POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

A Fair-Generative approach for Customer Relationship Management

Supervisors

Prof. Daniele APILETTI

Ing. Andrea BARDONE

Dott. Simone MANNI

Candidate

Alberto MORCAVALLO

Academic year 2023/2024

Abstract

The advent of generative models has emerged as a fundamental shift of perspective in modern machine learning, with the potential to revolutionize a variety of domains. The generation of data that closely resembles real-world events has enabled the transformation of various businesses, increasing the possibility of building newer and more successful ones. In this work, we demonstrate the design and implementation of a machine-learning pipeline for financial CRM (Customer Relationship Management), specifically for predicting credit customer churning. The developed end-to-end pipeline proposes a predictive model with a high predictive capacity of recognizing churning behaviours that adheres to transparency and discriminant-free paradigms. The approach combines a state-of-the-art tabular generative procedure exploiting Generative Adversarial Network architecture to mitigate the dataset's lack of representative samples with major company service solutions such as IBM DataStage and H2O AutoML for effectively assessing the data ingestion and model creation phases.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In today's highly competitive business landscape, customer churn has become a significant concern for financial institutions. Being able to identify those warning signs proper of a customer's churning behavior is crucial for the development of corrective policies able to prevent possible supplier changes. The concept of "*Customer churn*" refers to the phenomenon related to customer relationship management (**CRM**) where customers discontinue their relationship with a company, resulting in a loss of revenue and market share. Identifying and predicting customer churn is of utmost importance as it enables financial organizations to take proactive measures to retain valuable customers and minimize the impact of customer attrition.

The managerial importance of customer churn analysis has been emphasized in several studies. "Gupta et al. (JMR, 2004)" [1] estimates that a 1% decrease in churn rate, on average, increases a company's profits by 5%. Moreover, customer loss leads to an increase in the need for attracting new customers [2] being five to six times more expensive than customer retention [3].
It follows that reducing abandonment rates becomes a primary objective for service providers in industries characterized by a high monthly or annual churn rate. This kind of phenomenon despite having an immediate impact on the duration of the company-customer relationship, is an essential determinant of customer lifetime value (**CLV**), which represents the total net revenue that the company can expect from the customer over the entire duration of their relationship.

Furthermore, it is not always possible to observe with certainty the moment when a customer terminates their relationship with a service provider. In cases where the relationship is contractual, explicit termination action by the customer is required. In the case of non-contractual relationships, customer deactivation may never occur, or it may be unilaterally decided by the company when a predetermined maximum threshold of waiting time since the customer's last action is exceeded.

In the case of contractual relationships, it is important to distinguish between situations where the customer's exit can occur at any point in the timeline (*continuous-time*), highlighting voluntary abandonment by a customer without substantial exit constraints and situations where it can only occur at predetermined points in the timeline (*discrete-time*), where a standard subscription contract allows termination of the relationship only at the end of the contract period.

The phenomenon of customer churn needs to be predicted with acceptable accuracy if a company intends to concentrate its retention efforts on customers at the highest risk of relationship termination. For this purpose Machine learning techniques have emerged as powerful tools for predicting customer churn in various industries, including the financial sector. By leveraging historical customer data and utilizing sophisticated algorithms, machine learning models can uncover valuable insights and patterns that aid in predicting the likelihood of customer churn. These models help financial institutions identify the factors contributing to customer attrition, enabling them to develop effective retention strategies and mitigate customer churn.

The financial industry, encompassing banks, insurance companies, credit card providers, and investment firms, holds vast amounts of customer data, including transaction history, demographics, customer interactions, and product usage. This data, when properly analyzed and processed using machine learning techniques, can provide valuable insights into the factors that drive customer churn. By accurately predicting churn, financial institutions can tailor personalized interventions, such as targeted marketing campaigns, improved customer service, or customized product offerings, to retain at-risk customers and enhance overall customer satisfaction.

The application of machine learning in customer churn prediction involves the utilization of various algorithms and techniques. These include logistic regression, decision trees, random forests, support vector machines (SVM), artificial neural networks, and gradient boosting methods. Each of these algorithms has its strengths and weaknesses, and the choice of the appropriate model depends on the specific characteristics of the financial dataset and the prediction goals of the institution.

