaluation methods. It also explains the concepts on which XGBoost is built. The third chapter goes deeper into the features and mathematics behind the Extreme Gradient Boosting algorithm. Chapter 3 shows how to get a feel for the data and how to build and train the XGBoost model. Finally, in Chapter 4, the model evaluation is performed and the results are discussed.

Chapter 1

BACKGROUND AND RELATED WORK

Introduction

Credit card fraud detection plays an important role in reducing the losses caused by fraudulent transactions for banks. Recently, it has become an active research topic, especially with the exploding growth of Big Data and Machine Learning techniques. Machine Learning has become one of the cornerstones of information technology and thus a fairly central part of our lives. As the amount of data continues to grow, there is good reason to believe that smart data analytics will become even more ubiquitous as a necessary part of technological advancement.

In this chapter, we will see some related work on fraud detection and the limitations of previous models. We will also give an introduction to machine learning, focusing on the main concepts on which XGBoost is built. Later, we will attempt to develop an understanding of the theory behind XGBoost.

1.1 Machine Learning

To understand what Machine Learning (ML) is, we must first look at the basic concepts of Artificial Inteligence (AI). AI is defined as a program that has cognitive abilities similar to those of a human. Getting computers to think like humans and solve problems the way we do is one of the main ideas of artificial intelligence. Any computer program that exhibits features such as self-improvement, learning through inference, or even basic human tasks such as image recognition and language processing is considered a form of AI.

Simply put, ML is a branch of computer science that evolved from the study of pattern recognition and computational learning theory of AI. It deals with the construction

and study of algorithms that can learn and make predictions from data. Such algorithms work by building a model from example inputs to make data-driven predictions or decisions, rather than following strictly static program instructions.
As with any method, there are different ways to train machine learning algorithms, each with their own advantages and disadvantages.

1.1.1 Machine learning and Statistics

As an interdisciplinary field, ML shares common threads with the mathematical fields of statistics [20]. It is naturally a sub-field of computer science, as our goal is to program machines so that they will learn. However, in contrast with traditional AI, ML is not trying to build automated imitation of intelligent behavior, but rather to use the strengths and special abilities of computers to complement human intelligence, often performing tasks that fall way beyond human capabilities. For example, the ability to scan and process huge databases allows ML programs to detect patterns that are outside the scope of human perception.
There are many similarities between the disciplines of ML and statistics, both in terms of the goals and the techniques used. While ML is a computer science perspective on modeling data with a focus on algorithmic methods and modeling capabilities, statistics is a mathematical perspective on modeling data with a focus on data models and goodness of fit.

1.1.2 Machine learning approaches

The area of Machine Learning is often divided in subareas according to the kinds of problems being attacked. A rough categorization can be seen in Figure 1.1.

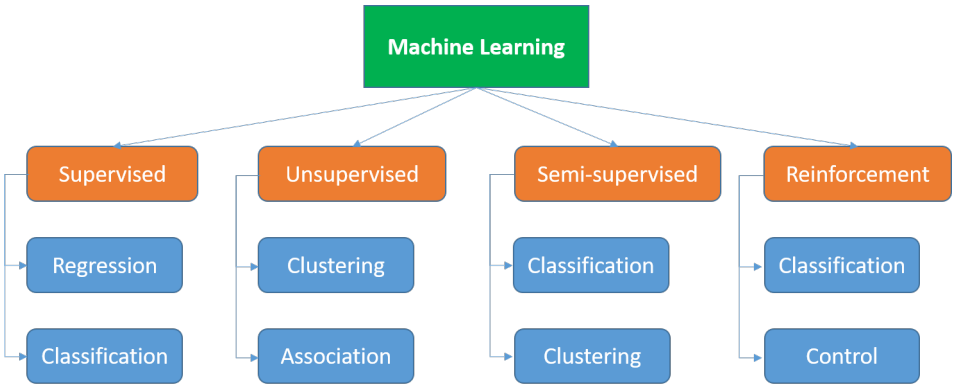

Figure 1.1: Machine Learning (ML) categories

Supervised Learning

Supervised learning algorithms are trained on labelled examples, such as an input where the desired output is known. The learning algorithm is given a set of inputs along with the corresponding correct outputs, it then learns by comparing its predicted outputs with the correct outputs to find errors and modifies the model accordingly.
Through methods such as classification, regression, prediction, and gradient boosting, supervised learning uses patterns to predict labelled values on additional unlabeled data.

Unsupervised learning

Unsupervised learning uses data that has no historical labels. The goal is to explore the data and find a structure in it. Therefore, the system is not given the correct answer, yet the algorithm is supposed to figure out what it is being shown.
Unsupervised learning works well on transactional data. For example, it can identify segments with similar attributes. Or it can find the main attributes that separate segments from each other. Popular techniques include self-organizing maps, nearest-neighbor mapping, k-means clustering, and singular value decomposition.

Semi-supervised Learning

Semi-supervised learning is used for the same applications as supervised learning. However, it uses both labeled and unlabeled data for training - typically a small amount of labeled data with a large amount of unlabeled data. This type of learning can be used with methods such as classification, regression, and prediction. Semi-supervised

learning is useful when the cost associated with labeling is too high to enable fully labeled training. An early example of this is identifying a person's face on a webcam.

Reinforcement Learning

In reinforcement learning, the algorithm uses trial and error to figure out which actions yield the greatest rewards. This type of learning has three main components: the agent (the learner or decision-maker), the environment (everything the agent interacts with), and the actions (what the agent can do). The goal is for the agent to select actions that maximize the expected reward in a given amount of time. The agent will achieve the goal much faster if it follows a good strategy. So the goal in reinforcement learning is to learn the best strategy. Reinforcement learning is commonly used for robotics, gaming, and navigation.

1.2 Related Work

In the past, fraud detection systems were developed based on a set of rules that can be easily circumvented by modern fraudsters. As a result, most companies now rely on machine learning to detect and combat fraudulent financial transactions. This involves comparing a transaction to other data points to determine if the flagged transaction matches the account holder's behavior. Depending on the type of transaction, the system can automatically reject a withdrawal or purchase until a human makes a decision, or flag the transaction for further investigation by security teams.
To develop these new modern systems, a lot of research has been done in the field of fraud detection through machine learning.

A. C. Bahnsen proposed a new set of features that analyzed consumer spending behavior. The features are useful in detecting credit card fraud. However, this model may take too long to classify a new transaction by calculating the features [2].

G. Rushin compared the three supervised classification models: logistic regression, Gradient Boosting Machine (GBM), and Deep Learning (DL) in fraud detection. And two methods were used in feature processing. However, this process takes a lot of time

because feature selection is not applied [1].

Zhang Yongbin proposed a behavior-based model for credit card fraud detection. Here, the historical behavioral pattern of the customer is used to detect fraud. The transaction record of a single credit card is used to build the model. In this model, the unsupervised self-organizing map method is used to detect the outliers from the normal ones [3].

All these proposed machine learning methods addressing the problem of fraud detection have shown promising results, but it is still very difficult to detect fraudulent transactions accurately and promptly due to the imbalance of data and the large variation of fraudulent transactions.
XGBoost has shown great performance in many projects since its development. In this project, we will build an XGBoost classifier model for detecting fraudulent transactions to see how well it performs on a very unbalanced dataset.

1.3 Classification Algorithms

A machine learning algorithm is a program with a specific way of adjusting its own parameters based on feedback about its previous performance in making predictions about a dataset. Similar to how humans change the way they process data through learning, machine learning algorithms adapt to perform better when exposed to more data.
A classification algorithm, in general, is a function that weighs the input features so that the output separates one class into positive values and the other into negative values. Classifier training is performed to identify the weights (and functions) that provide the most accurate and best separation of the two classes of data.
Most common Classification algorithms are K nearest Neighbour, Support Vector Machines (SVM), Naive Bayes, and Decision Trees.
XGBoost can be used for both classification and regression problems. In our case, since the goal is to predict a class of a transaction, we will use XGBoost for classification. Therefore, we will only consider this case in this project.

Before we try to explain how XGBoost works, let's first go over the basics that lead up to it. In Figure 1.2, we see the evolution of models over time from decision trees to XGBoost.

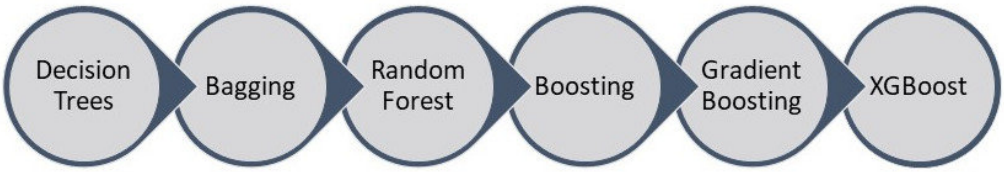

Figure 1.2: Evolution of XGBoost Algorithm from Decision Trees [22]

1.3.1 Decision Trees

A decision tree is a series of nodes, a directed graph that starts at the base with a single node and extends to the many leaf nodes that represent the categories that the tree can classify. Another way to think of a decision tree is as a flowchart, where the flow starts at the root node and ends with a decision at the leaves.

It is a decision support tool that uses a tree-like graph to represent the predictions that result from a series of feature-based splits [23].

The figure below shows a the schematic of a decision tree.

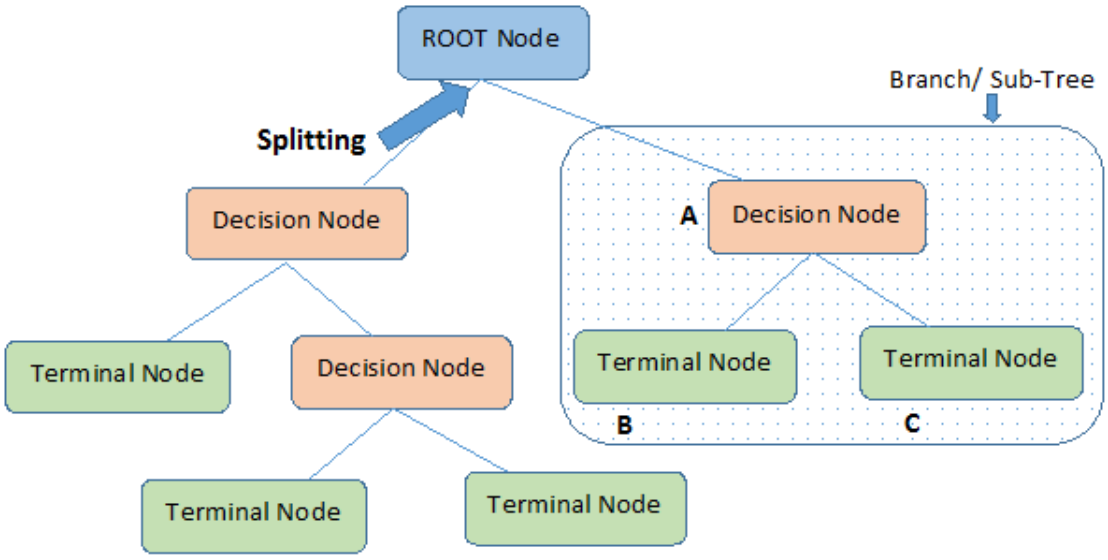

Figure 1.3: Schematic illustration of a Decision Tree [23]

Tree-based algorithms are considered one of the best and most widely used methods

of supervised learning. They enable predictive models with high accuracy, stability and ease of interpretation [24]. Unlike linear models, they represent non-linear relationships very well and are adaptive in solving any type of problem (classification or regression). While decision trees are one of the most easily interpretable models, but they exhibit highly variable behavior (they are known to be associated with high variance).

Bias-Variance Trade-Off

1- Bias Error

Bias is the simplifying assumptions a model makes to make the objective function easier to learn. In general, linear algorithms have a high bias, making them quick to learn and easier to understand, but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithm's bias.

2- Variance Error

Variance is the amount by which the estimate of the objective function would change if different training data were used.

There is a tradeoff between these two concerns at play, and the algorithm chosen and the way it is configured lead to different equilibria in this tradeoff. The objective function is estimated by a machine learning algorithm from the training data, so we should expect the algorithm to have some variance. Ideally, it should not change too much from one training dataset to the next, which means that the algorithm is good at finding out the hidden underlying association between the inputs and the output variables.

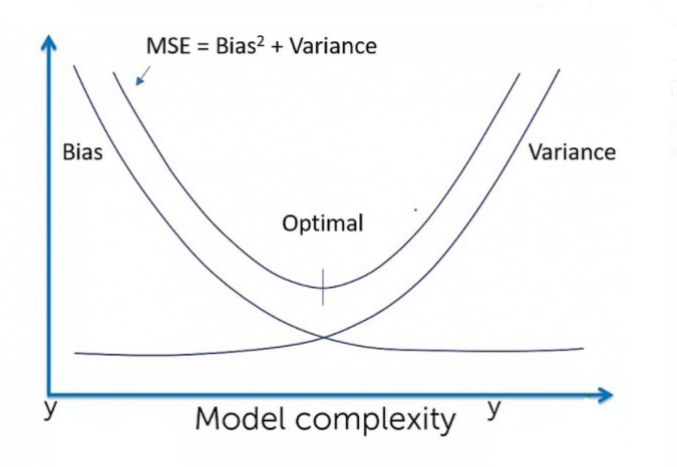

Figure 1.4: Bias variance tradeoff based on model complexity [33]

1.3.2 Bagging

Bagging or Bootstrap Aggregation was formally introduced by Leo Breiman in 1996 [27]. Bagging is an Ensemble Learning technique that aims to reduce learning error by implementing a set of homogeneous machine learning algorithms. The key idea of bagging is to use multiple base learners trained separately with a random sample from the training set, which produce a more stable and accurate model through a tuning or averaging approach.

The main two components of bagging technique are: the random sampling with replacement (bootstraping) and the set of homogeneous machine learning algorithms (ensemble learning).

Bagging process

The bagging process is quite easy to understand, first it is extracted "n" subsets from the training set, then these subsets are used to train "n" base learners of the same type. For making a prediction, each one of the "n" learners are feed with the test sample, the output of each learner is averaged (in case of regression) or voted (in case of classification).

Figure below shows an overview of the bagging architecture.

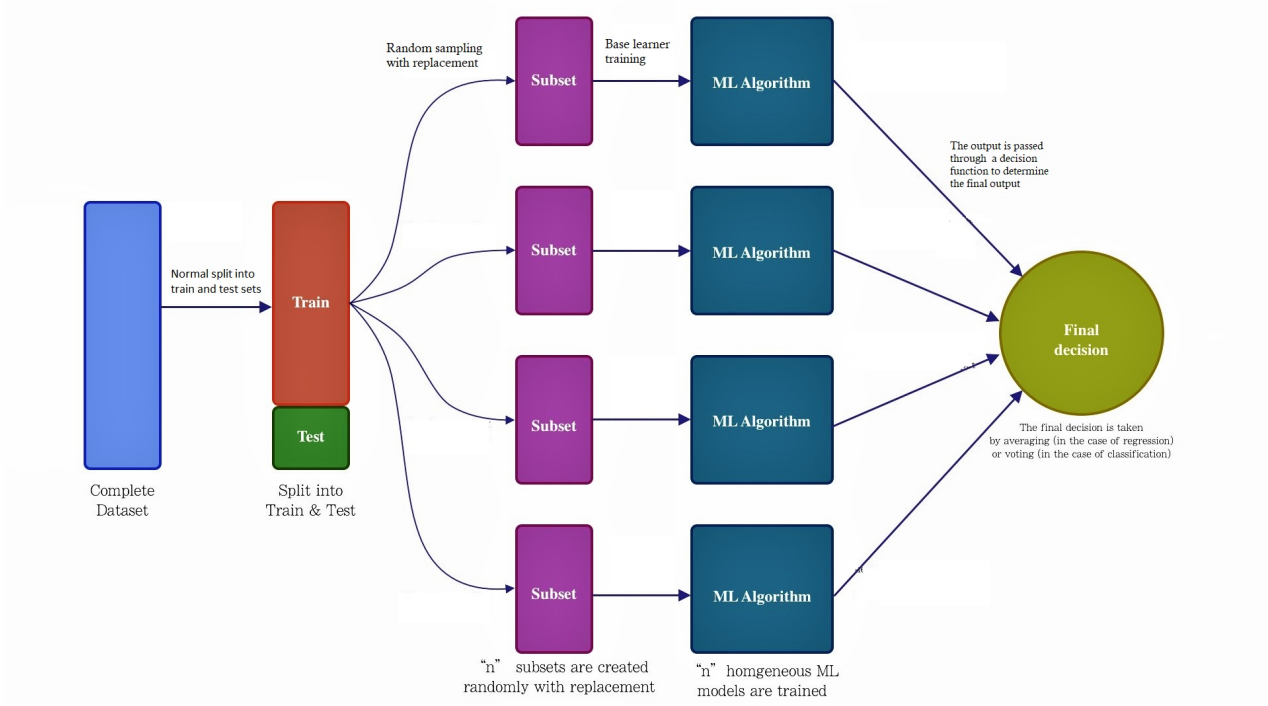

Figure 1.5: Bagging: a descriptive architecture

1.3.3 Random Forest

Random forests are made of many decision trees. They are ensembles of decision trees, each is created by using a subset of the attributes used to classify a given population. Those decision trees vote on how to classify a given instance of input data, and the random forest bootstraps those votes to choose the best prediction. This is done to prevent overfitting, a common flaw of decision trees.

A random forest is a supervised classification algorithm. It creates a forest (many decision trees) and orders their nodes and splits randomly. The more trees in the forest, the better the results it can produce.

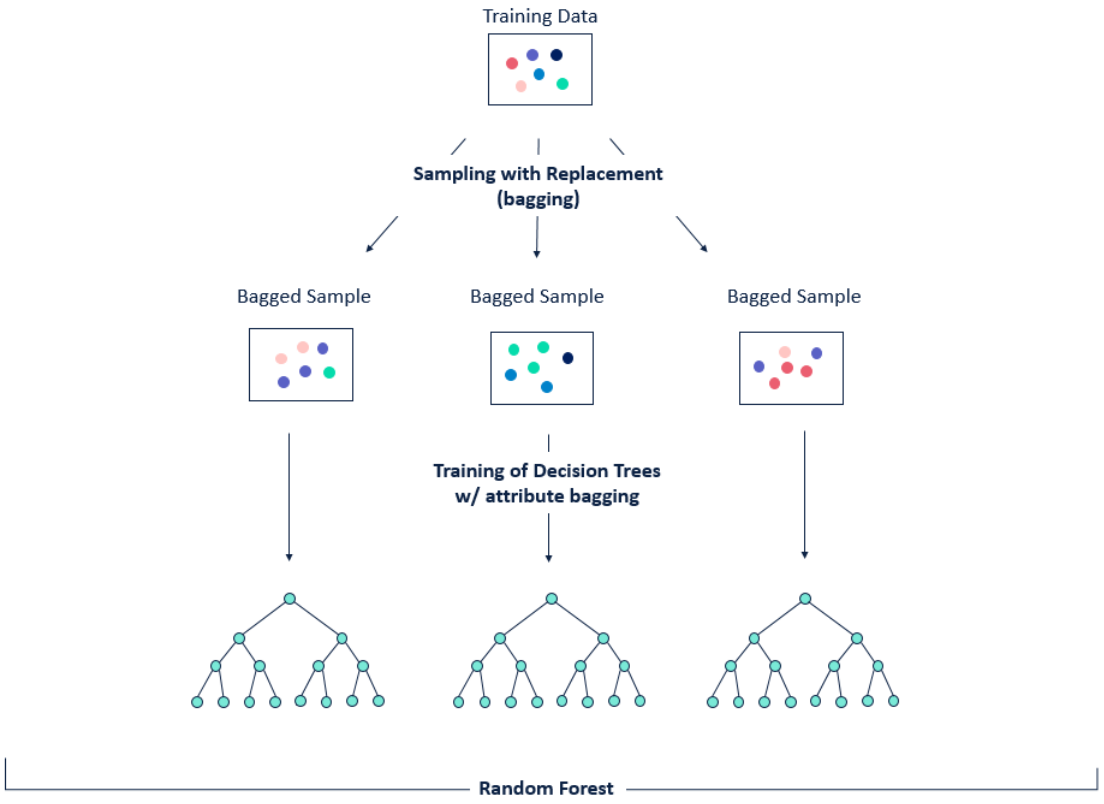

Figure 1.6: Random Forest representation [31]

1.3.4 Boosting

The initial work on boosting algorithms was done by Freund and Schapire [7]. Boosting can be understood by contrasting it to bagging. In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals.

Figure 1.7: Boosting approach

The base learners in boosting are weak learners in which the bias is high, and the predictive power is just a tad better than random guessing. Each of these learners contributes some vital information for prediction, enabling the boosting technique to produce a strong learner by effectively combining these learners. The final strong learner brings down both the bias and the variance.