Furthermore, feature engineering plays a vital role in customer churn prediction. By selecting and engineering relevant features from the available data, financial institutions can enhance the predictive power of machine learning models. These features may include customer demographics, transactional behavior, customer lifecycle events, customer complaints, usage patterns, customer engagement metrics, and other various derived customer attributes. Careful feature selection and engineering are crucial in capturing the underlying dynamics that contribute to customer churn and improving the accuracy of the prediction models.

To enforce those considerations, the feature engineering phase should also ensure ethical and unbiased decision-making, avoiding the possibility of discriminatory outcomes. In order to ensure fairness in customer churn prediction, it is essential to apply *fairness-aware* techniques during the entire machine learning pipeline, being able to provide a model which is aligned with ethical principles ensuring equal treatment for all customers. In this way, the decision-making process is a promoter of transparency, accountability, and trust through customers and regulations. Therefore, by addressing model fairness, businesses can mitigate the risk of legal challenges and potential reputational damage associated with discriminatory practices.

Based on these considerations the following study proposed the creation of a machine learning pipeline able to manage the phenomenon of *Credit Card Customer Churn* that goes beyond the prediction accuracy, being at the same time comprehensible and regulatory-compliant.

Chapter 2

The Imbalance Dataset Problem

2.1 Imbalance Dataset

Due to the nature of the churn phenomenon, all those analysis which aim to analyze the customer's likelihood to cancel their subscriptions or services, fall into the category of imbalance dataset problems.

The objective of churn prediction is to properly identify customers who are likely to leave a product or a service, by always referring to a situation where the distribution of the classes is skewed, with one class, typically not the one of interest, being significantly more prevalent than the other. Through the analysis of several studies facing this kind of situation, it has been possible to derive some fundamental characteristics of these problems when analyzed through machine learning techniques:

- **Imbalance Class Distribution**: The churned customers (minority class) are often not well represented, having cardinality outnumbered by the non-churned customers (majority class). Based on the proportion of minority classes different degrees of imbalance can be identified:

Degree of Imbalance	**Proportion of Minority Class**
Mild	20-40% of the Dataset
Moderate	1-20% of the Dataset
Extreme	<1% of the Dataset

Table 2.1: Imbalance Typologies

- **Model Bias**: Imbalance datasets can affect the model's training. If the model is not trained with a representative sample of the minority class, it may not learn enough about the churn patterns and it may struggle to make accurate predictions for the minority class.

- **Impact on Model Performance**: Class imbalance can negatively impact the performance of machine learning models. Since models are often biased towards the majority class, they may have lower sensitivity or recall for the minority class, resulting in a compromised model's ability to predict churn instances.

- **Evaluation Metrics**: Traditional evaluation metrics such as accuracy can be misleading in imbalanced scenarios. For example, if a model predicts all instances as non-churned, based on the degree of imbalance contained in the dataset, it may achieve a high accuracy value. However, the model would fail to capture churn events. Therefore, proper evaluation metrics able to catch this data distribution discrepancy are often more suitable.

Chapter 3

Imbalance Management

Data imbalance is a significant concern in research and development, as data is considered an asset and a crucial component of decision-making in education and business.

The inevitable impacts of this kind of phenomenon on the algorithm's performance [4], and on the efficiency of the predictive model's capability [5], have brought many scholars and researchers over the years to develop different approaches to improve the classification tasks involving imbalance situations.

Those methods can be divided into three categories:

- **Algorithm Level Methods**
- **Data Level Methods**
- **Generative Methods**

3.1 Algorithm Level Methods

Algorithm-level solutions aim to modify the classifier learning procedure, by adjusting the model's parameter settings to prioritize some predictions with respect to others. These methodologies do not cause any shifts in data distributions, being more adaptable to various types of imbalanced datasets, at the cost of being specific only for a given classifier type.

It is possible to differentiate this approach into:

- Weighted Learning
- Threshold Method

3.1.1 Weighted Learning

Weighted learning is a technique employed to tackle class imbalance in machine learning. It entails assigning varying weights or importance to different classes based on their imbalance within the training data. By assigning higher weights to the minority class and lower weights to the majority class, the classifier is trained to prioritize learning from and correctly classifying the minority class instances.