In contrast to bagging techniques like Random Forest, in which trees are grown to their maximum extent, boosting makes use of trees with fewer splits. Such small trees, which are not very deep, are highly interpretable.

The boosting technique has been studied and improved over the years, several variations have been added to the core idea of boosting, GB and XGBoost are some of them. The main distinguishing feature between boosting-based techniques is the way in which errors are penalized (by modifying weights or minimizing a loss function) and the way in which the data is sampled.

1.3.5 Gradient Boosting

Gradient Boosting machine was introduced by Friedmane, it is a more robust version of boosting.

The core idea of the algorithms is based on minimizing the residuals of each learner base in a sequential manner, this minimization is performed by computing the gradient

applied to a specific loss function. Then, each base learner added to the sequence minimizes the residuals determined by the previous base learner. This is repeated until the loss function is as close to zero as possible or until a certain number of base learners are completed. Finally, to make a prediction, each of the base learners is fed the test data whose outputs are parameterized and then summed to produce the final prediction.

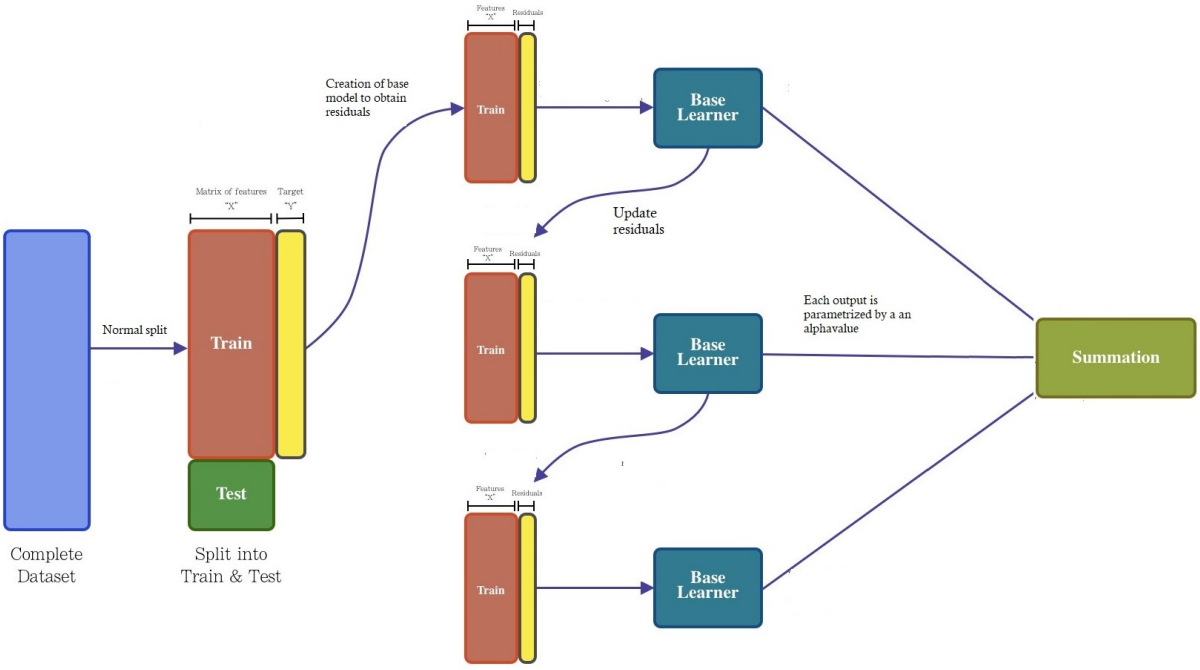

Figure 1.8: Gradient Boosting: a descriptive architecture

1.4 Performance Evaluation of Machine Learning Model

1.4.1 Confusion Matrix

Confusion matrix is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values where predicted values are described as Positive and Negative and actual values as True and False.

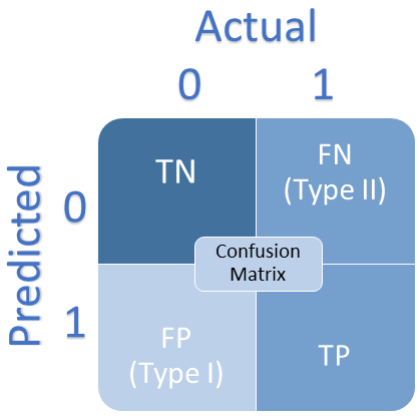

Figure 1.9: Confusion matrix [32]

True Positive: Whe the actual class is 1 and model predicted 1. (Model correctly predicted a fraudulent transaction)

False Negative: when the actual class is 1 and model predicted 0. (Model predicted non fraudulent transaction when it actually is one)

False Positive: When the actual class is 0 and model predicted 1. (Model predicted fraudulent transaction when it is actually not)

True Negative: When the actual class is 0 and model predicted 0. (Model correctly predicted a non fraudulent transaction)

1.4.2 Accuracy

Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

$$accuracy = \frac{correct\ \ predictions}{all\ \ predictions} \tag{1.1}$$

1.4.3 Precision

Precision is defined as the fraction of relevant examples (true positives) among all of the examples which were predicted to belong in a certain class or also it can be said that it is the percentage of the results which are relevant.

$$Precision = \frac{true\ \ positives}{true\ \ positives + false\ \ positives} \tag{1.2}$$

1.4.4 Recall

Recall is defined as the fraction of examples which were predicted to belong to a class with respect to all of the examples that truly belong in the class. In other words True Positive Rate, it refers to the percentage of total relevant results correctly classified.

$$Recall = \frac{true\ \ positives}{true\ \ positives + false\ \ negatives} \tag{1.3}$$

1.4.5 F1 score

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution.

F1 was introduced due to the trade-off between precision and recall, as aiming for high precision and high recall value both at the same time is not possible, aiming for a good F1 score is more convenient.

$$F1\ \ score = 2 * \frac{(Recall * Precision)}{(Recall + Precision)} \tag{1.4}$$

1.4.6 ROC Curve and AUC

ROC curve

An ROC curve (Receiver operating characteristic) is a graph showing the performance of a classification model at all classification thresholds, it summarizes every confusion matrix made by certain threshold into a graph with two parameters: True Positive Rate, False Positive Rate

In a ROC curve, a higher X-axis value indicates a higher number of False positives than True negatives. While a higher Y-axis value indicates a higher number of True positives than False negatives. So, the choice of the threshold depends on the ability to balance between False positives and False negatives.

The following figure shows a typical ROC curve.

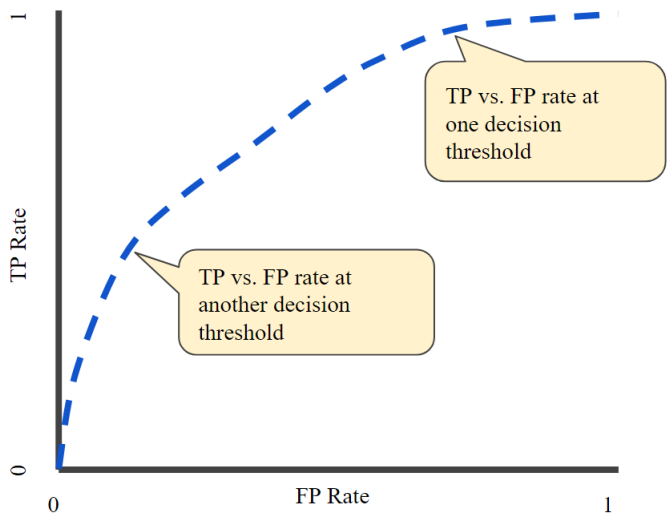

Figure 1.10: TP vs. FP rates at different classification thresholds [29]

AUC: Area Under the ROC Curve

AUC stands for Area under the ROC Curve. That is, AUC measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). It provides an aggregate measure of performance across all possible classification thresholds.

AUC is desirable because is it scale-invariant(measures how well predictions are ranked, rather than their absolute values), and it is classification-threshold-invariant (measures the quality of the model's predictions irrespective of what classification threshold is chosen).

1.4.7 Learning Curve

A learning curve (or training curve) plots the optimal value of a model's accuracy/logloss for a training set against that evaluated on a validation data set with same parameters. It is a tool to find out how much the machine model benefits from adding more training data and whether the estimator suffers more from a variance error or a bias error. If both the validation score and the training score does not improve or gets worse at a certain amount of data supplied, it means it will not benefit much from more training data.

The shape and dynamics of a learning curve can be used to diagnose the behavior of a machine learning model and in turn perhaps suggest at the type of configuration changes that may be made to improve learning and/or performance.

There are three common dynamics that are likely to observe in learning curves; they are:

1- Underfit: Underfitting refers to a model that can neither model the training data nor generalize to new data.

2- Overfit: Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. In other words, it occurs if the model or algorithm shows low bias but high variance.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function.

3- Good Fit: It is at the point just before the error on the test data set begins to increase that the model has good skill on both the training data set and the unseen test data set.

Conclusion

In this chapter, an overview of Machine Learning and its approaches has been presented. Various classification algorithms were explained, focusing on decision trees and boosting algorithms. A description of the metrics used for the evaluation of ML models was also provided.

Chapter 2

XGBOOST ALGORITHM

Introduction

Using trees as base models for boosting is a very popular choice. Seeing how trees have many benefits that boosted trees inherit while the predictive ability is greatly increased through boosting, this is perhaps not very surprising. The main drawback of boosted tree models compared to single tree models is that most of the interpretability is lost.

2.1 Boosting Algorithm

Boosting fits ensemble models of the kind

$$f(x) = \sum_{m=0}^{M} f_m(x) \tag{2.1}$$

These can be rewritten as adaptive basis function models

$$f(x) = \theta_0 + \sum_{m=1}^{M} \theta_m \phi_m(x) \tag{2.2}$$

where $f_0(x) = \theta_0$ and $f_m(x) = \theta_m \theta_m(x)$ for $m = 1, ...M$.

Boosting works by using a base learner $\mathcal{L}_\Phi$ to sequentially add basis functions, sometimes called base models, $\theta_1, ..., \theta_M \in \Phi$ that improves the fit of the current model.

Most boosting algorithms can be seen to solve

$$\{\hat{\theta_m}, \hat{\phi_m}\} = \arg min_{\{\theta_m, \phi_m\}} \sum_{i=1}^{n} L(y_i, \hat{f}^{(m-1)}(x_i) + \theta_m \phi_m(x_i)) \tag{2.3}$$

Gradient Boosting can be viewed as a general algorithm that solve the previous Equation approximately for any suitable loss function.