In weighted classification, the weights are typically incorporated into the training algorithm or the loss function used for optimization. The objective is to provide more significance to the minority class samples, enabling the classifier to concentrate on accurately classifying these instances. The specific weight assigned to each class depends on the severity of the class imbalance and the desired balance between precision and recall.

There are several approaches for implementing weighted classification:

- **Weighted Loss Functions**: The loss function utilized in the training algorithm is modified to include class weights. By assigning higher penalties for misclassifications of the minority class, the loss function encourages the classifier to prioritize the correct classification of minority class instances.

- **Sample Weights**: Each instance in the training data is assigned a weight based on its class. During training, these weights are used to adjust the contribution of each instance to the model's learning process. Instances from the minority class are assigned higher weights to increase their influence on the model.

Depending on the typology of the model involved, each has an inherent mechanism to handle weighted classification. For example, in support vector machines (SVM), the C parameter can be adjusted to control the weight assigned to different classes [6]. Similarly, decision tree algorithms often provide options to specify class weights or cost matrices [7].

Weighted classification enables the model to consider the imbalanced nature of the data and make more informed predictions. It can assist in enhancing performance metrics for the minority class, such as recall or sensitivity, which are often more crucial in imbalanced scenarios. However, it is important to carefully select appropriate weights that accurately represent the class imbalance and evaluate the model's performance on multiple metrics to ensure balanced performance across all classes.

3.1.2 Threshold Method

Some classifiers, like Random Forest and certain Neural Networks, provide a score indicating the likelihood or degree to which an example belongs to a particular class. This score-based ranking can be utilized to create multiple classifiers by adjusting the threshold for classifying an example as belonging to a specific class.

By varying the threshold, we can control the trade-off between precision and recall or the balance between false positives and false negatives. A lower threshold would result in more examples being classified as positive, potentially increasing recall but also introducing more false positives. Conversely, a higher threshold would lead to stricter classification, potentially improving precision but possibly decreasing recall.

This technique allows us to generate multiple classifiers with different thresholds, each catering to specific requirements or preferences. For example, if the focus is on minimizing false negatives, we can choose a lower threshold to capture more positive instances, even if it results in a higher false positive rate. On the other hand, if the emphasis is on precision and minimizing false positives, a higher threshold can be employed to ensure a more conservative classification.

By adjusting the threshold, we can obtain classifiers with varying levels of sensitivity and specificity, providing flexibility in addressing specific needs and balancing the desired classification outcomes [8].

3.2 Data Level Methods

Data level solutions geared towards the manipulation of data distribution in order to achieve a more equitable balance within the training dataset. This is achieved through various sampling methods that aim to create a more appropriate representation of data instances.

Based on the analysis of interest in managing class imbalance, it is possible to categorize the sampling strategies in two families of technique. These strategies, if necessary, can be used in conjunction depending on the specific characteristics of the dataset and the problem at hand:

- **Over-sampling**
- **Under-sampling**

3.2.1 Over-sampling

These techniques are focused on increasing the number of instances in the minority class, reducing the disparity ratio between classes. New generated data are appended to the primary dataset, keeping all the initial information preserved [3].

It is possible to distinguish different kinds of Oversampling techniques:

- **ROS**
- **SMOTE**
- **Bordeline SMOTE**
- **SVM SMOTE**
- **Kmeans SMOTE**
- **ADASYN**

Random Oversampling (ROS):

Random oversampling is a naïve resampling technique used to address imbalanced datasets. It involves randomly selecting examples from the minority class with replacements and adding them to the training dataset. This technique does not assume any specific data distribution or employ heuristics.

Random oversampling can be effective for those machine learning algorithms sensitive to skewed distributions, particularly when duplicated examples of a class can influence model fitting, such as gradient-based algorithms.

However, attempting to achieve a balanced distribution in severely imbalanced datasets can lead to overfitting the minority class. This occurs because by replicating exact copies of the minority class, the algorithm reduced its generalization capability, causing symbolic classifiers to construct seemingly accurate rules which perform poorly on validation and test sets [9].