Gradient Boosting

Before the update at iteration m is made, the estimate of f is given by $f^{(m-1)}$. At this current estimate $f^{(m-1)}$, the direction of steepest descent of the risk is given by the

negative gradient.

$$-\hat{g}_m(x_i) = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}^{(m-1)}(x)} \tag{2.4}$$

Note that this empirical gradient is only defined at the data points $\{x_i\}_{i=1}^n$ to generalize to other points in X and prevent overfitting, we need to learn an approximate negative gradient using a restricted set of possible functions. We thus constrain the set of possible solutions to a set of basis functions Φ. At iteration m, the basis function $\phi_m \in \Phi$ is learnt from the data. The basis function we seek should produce output $\{\phi_m(x_i)\}_{i=1}^n$ which is most highly correlated with the negative gradient $\{-\hat{g}_m(x_i)\}_{i=1}^n$. This is obtained by

$$\hat{\phi}_m = \arg min \sum_{i=1}^{n} [(-\hat{g}_m(x_i)) - \beta\phi(x_i)]^2 \tag{2.5}$$

The basis function ϕ_m is learnt using a base learner where the squared error loss is used as a surrogate loss.

This procedure can be viewed as learning a constrained step direction for gradient descent, where the direction is constrained to be member of a set of basis functions Φ. The step length ρ_m to take in this step direction can subsequently be determined using line search

$$\hat{\rho}_m = \arg min \sum_{i=1}^{n} L(y_i, \hat{f}^{(m-1)}(x_i) + \rho\hat{\phi}_m(x_i)) \tag{2.6}$$

Shrinkage, where the step length at each iteration is multiplied by some factor $0 < \eta \leq 1$. This can be viewed as a regularization technique, as the components of the model are shrunk towards zero. The factor η is sometimes referred to as the learning rate as lowering it can slow down learning.

Combining all this, the "step" taken at each iteration m is given by

$$\hat{f}_m(x) = \eta\hat{\rho}_m\hat{\phi}_m(x) \tag{2.7}$$

where η is the learning rate. Doing this iteratively yields the gradient boosting procedure. The procedure is typically initialized using a constant, i.e. $f_0(x) = \phi_0$, where

$$\hat{\theta}_0 = \arg min \sum_{i=1}^{n} L(y_i, \theta) \tag{2.8}$$

2.2 Regularization

Regularization of additive tree models can be achieved in many ways. First of all, one can regularize the basis function expansion.
In this section, we will introduce the different regularization techniques used for additive tree models.

2.2.1 Model Complexity

The complexity of an additive tree model can be seen to depend on many parameters. First of all, the number of trees is clearly related to the complexity
For additive tree models, we also have the complexities of each of the individual trees. When defining the complexity of an additive tree model we can thus additionally take the complexities of the individual trees.

2.2.2 Complexity Constraints

Constraining the complexity of an additive tree model amounts to constraining the complexity of the basis function expansion and each of the basis functions. That is, we can for example constrain the number of trees. We can further constrain the complexity of an additive tree model by restricting the maximum number of terminal nodes of each individual tree. Another way is to limit the minimum number of observations falling in any terminal node.

2.2.3 Complexity Penalization

XGBoost offers the possibility of constraining the complexity of the individual trees. However, it additionally offer the possibility of penalizing the complexity of the trees. Before XGBoost, complexity penalization was not commonly used for additive tree models. This is one of the core improvements of XGBoost over Multivariate Adaptive Regression Trees (MART).
The penalization terms of the objective function can be written

$$\Omega(f) = \sum_{m=1}^{M} \left[\gamma T_m + \frac{1}{2}\lambda \|w_m\|_2^2 + \alpha \|w_m\|_1 \right] \tag{2.9}$$

The penalty is the sum of the complexity penalties of the individual trees in the additive tree model.

We see that the regularization term includes penalization of the number of terminal nodes of each individual tree through γ. Additionally, the objective includes $l2$ regularization of the leaf weights. These two penalization terms are described by Chen and Guestrin (2016). The last term in the equation $l1$ is regularization on the term weights.

2.3 Gradient Tree Boosting (XGBoost Algorithm)

XGBoost employ a different boosting algorithm for fitting additive tree models. We will refer to these as Gradient Tree Boosting (GTB).

In this section, we will develop this tree boosting algorithm.

At each iteration m, both these algorithms seek to minimize the FSAM criterion

$$J_m(\phi_m) = \sum_{i=1}^{n} L(y_i, \hat{f}^{(m-1)}(x_i) + \phi_m(x_i)) \tag{2.10}$$

For the tree boosting algorithms, the basis functions are trees

$$\phi_m(x) = \sum_{j=1}^{T} w_{jm} I(x \in R_{jm}) \tag{2.11}$$

GTB is a modification of regular gradient boosting to the case where the basis functions are trees.

A tree is learnt using the criterion

$$\hat{J}_m(\phi_m) = \sum_{i=1}^{n} [(-\hat{g}_m(x_i)) - \beta\phi(x_i)]^2 \tag{2.12}$$

which is an approximation to the criterion in Equation 5. Next, a line search step is performed. The GTB algorithm is thus not just a special case of gradient boosting where the base models are trees, but a slightly different algorithm which is based on the general GB algorithm.

We will now develop the Gradient Tree Boosting algorithm. That is we will show how the structure and leaf weights of the tree is learnt in three stages.

Learning the Weights for a Given Structure

We can rewrite the criterion in Equation 12 as

$$\hat{J}_m(\phi_m) = \sum_{j=1}^{T}\left[G_{jm}wjm + \frac{1}{2}n_{jm}w_{jm}^2\right] \tag{2.13}$$

where n_{jm} denote the number of points x_i falling in region R_jm. For a proposed, fixed structure, the weights are thus given by

$$\hat{w}_{jm} = -\frac{G_{jm}}{n_{jm}}, \qquad j = 1, ..., T. \tag{2.14}$$

Learning the Structure

Learning the structure of a tree amounts to searching for splits. For each split, a series of candidate splits are proposed, and the one which minimizes empirical risk is chosen. Equivalently, we seek the split which maximizes the gain, which is empirical risk reduction of a proposed split.

Plugging the weights from Equation 14 into the empirical risk in Equation 13, we find the criterion for a fixed structure to be

$$\hat{J}_m(\phi_m) = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_{jm}^2}{n_{jm}} \tag{2.15}$$

The gain used to determine the splits are thus given by

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{n_L} + \frac{G_R^2}{n_R} + \frac{G_{jm}^2}{n_{jm}}\right] \tag{2.16}$$

Learning the Final Weights

In the general Gradient Boosting, a line search step is performed to determine the "step" length to take in function space. For Gradient Tree Boosting however,*Friedman (2001)* presented a special enhancement. He noted that an additive tree model can be considered a basis function expansion where the basis functions are themselves basis function expansions. He further took the view that at each iteration, the algorithm was fitting T separate basis functions, one for each region of the tree. He therefore proposed to do T line search steps, one for each region $R1, ..., RT$, instead of one for the whole tree.

The optimization problem to be solved in the line search step is thus

$$\hat{w}_{jm} = \arg min \sum_i L(y_i, \hat{f}^{m-1}(x_i) + w_j), \qquad j = 1, ..., T. \tag{2.17}$$

The final leaf weights are thus determined using T separate line search steps. This greatly simplifies the line search step, which will often be difficult otherwise.

Summing up these steps, we get theGradient Tree Boosting Algorithm, which is outlined in the Algorithm.

Algorithm 1 Gradient tree boosting

Input:

Data set D

A loss function L.

The number of iterations M.

The learning rate η.

The number of terminal nodes T

1: Initialize $\hat{f}(x) = \hat{f}_0(x) = \hat{\theta}_0 = \arg min \sum_{i=1}^{n} L(y_i, \theta)$

2: **for** $m = 1, 2, .., M$ **do**

3:

$$\hat{g}_m(x_i) = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f(x)=\hat{f}^{(m-1)}(x)}$$

4: Determine the structure

5: $\{\hat{R}_{jm}\}_{j=1}^{T}$ by selecting splits which maximize $Gain = \frac{1}{2}\left[\frac{G_L^2}{n_L} + \frac{G_R^2}{n_R} + \frac{G_{jm}^2}{n_{jm}}\right]$

6: Determine the leaf weights $\{\hat{w}_{jm}\}_{j=1}^{T}$ for the learnt structure by

7: $\hat{w}_{jm} = \arg min \sum_i L(y_i, \hat{f}^{m-1}(x_i) + w_j), \qquad j = 1, ...$

8: $\hat{f}_m(x) = \sum_{j=1}^{T}\left[w_{jm} I(x_i \in \hat{R}_{jm})\right]$

9: $\hat{f}_m(x) = \hat{f}_{m-1}(x) + \hat{f}_m(x)$

10: **end**

Output: $\hat{f}_m(x) = \hat{f}_M(x) = \sum_{m=0}^{M} \hat{f}(x)$

2.4 Xgboost Hyperparameters

As seen from the previous sections, the two main hyperparameters of GB are the number of iterations or basis functions M and the learning rate or shrinkage parameter η.

There are many more other hyper parameters of the XGBoost algorithm. The overall parameters have been divided into 3 categories by XGBoost authors, we will outline

the most relevant ones.

2.4.1 General Parameters

These parameters define the overall functionality of XGBoost.

- **booster** [default=gbtree]

  Select the type of model to run at each iteration. It has 2 options:

  - gbtree: tree-based models

  - gblinear: linear models

2.4.2 Booster Parameters

- **eta** [default=0.3]

  Analogous to learning rate in GBM.

  Makes the model more robust by shrinking the weights on each step

  Typical final values to be used: 0.01-0.2

- **min-child-weight**[default=1]

  Defines the minimum sum of weights of all observations required in a child.

- **max-depth**[default=6]

  The maximum depth of a tree.

  Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.

  Typical values: 3-10

- **max-leaf-nodes**

  The maximum number of terminal nodes or leaves in a tree.

- **gamma**[default=0]

  A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.

- **subsample** [default=1]

  Denotes the fraction of observations to be randomly samples for each tree.

  Lower values make the algorithm more conservative and prevents overfitting but

too small values might lead to under-fitting.

Typical values: 0.5-1

- **lambda** [default=1]

  This used to handle the regularization part of XGBoost. Though not used often, it should be explored to reduce overfitting.

- **alpha** [default=0]

  Can be used in case of very high dimensionality so that the algorithm runs faster when implemented.

- **scale-pos-weight** [default=1]

  A value greater than 0 should be used in case of high class imbalance as it helps in faster convergence.

2.4.3 Learning Task Parameters

These parameters are used to define the optimization objective the metric to be calculated at each step.

- **objective** [default=reg:linear]

  This defines the loss function to be minimized.

- **eval-metric** [The default values are rmse for regression and error for classification.]

  Typical values are: rmse, Mean absolute error, logloss, Binary classification error rate, Multiclass classification error rate, mlogloss, Area under the curve.

- **Seed**[default=0]

  Can be used for generating reproducible results and also for parameter tuning.

Conclusion

Through this chapter, we focused of detailing the development of the Boosting and Gradient Tree Boosting (XGBoost algorithm) algorithm. With explaining the regularization techniques and XGBoost hyper-parameters tuning for optimized model.

Chapter 3

XGBOOST MODELS IMPLEMENTATION

Introduction

The credit card fraud detection problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be a fraud. This model is then used to identify whether a new transaction is fraudulent or not.
Our objectives from implementing the model are:
- Detect 100% of the fraudulent transactions.
- Minimize the incorrect fraud classifications .
- Reduce training time.

3.1 Environment Set-up

To facilitate machine learning research and application development, a variety of programming languages and environments have been employed in the past. Thanks to the general-purpose python language's enormous surge in popularity within the scientific computing community during the last decade, most modern ML and DL packages are now python-based. Python is a high-level interpreted programming language with a focus on readability that is well-known for being simple to learn while still being able to leverage the power of systems-level programming languages when needed.

3.1.1 Python Language and Jupyter Notebook Environment

Python is particularly appealing for workloads in DS, ML, and scientific computing due to the community surrounding the available tools and libraries, in addition to the benefits of the language itself. According to a recent KDnuggets poll of over 1800

people's preferences in analytics, data science, and ML, the following are the top three. In 2019, Python remained at the top of the most extensively used languages list [10]. In order to implement the XGBoost model to solve the problem of credit card fraud, Jupiter notebook and several python and machine learning libraries were used. In the next sections we will see each library used.

3.1.2 Libraries

In order to facilitate the data collection, preprocessing as well as the conduction of experiments using ML models, several software packages and libraries were used.

1. **Numpy**

   Numpy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more [11].

   The reason it is so important forDS with python is that almost all libraries in the pyData Ecosystem rely on Numpy as one of their main building blocks. Also Numpy is also incredibly fast, as it has binding to C libraries.

2. **Pandas**

   Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language [12].

   Panadas is build on top of Numpy, it allows for fast analysis and data cleaning and preparation. Furthermore, it has built-in visualization features and can work with data from a variety of sources.

3. **Matplotlib**

   Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib

can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits [14].

4. **Seaborn**

   Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures.

   Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them [13].

5. **Plotly**

   The plotly Python library is an interactive, open-source plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3-dimensional use-cases.

   Built on top of the Plotly JavaScript library (plotly.js), plotly enables Python users to create beautiful interactive web-based visualizations that can be displayed in Jupyter notebooks, saved to standalone HTML files, or served as part of pure Python-built web applications using Dash [15].

Machine Learning Libraries

6. **XGboost**

   XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. that implements ML algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples [16].

7. **Scikit-learn**

   scikit-learn is a library Built on NumPy, SciPy, and Matplotlib, very simple and

efficient tools for predictive data analysis. With scikit-learn various problems such as Classification, Regression, Clustering, Dimensionality reduction, Model selection, and Preprocessing can be solved [17].

3.2 Data Preprocessing

Before building the ML model it is inevitable to have a sense of data beforehand and do some pre-processing if necessary.

3.2.1 General Statistics

The dataset contains transactions made by credit cards in September 2013 by European cardholders. It has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. This dataset presents transactions that occurred in two days, where we have **492** frauds out of **284,807** transactions.
Taking a glimpse at the data features using "data.head()" function.

```
In [15]: print(data_df.head())
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0
```

Figure 3.1: Data set overview

As can be seen from Fig 3.1, the dataset contains 31 columns. Except for the transaction and amount columns the other columns labels are unknown. The original features and the background information about the data have not been provided due to confidentiality issues, and the data contains only numerical input variables which are the result of a PCA transformation.
The only thing that is known is that those columns that are unknown have been scaled

already.

- **PCA Transformation**

  Principal Component Analysis (PCA) is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

  Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process [18].

The data contains 284807 rows and 31 columns, in order to proceed with this dataset, it must be insured that there is no missing data.

	Time	V16	Amount	V28	V27	V26	V25	V24	V23	V22	...	V10	V9	V8	V7	V6	V5	V4	V3	V2	Class
Total	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Percent	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2 rows × 31 columns

Figure 3.2: Checking for missing data

As can be seen in the Fig 3.2, there is no missing data in the entire dataset and now we can proceed with the next steps.

3.2.2 Transactions Class Distribution

Let's check the balance between fraudulent and non fraudulent transactions and plot the distribution.

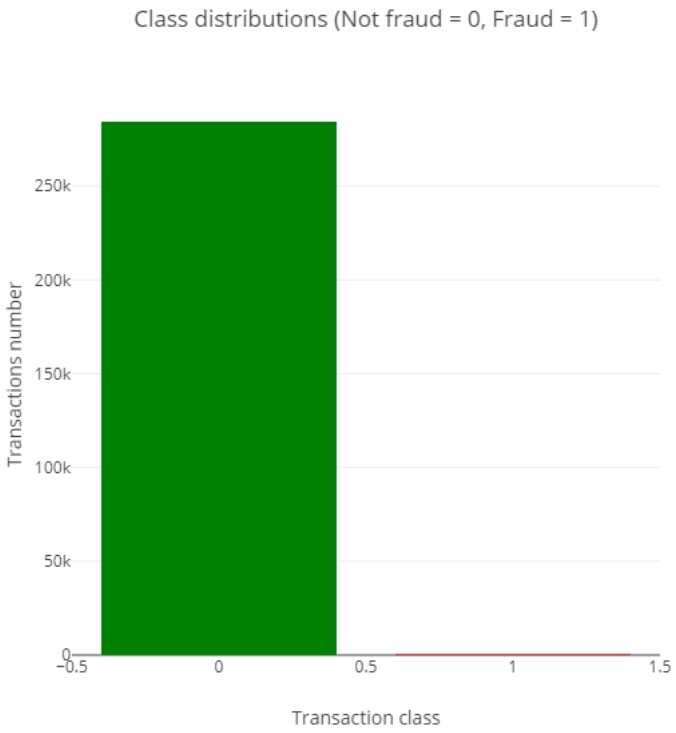

Figure 3.3: Class distribution histogram plot

It is evident from of the distributions of both classes in Fig 3.3 that data is highly imbalanced, the fraud transactions account only for **0.172%** of total transactions. XGBoost is developed to deal with high unbalanced data, we will later see the performance of the algorithm despite this unbalance.

3.2.3 Comparison of Features Distribution of the Two Classes

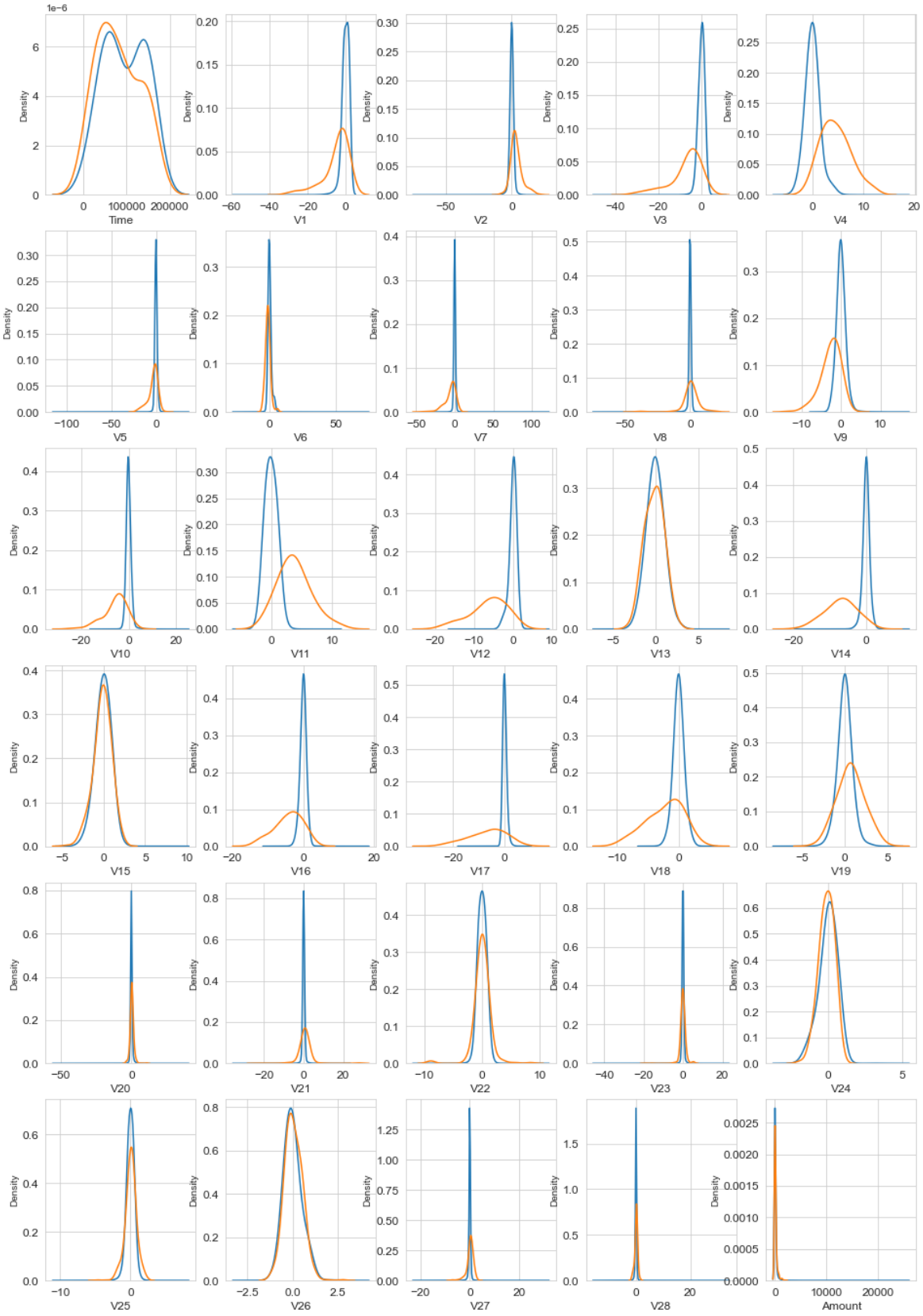

Figure 3.4: Comparison of features distribution of the two classes

Figure 3.4 shows a comparison of features distribution of the two classes. Looking at the different distributions of each feature, we can see that in general, with just few exceptions (Time and Amount), the features distribution for legitimate transactions (Class = 0) is centered around 0, sometime with a long queue at one of the extremities. In the same time, the fraudulent transactions (Class = 1) have a skewed (asymmetric) distribution.