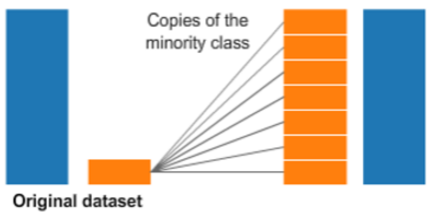


Figure 3.1: Random Overampling [10]

SMOTE:

The **Synthetic Minority Over-Sampling Technique** (SMOTE) [11] is a method used to address imbalanced datasets by increasing the representation of the minority class. It involves generating new synthetic examples in the "*feature space*" of the existing data.

SMOTE is based on the concept of introducing synthetic data points that represent the minority class. This is achieved by considering the minority samples and their nearest neighbours. Specifically, the generated examples lie along the line segments connecting each minority class sample with its k nearest neighbours.

The process can be divided into the following steps , which are repeated until the desired proportion of the majority/minority class is achieved:

1. Randomly select an instance point from the minority class.

2. Calculate the Euclidean Distance between the randomly chosen data point and one of its k nearest neighbours belonging to the same class.

3. Multiply the difference between the feature vector (sample) under consideration and the chosen nearest neighbour by a random number between 0 and 1.

4. Add the result to the minority class as a synthetic sample.

This process helps to learn more general regions for the minority class samples, rather than being overshadowed by the majority class samples around them. However, despite the synthetic data generation capability, the SMOTE technique shows some pitfalls:

- It effectively addresses between-class imbalance; however, ignoring within-class imbalance and small disjuncts. As a matter of fact, the random selection of minority class instances for oversampling is done with uniform probability, leading to areas where many minority samples have a high probability of being further inflated, while sparsely populated minority areas may remain sparse [12].

- Minority classes that are outlying and appear within the majority class can create a problem for SMOTE by creating a line bridge with the majority class. The method is susceptible to noise generation because it does not distinguish overlapping class regions from so-called safe areas. This issue becomes evident when linear interpolation is applied to a noisy minority sample located among majority class instances [13].

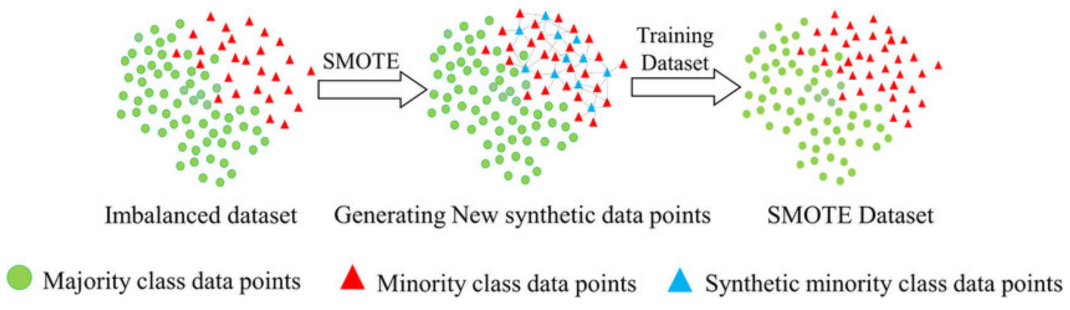

SMOTE
Training Dataset
Imbalanced dataset
Generating New synthetic data points
SMOTE Dataset
Majority class data points
Minority class data points
Synthetic minority class data points

Figure 3.2: SMOTE [14]