For some of the features we can observe a good selectivity in terms of distribution for the two values of Class:

Features V4, V11 have clearly separated distributions for Class values 0 and 1. Features V12, V14, V18 are partially separated. Features V1, V2, V3, V10 have a quite distinct profile. Features V25, V26, V28 have similar profiles for the two values of Class.

We now know that V4, V11, V12, V14, V18, V1, V2, V3, V10 will be the features that will have the most impact in the training phase.

3.2.4 Transaction Time feature distribution

Let's look at the distribution of Time feature of both classes. This can be shown in Fig 3.5.

Note that the time starts at zero corresponding to midnight and ends at 48 hours.

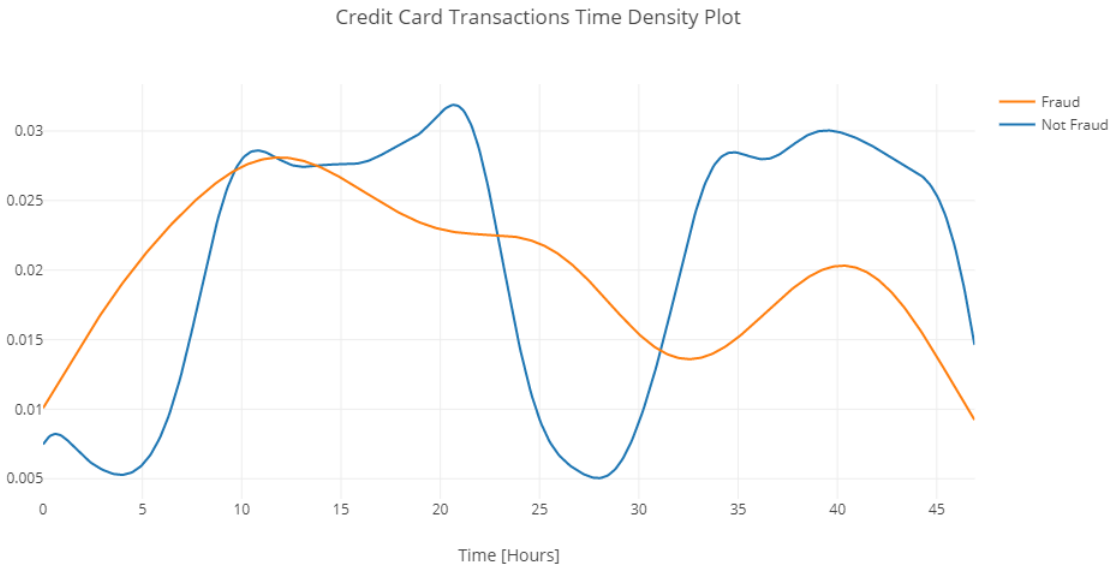

Figure 3.5: Transactions time density plot

Looking at the distribution of transaction over time it can be seen that fraudulent

transactions have an even distribution than valid transactions, including the low real transaction times, during night in Europe timezone. Which makes sense, since in real life fraud occurs on unusual times like night.

3.2.5 Transaction Amount Feature Distribution

Now let's check the Amount feature.

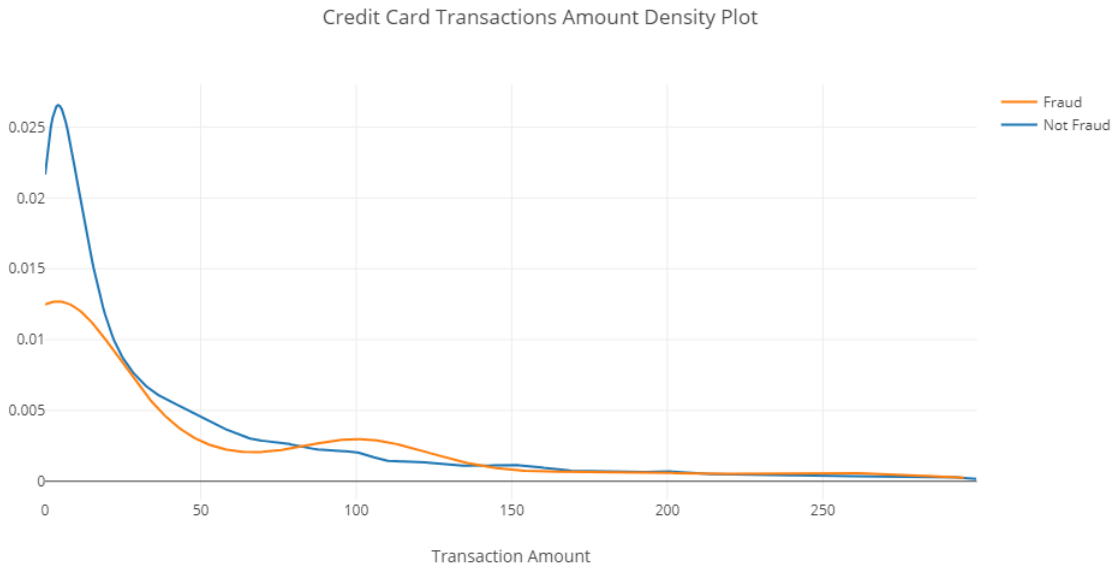

Figure 3.6: Transactions amount distribution

From the distribution plot of Transactions Amount in Fig3.6, it can be said that most transactions amount, regardless of the class, are between 0 and 150. Distinctly, fraudulent transaction have lower maximum amount value, this can be interpreted as the person committing fraud tries to keep a low profile by not flagging the system by making high amount purchases.

3.3 XGBoost Model Implementation

3.3.1 Splitting the data

Before we can train the model, we need to import our dataset to the jupyter notebook and split it into three subsets before supplying it to the model for training.

– Training Dataset: The sample of data used to fit the model.

- Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.

- Test Dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

We use the validation set to evaluate results from the training set. Then, we use the test set to double-check your evaluation after the model has ”passed” the validation set. Figure 3.7 shows this workflow.

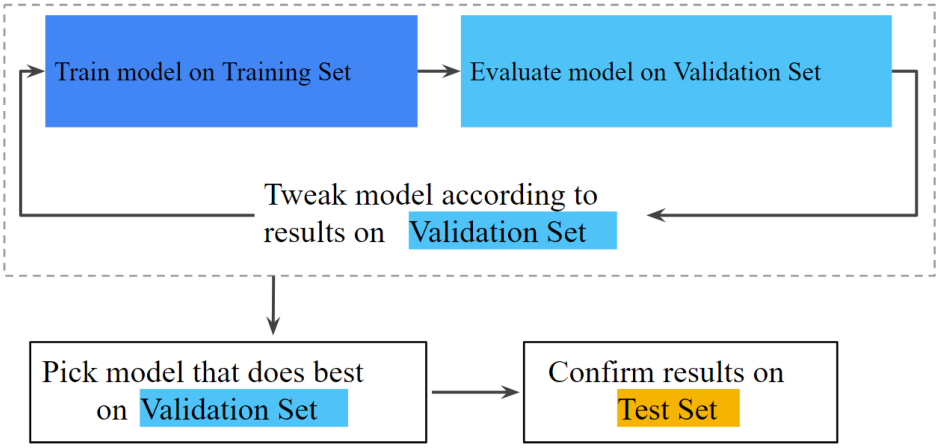

Figure 3.7: Train-Test split evaluation [28]

3.3.2 Tuning XGBoost Hyper-parameters

Next, we set up XGBoost using the hyper-parameters, we manually tuned the parameters and found the most optimal values.
The next step is to train the model with the training and validation sets. The code is shown in Code 3.1

code 3.1: Training the XGBoost model

```
model = xgb.train(params,
                  dtrain,
                  MAX_ROUNDS,
                  watchlist,
```

```
5                   evals_result=evals_result ,
6                   early_stopping_rounds=EARLY_STOP,
7                   maximize=True ,
8                   verbose_eval=VERBOSE_EVAL)
```

After the training has completed, we evaluate the model on the test set and show the classification report using Code 3.2

code 3.2: Testing the XGBoost model

```
1 predections = model.predict(dtest)
```

The model evaluation and the results for this model are addressed in the chapter of Results and Discussion.

3.4 XGBoost Model With Undersampling

3.4.1 Random Undersampling Method

This bias in the training dataset can influence the prediction of machine learning algorithms, leading some to ignore the minority class entirely. This is a problem as it is typically the minority class on which predictions are most important. Also, the data set is very large and it takes time to train the model. By reducing the size of data, the training time will also be reduced.

Since one of our objectives is to detect fraud transactions more precisely rather than improving the accuracy of the model, using undersampling is expected to help us to achieve better fraud detection as it doesn't make any changes in fraud data.

In this sampling technique, we randomly select the majority class instances and add them to the minority class. We add the number of majority instances in such a way that the majority and minority class ration becomes 1:1.

3.4.2 New Dataset Class Distribution

Figure 3.8 shows the new ditribution of the dataset.

Figure 3.8: Class distribution of the new Data

After performing random undersampling, we build our XGBoost model following the same steps as before.

3.5 Implementing Classification Models

Although our work did not focus on comparing classification models but rather on implementing an XGBoost model but we wanted to confirm the superiority of XGBoost when it comes to accuracy. We implemented 4 different classification models: Decision Tree Classifier, Random Forrest Classifier, Support Vector Machines (SVM), and K Nearest Neighbhors (KNN) Using the same data to compare their accuracy with that of XGBoost.

3.5.1 K Nearest Neighbhour (KNN)

A K Nearest Neighbhors algorithm, is an approach to data classification that estimates how likely a data point is to be a member of one group or another, depending on which group the data points closest to it are in.

KNN is a nonparametric lazy supervised learning algorithm. Nonparametric means that the model does not make any assumptions about the underlying data distribution. In other words, the structure of the model is determined by data. Laziness means no or very little training.

The algorithm flow is as follows.

The following operations are performed for each data to be predicted in turn:

- Calculate the distance between the point in the known training data set and the current point.

- Sort in order of increasing distance.

- Select k points with the smallest distance from the current point.

- Determine the frequency of the first k points in the category.

- Return the category with the highest frequency of the first k points as the predictive classification of the current point. Figure 3.9 shows a representation of K Nearest Neighbhour.