Algorithm 1 SMOTE Algorithm

```
1: procedure SMOTE($\alpha, k$)
2:     ▷ $\alpha$ is the proportion of minority class respect to majority after resampling
3:     ▷ $k$ is the number of nearest neighbors
4:     ▷ $i \leftarrow 0$
5:     ▷ $j \leftarrow 0$
6:     ▷ Initialization
7:     $M \leftarrow$ number of majority class samples
8:     $numattrs \leftarrow$ number of features
9:     ▷ Execution
10:    while $i \leq \alpha \times M$ do                    ▷ Generative Procedure
11:        Randomly select a point of the minority class:
12:        $X = \left[x^{(1)}, x^{(2)}, ...x^{(numattrs)}\right]$
13:        Compute its $k$ nearest neighbours
14:        Select a random number between 1 and $k$
15:        ▷ The selected nearest neighbor $X_{nn} = \left[x_{nn}^{(1)}, x_{nn}^{(2)}, ...x_{nn}^{(numattrs)}\right]$
16:        while $j \leq numattrs$ do
17:            Select a random number $gap^{(j)} \in (0,1)$
18:            Compute $x_{synth}^{(j)} = x^{(j)} + gap^{(j)} \times (x_{nn}^{(j)} - x^{(j)})$
19:        end while
20:        $X_{synth} = \left[x_{synth}^{(1)}, x_{synth}^{(2)}, ...x_{synth}^{(numattrs)}\right]$
21:        ▷ Add the new synthetic point to the the Original Dataset
22:    end while
23: end procedure
```

Borderline SMOTE:

Borderline SMOTE [15] varies the original SMOTE algorithm to learn the borderline between each class during the training process. The minority class examples are divided into different groups:

- **NOISE**: These instances are rare and likely incorrect. They are located in areas predominantly occupied by the majority class instances.
- **DANGER**: These instances are situated in the vicinity of class boundaries and often overlap with majority class instances.
- **SAFE**: These instances are relatively easier to recognize, and they serve as the primary representatives of the minority class.

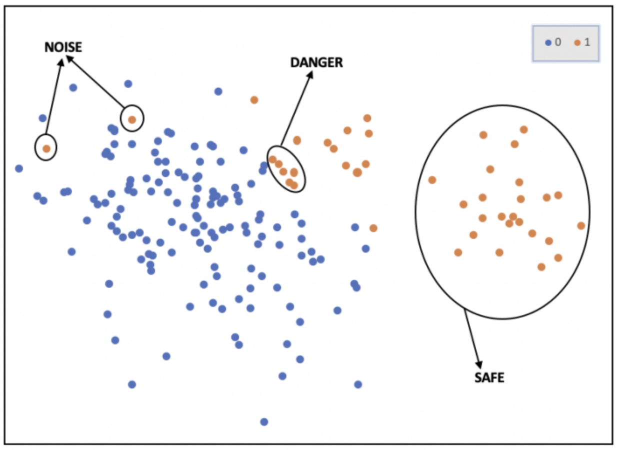


Figure 3.3: Borderline SMOTE clustering [16]

As reported in Algorithm2, during the training process, the resampling strategy attempts to learn the decision boundaries separating the different classes and so forth, the oversampling process is able to focus only on the minority class examples that are close to the bordelines, ensuring that noisy samples are not created within the majority class region or far from the boundary.

Therefore, after all instances of the minority class have been categorized, synthetic instances are then created along the line between DANGER instances and their nearest minority class neighbors.

Furthermore, to enhance the possibility to create new instances but leveraging also the area separating both classes, a variation of the original algorithm has been introduced, denoted as Borderline-SMOTE2. The new proposal generates synthetic data by creating a line between DANGER instances and their nearest neighbors by considering both sets P and N. However, to ensure that the newly created synthetic instances are closer to the minority class, the random number *gap* ranges between 0 and 0.5.

Algorithm 2 Borderline-SMOTE1

```
1: procedure BORDERLINE-SMOTE1($\alpha, k, m$)
2:     ▷ $\alpha$ is the proportion of minority class respect to majority after resampling
3:     ▷ $m$ is the number of nearest neighbors for point labeling
4:     ▷ $k$ is the number of nearest neighbors
5:     ▷ Initialization:
6:     $Q \leftarrow$ number of minority class samples
7:     $M \leftarrow$ number of majority class samples
8:     $P = \{ p_1, p_1, ..., p_Q \}$                    ▷ Minority class $P$
9:     $N = \{ n_1, n_1, ..., n_M \}$                    ▷ Majority class $N$
10:    ▷ Step 1:
11:    For every point $p_i$ in the minority class, calculate its $m$ nearest neighbours
       in the whole dataset.
12:    Set $m'$ the number of majority instances among $m$ nearest neighbours
13:    ▷ $m' \in [0, m]$
14:    ▷ Step 2:
15:    if $m' = m$ then
16:        $p_i$ is a NOISE point
17:    end if
18:    if $\frac{m}{2} \leq m' \leq m$ then
19:        $p_i$ is a DANGER point
20:    end if
21:    if $0 \leq m' \leq \frac{m}{2}$ then
22:        $p_i$ is a SAFE point
23:    end if
24:    ▷ Step 3:
25:    for all point $\in$ DANGER do
26:        SMOTE $(\alpha, k)$
27:    end for
28: end procedure
```

SVM SMOTE:

SVM SMOTE [17], in comparison to the previous technique, leverages the Support Vector Machine (SVM) algorithm to detect misclassified instances along the decision boundary.

By utilizing SVM, the algorithm identifies the decision boundary defined by support vectors. Subsequently, new instances are generated randomly along the lines connecting each support vector from the minority class with its nearest neighbors.

Algorithm 3 SVM-SMOTE

```
1: procedure SVM-SMOTE($\alpha, k, m$)
2:     ▷ $\alpha$ is the proportion of minority class respect to majority after resampling
3:     ▷ $k$ is the number of nearest neighbors
4:     ▷ $m$ is the number of nearest neighbor to determine if a minority sample is in DANGER
5:     ▷ Initialization:
6:     $SV^+ \leftarrow$ set of the minority class support vectors
7:     ▷ Step 1:
8:     For each $sv_i^+ \subseteq SV^+$ calculate the m nearest neighbours in the whole dataset.
9:     Set $m'$ the number of majority instances among $m$ nearest neighbours
10:    ▷ $m' \in [0, m]$
11:    ▷ Step 2:
12:    if $0 \leq m' \leq \frac{m}{2}$ then
13:        Apply the SMOTE Generative Procedure by considering as minority class point just the $sv_i^+$:
14:        $X_{synth} = sv_i^+ + gap \times (sv_i^+ - sv_{i,nn}^+)$        ▷ Extrapolation
15:        ▷ With $sv_{i,nn}^+$ one of the $k$ nearest neighbor of $sv_i^+$
16:    end if
17:    if $\frac{m}{2} \leq m' \leq m$ then
18:        Apply the SMOTE Generative Procedure by considering as minority class point just the $sv_i^+$:
19:        $X_{synth} = sv_i^+ + gap \times (sv_{i,nn}^+ - sv_i^+)$        ▷ Interpolation
20:        ▷ With $sv_{i,nn}^+$ one of the $k$ nearest neighbor of $sv_i^+$
21:    end if
22: end procedure
```

In order to expand the region of the minority class, despite the already exploited interpolation between data points, one noteworthy characteristic of this technique stands in the utilization of extrapolation to increase the minority class area even when surrounded by a large number of majority class points, enhancing the likelihood of encountering minority class instances in proximity to the optimal boundary. The number of neighbors used for extrapolation or interpolation depends on the density of the majority class instances surrounding the minority class instance.

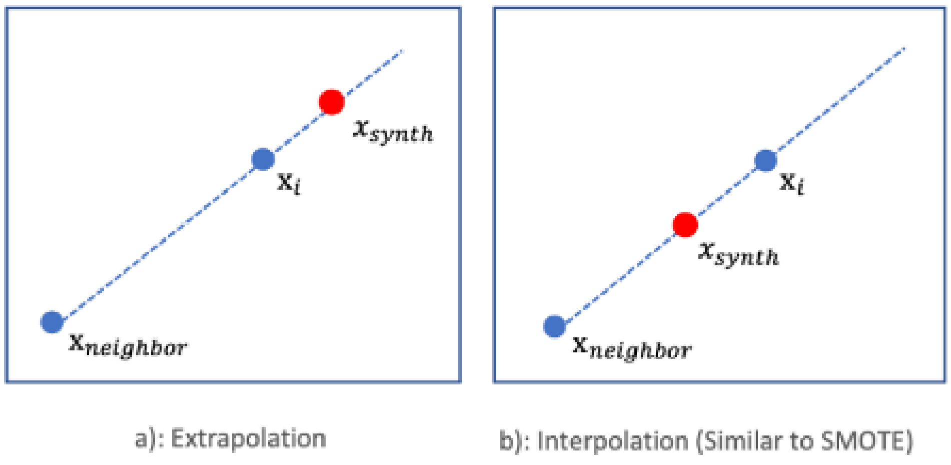


Figure 3.4: Extrapolation (a) and Interpolation (b) techniques used to generate synthetic data [16]