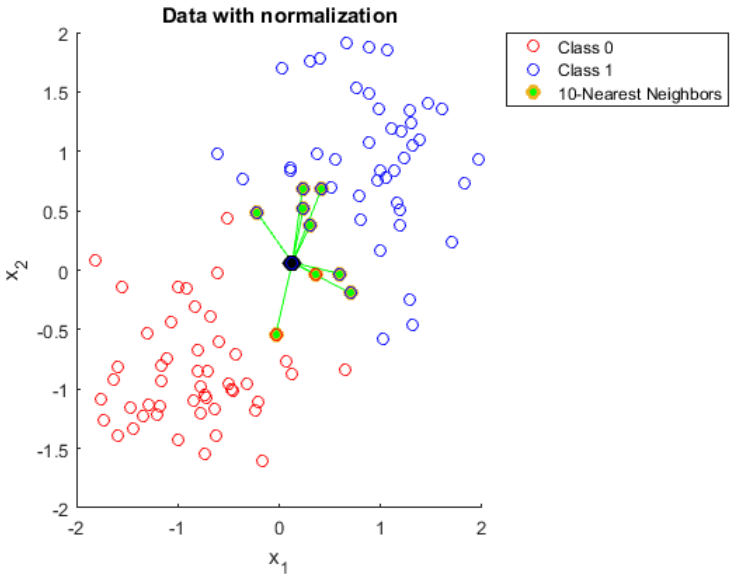

Figure 3.9: K Nearest Neighbhour representation [5]

3.5.2 Support Vector Machine (SVM)

The goal of the Support Vector Machine algorithm is to find a hyperplane in N-dimensional space (N - the number of features) that uniquely classifies the data points. To separate the two classes of data points, many possible hyperplanes could be chosen. Our goal is to find a plane that has the maximum margin, i.e., the maximum distance between the data points of the two classes. Maximizing the margin distance provides some gain so that future data points can be classified with more confidence. Figure 3.10 shows a representation of Support Vector Machine.

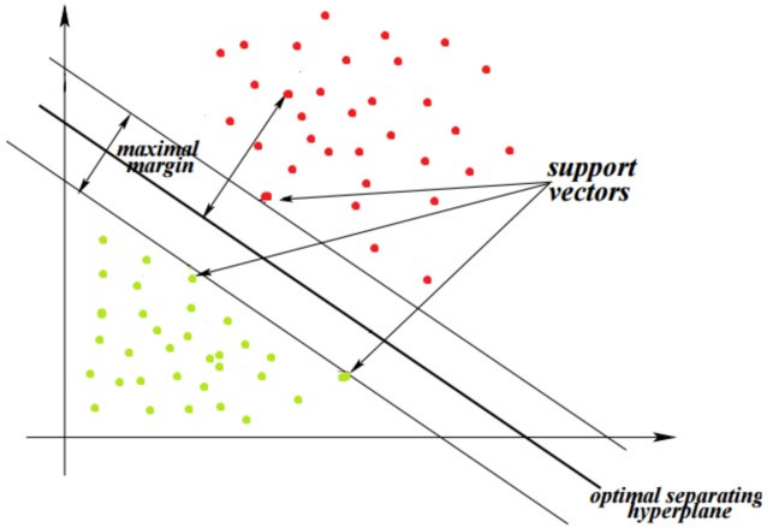

Figure 3.10: Support Vector Machine representation [4]

Conclusion

In this chapter, we have given an overview of the language, environment, and libraries used to implement our project. Next, we made a description of the dataset, where we considered the structure of the data and its distribution. Later, we explained in detail the process of preparing and training the XGBoost models and how we performed random undersampling on the data. Finally, we implemented 4 other classification models on the same data to compare their accuracies with XGBoost.

Chapter 4

EXPERIMENTAL RESULTS

Introduction

In this chapter, we will evaluate the performance of our two models: Model 1 (XGBoost without undersampling), Model 2 (XGBoost with undersampling) by checking the evaluation metrics presented in Chapter 2.

4.1 Experimental results

To be reminded of our objectives:
- High accuracy
- High fraud detection (True Positives) rate and low false alarms (False Positives), in other words increasing recall and precision scores
- Good fitting model.

4.1.1 Models evaluation

Confusion Matrices

After training the XGBoost models and fitting them on the testing data, we used a special function of the sci-kit learn metrics library to make the confusion matrices. The outputs are as shown in Table 4.1 and Table 4.2

Class	Predicted 0	Predicted 1
Actual 0	56769	95
Actual 1	13	85

Table 4.1: Confusion matrix: XGBoost without undersampling

Class	Predicted 0	Predicted 1
Actual 0	43	2
Actual 1	2	36

Table 4.2: Confusion matrix: XGBoost with undersampling

From the confusion matrix we get the evaluation metrics, Table 4.3 shows us the values of the evaluation metrics for the XGBoost model when we test it on the test dataset.

Metric	Without Undersampling	With Undersampling
Accuracy	0.998	0.95
Recall	0.87	0.95
Precision	0.47	0.95
F1	0.61	0.95

Table 4.3: Evaluation metrics results

As can bee seen in Table 4.3 accuracy of of model 1 **(0.998)** is greater than that of model 2 **(0.95)**. On the other side, when we applied undersampling technique we can observe a great improvement in recall and precision rates which have increased from **0.87** to **0.95** and from **0.47** to **0.95** respectively.

ROC Curve

In order to visualize the rate between true positive rate and true negative rate, we plotted the ROC curves in Figure 4.1.

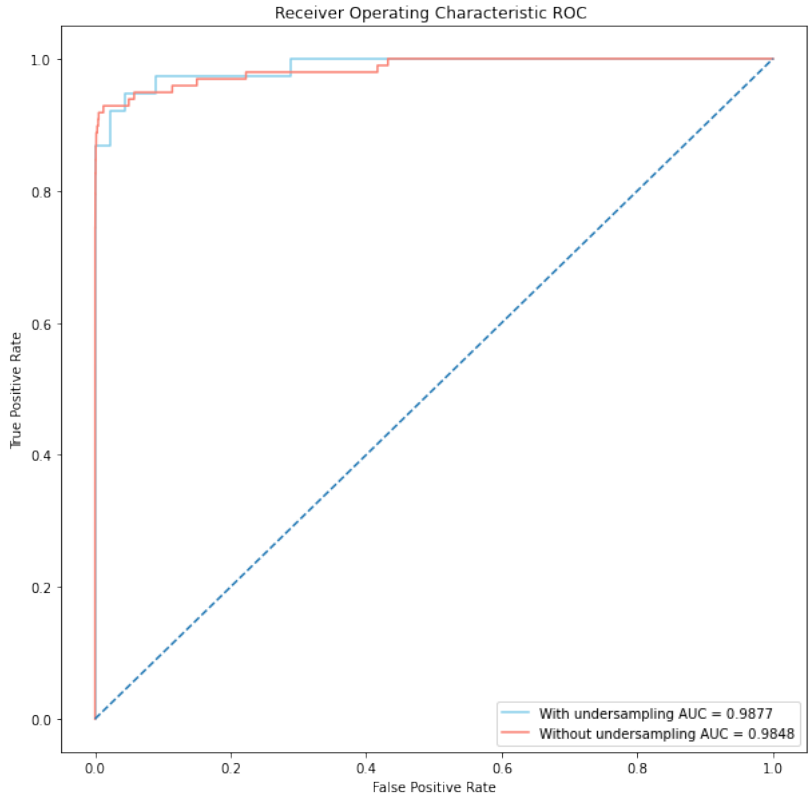

Figure 4.1: Comparison between ROC curves of the two models

From the curves, we can see that the XGBoost with undersampling slightly performs better than the XGBoost without undersampling. The first model has an **0.9848** auc accuracy while model 2 has **0.9877** auc accuracy.

Comparing the Learning Curves of the model 1 and model 2

Figure 4.2 and Figure 4.3 show the learning curves of model 1 and model 2 respectively.

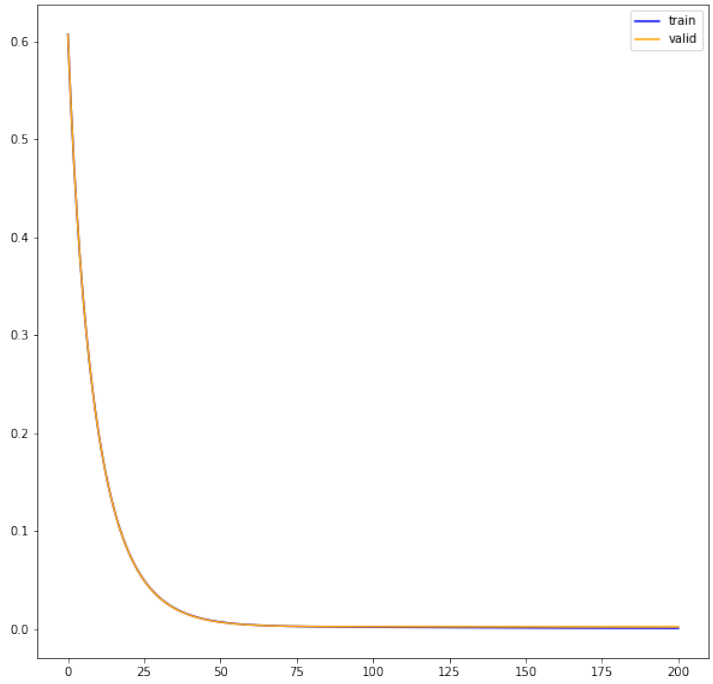

Figure 4.2: Learning curve: XGBoost model without undersampling

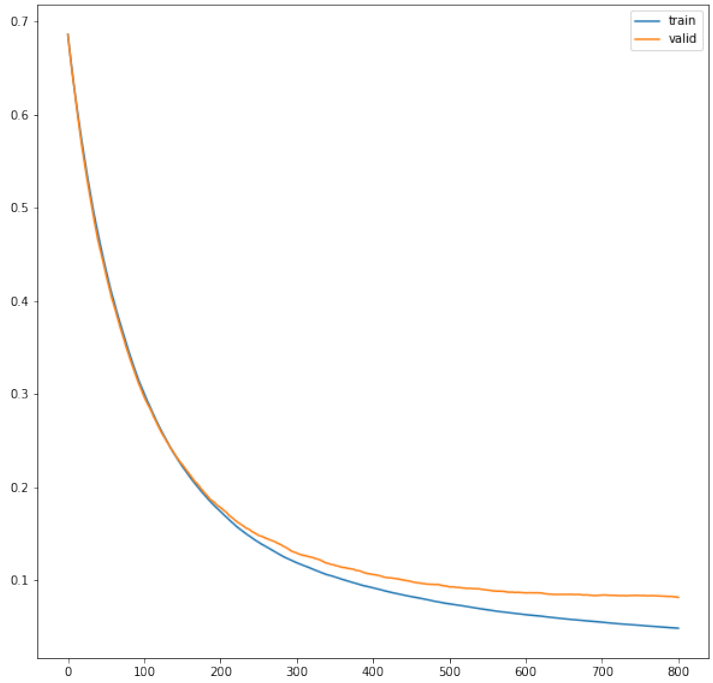

Figure 4.3: Learning curve: XGBoost model with undersampling

From the learning curves, we can say that model 1 (XGBoost without undersampling) has a great fit (no problem of overfitting or underfitting) because we can observe that the training loss curve decreases up to a point of stability and the validation loss curve also decreases up to a point of stability and has no gap with the training loss.

On the other hand, model 1 (XGBoost with undersampling), we notice an emerging gap between the training and validation learning curves, but a continuous decrease in logloss, indicating that the model is overfitting and is able to continue learning if more data points were used.

Performance of all the proposed models.

Table 4.4 summarizes the performance of all of the proposed models. We can clearly

Model	Training accuracy	Testing accuracy
XGBoost (no undersampling)	0.99	0.984
XGBoost (undersampling)	0.99	0.987
K Neighbhour	0.63	0.62
SVC	0.54	0.56
Decision Tree Classifier	0.90	0.92
Random Forest Classifier	0.94	0.93

Table 4.4: Summary of the performance of all the proposed models

see that the XGBoost classifier outperforms the other types of classification models, and this confirms what we expected at the beginning of our work. We notice that the two Decision Trees based classifiers (Decision Tree Classifier and Random Forest Classifier) performed better compared to other classifiers (SVC and KNN) but still did not mount to the performance of XGBoost.

4.2 Discussion

As expected, Extreme Gradient Boosting algorithm showed great performance in classifying the transaction class. It provided very accurate results due to its various techniques: regularization to avoid overfitting, parallel tree creation, built-in cross-validation function, and tree pruning. However, the true-positive rate was low. The imbalance of positive and negative examples in the training set effected the classifier's performance. Using subsampling improved the sensitivity of our classifiers by dealing with this imbalance and improved training time by reducing data set size.

Conclusion

This chapter focused on the results of the trained models. First, we showed the results of testing the model Extreme Gradient Boosting model with and without the subsampling technique on the test set. Then we compared the performance of the two approaches using evaluation metrics and learning graphs. Lastely, we compared performances of XGBoost with the other types of classifiers.

GENERAL CONCLUSION

This work aimed to build a machine learning model for Credit Card Fraud Detection. The Machine Learning model is based on a Gradient Tree Boosting algorithms called Extreme Gradient Boosting (XGBoost). For better sensitivity on top of the XGBoost system's optimization and algorithmic enhancements, the Random Undersampling method is used to deal with the imbalanced Data set problem. the results of the models showed an outstanding performance of the XGBoost algorithm; an AUC accuracy of *0.9848* with *0.87* sensitivity, especially when combined with the undersampling method, the accuracy and Recall increased to *0.9877* and *0.95* respectively. Moreover, XGBoost classifier outperformed the other classification models that we have built.

Future work

This work has the purpose of detecting credit card fraud, but it was done in an offline way. To stop fraud before it happens, our model should be deployed to analyze new transaction requests in real-time and make a decision; either allow the transaction to go through or subject it to authentication techniques. To achieve this, we propose to integrate the model with some of the analytics engines for big data processing such as Apache Spark and Flask.

Credit cards are not only used for online purchases, offline purchases at a payment terminal (POS) still account for a portion of the transactions made with credit cards. Nowadays, with the power of IoT, POS systems are undergoing major developments, one of them POS Fraud Prevention. The ability to use XGBoost to prevent credit card fraud onsite would be a great addition to the system.

Acronyms

AI Artificial Inteligence. 3, 4

API Application Programming Interface. 29

AUC Area Under Curve. 17

DL Deep Learning. 6, 27

DS Data Science. 27, 28

GB Gradient Boosting. 13, 19, 22–24, 29

GBDT Gradient Boosted Decision Trees. 29

GBM Gradient Boosting Machine. 6, 25, 29

GTB Gradient Tree Boosting. 22–24, 26

KNN K Nearest Neighbhors. 38

MART Adaptive Regression Trees. 21

ML Machine Learning. vii, 3–5, 18, 27–30

PCA Principal Component Analysis. 30, 31

ROC Receiver operating characteristic. vii, 16, 17, 42, 43

SVM Support Vector Machines. 38

SVM Support Vector Machines. 7

XGBoost Extreme Gradient Boosting. v, vii, 1–3, 7, 8, 13, 21, 22, 24–26, 28, 29, 32, 36, 38, 41, 43–47

Bibliography

[1] Rushin, G.; Stancil, C.; Sun, M.; Adams, S.; Beling, P. *Horse race analysis in credit card fraud-deep learning, logistic regression, and Gradient Boosted Tree. Systems and Information Engineering Design Symposium.* (2017).

[2] A. C. Bahnsen. *Detecting Credit Card Fraud Using Periodic Features.* (Dec, 2015). Available: `https://www.researchgate.net/publication/300414659_Detecting_Credit_Card_Fraud_Using_Periodic_Features`

[3] Yongbin Zhang; Fucheng You; Huaqun Liu. *Behavior-Based Credit Card Fraud Detecting Model.* (Aug, 2009). Available: `https://ieeexplore.ieee.org/document/5331646?denied=\`

[4] Satya Mallick. *Support Vector Machines (SVM).* (Jul 11, 2018). Available: `https://learnopencv.com/support-vector-machines-svm/`

[5] Develop Paper. *Machine Learning Sharing-KNN Algorithms and Numpy Implementation.* (Jul, 2017). Available: `https://developpaper.com/machine-learning-sharing-knn-algorithms-and-numpy-implementation/`

[6] Federal Bureau of Investigation. (n.d.).*Scams and safety, Credit Card Fraud.* Available: `https://www.fbi.gov/scams-and-safety/common-scams-and-crimes/credit-card-fraud`

[7] Y. Freund; R.E. Schapire. *A Short Introduction to Boosting.* (sep, 1999). Available: `https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf`

[8] Federal Trade Commission. (2020). *Consumer Sentinel Network Data Book 2019* [Online]. Available: `https://www.ftc.gov/system/files/documents/reports/consumer-sentinel-network-data-book-2019/consumer_sentinel_network_data_book_2019.pdf`

[9] Federal Trade Commission. (2021). *Consumer Sentinel Network Data Book 2020.* Available: `https://www.ftc.gov/system/files/documents/reports/`

consumer-sentinel-network-data-book-2020/csn_annual_data_book_2020.pdf

[10] Gregory Piatetsky. (2021). *Python Leads the 11 Top Data Science, Machine Learning Platforms Trends and Analysis.2019.* Available: https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html

[11] Numpy community. *What is NumPy.* (2021).Available: https://numpy.org/doc/stable/user/whatisnumpy.html

[12] Pandas development team. *Pandas documentation.*(n.d.). Available: https://pandas.pydata.org/docs/

[13] Seaborn Documentation. *An introduction to Seaborn.* (n.d.), Available: https://seaborn.pydata.org/introduction.html

[14] Python Software Foundation. *Python plotting package.* (n.d.). Available: https://pypi.org/project/matplotlib

[15] Plotly. *Plotly Python Open Source Graphing Library.* (n.d.). Available: https://plotly.com/python/.

[16] XGBoost developers. *XGBoost documentation.* (n.d.). Available: https://xgboost.readthedocs.io/en/latest/

[17] Sci-kit learn. *scikit-learn Machine learning in Python.* (n.d.). Available: https://scikit-learn.org/stable/

[18] Zakaria Jaadi. *A step by step explanation of Principal Component Analysis (PCA).* (April 7, 2021). Available: https://builtin.com/data-science/step-step-explanation-principal-component-analysis

[19] Pat Langley. *The changing science of machine learning.* (Feb 18, 2011). https://link.springer.com/content/pdf/10.1007/s10994-011-5242-y.pdf

[20] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge University Press, 2014. Available: https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf

[21] Sunil Ray. *commonly used machine learning algorithms*. (Sep 9, 2017). Available: https://www.analyticsvidhya.com/blog/2017/09/

[22] Nabipour. *Deep Learning for Stock Market Prediction*. (2020). Available: https://www.researchgate.net/publication/343339583_Deep_Learning_for_Stock_Market_Prediction

[23] Chris Nicholson. *A Beginner's Guide to Important Topics in AI, Machine Learning, and Deep Learning*. (n.d.). Available: https://wiki.pathmind.com/decision-tree

[24] Kunal Jain. *Learn How to Apply Tree Based Algorithms like Decision Trees and Random Forest*. (n.d.). Available: https://courses.analyticsvidhya.com/courses/ebook-tree-based-algorithm

[25] Aishwarya Singh. *4 Boosting Algorithms You Should Know GBM, XGBoost, LightGBM, and CatBoost*. (Feb 13, 2020). Available: https://www.analyticsvidhya.com/blog/2020/02/4-boosting-algorithms-machine-learning/

[26] Himanshu Sharma. *XGBoost The Miracle Worker*. (n.d.). Available: https://medium.com/almabetter/xgboost-the-miracle-worker-7518dd55abf3

[27] Leo Breiman. *Bagging Predictors*. (August, 1996). Available: https://link.springer.com/article/10.1023/A:1018054314350

[28] Google Developers. *Validation Set: Another Partition*. (n.d.). Available: https://developers.google.com/machine-learning/crash-course/validation/another-partition

[29] Google Developers. *Classification: ROC Curve and AUC*. (n.d.). Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

[30] Manpreet Singh Minhas. *Techniques for handling underfitting and overfitting in Machine Learning*. (n.d.). Available: https://towardsdatascience.com

[31] Sydney F. *Seeing the Forest for the Trees: An Introduction to Random Forest*. (Aug 3, 2019). Available: https://community.alteryx.com

[32] AIML. *Confusion about Confusion Matrix*. (Apr, 2019). Available: `https://blog.mur.li/confusion-about-confusion-matrix/`

[33] Rajasayanam. *A Measure of Bias and Variance: An Experiment*. (Dec 2, 2020). Available: `https://www.analyticsvidhya.com/blog/2020/12/a-measure-of-bias-and-variance-an-experiment/`