Kmeans SMOTE:

The **Kmeans-SMOTE** [18] algorithm combines the K-means clustering algorithm with SMOTE oversampling to achieve dataset rebalancing. This approach aims to address both between-class and within-class imbalance, taking into account both cluster's size and density and avoids data to be generated in safe areas. The algorithm comprises three main steps:

1. Clustering: Instances are grouped into k clusters using the K-means clustering algorithm.

2. Filtering: The clusters to be oversampled are selected, and the number of new instances to be generated in each cluster is determined. Cluster selection is based on the imbalance ratio (IR), proportion of minority and majority instances within each cluster. By default, clusters with at least 50% minority instances are chosen for oversampling. Sampling weights are assigned to the selected clusters to determine the number of instances to be generated.

3. Oversampling: SMOTE is applied to each selected cluster with $IR \geq 1$. For each chosen minority instance p_i within a cluster, a random minority neighbor p_{nn} is selected, and a new instance X_{synth} is generated by through interpolation. This process is repeated until the desired number of minority instances is reached.

The utilization of K-means clustering allows the oversampling algorithm to identify overlapping class regions and avoid data generation in safe areas, promoting the creation of new samples in sparse clusters. The filtering step plays a crucial role in the identification of the proper clusters for oversampling and in the determination of the number of samples to be generated in each cluster. By promoting the creation of new data in those clusters predominantly occupied by the minority class, this approach reduces the likelihood of noise generation, resulting in a more balanced distribution of samples within the minority class. Indeed, the new synthethic data are allocated primarily into sparse minority clusters rather than dense ones.

The selection of clusters for oversampling is adjusted through the imbalance ratio threshold (IR). Its increasing makes cluster selection more stringent, requiring a higher proportion of minority instances for a cluster to be chosen. Conversely, lowering the threshold relaxes the selection criterion, allowing clusters with a higher majority proportion to be selected.

To determine the distribution of generated samples, during the filtering step, sampling weights are assigned to each selected cluster.An high sampling weight indicates a low density of minority samples within the cluster, leading to more generated samples. The sampling weight depends on the density of an individual cluster relative to the average density of all selected clusters.

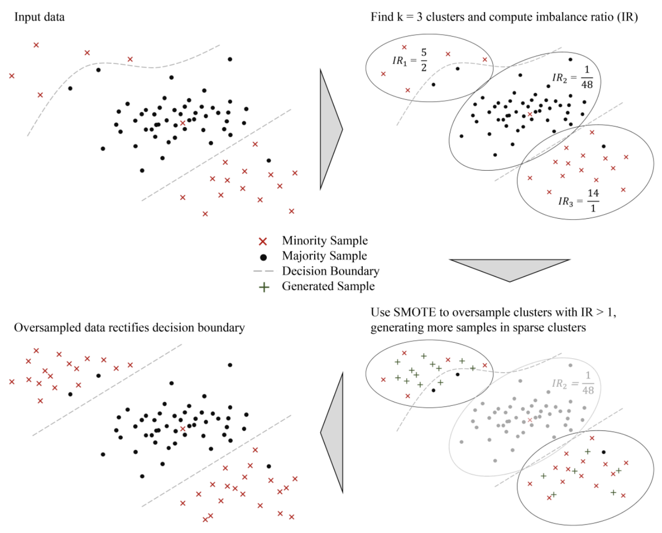

Input data
Find k = 3 clusters and compute imbalance ratio (IR)
$IR_1 = \frac{5}{2}$
$IR_2 = \frac{1}{48}$
$IR_3 = \frac{14}{1}$
Minority Sample
Majority Sample
Decision Boundary
Generated Sample
Oversampled data rectifies decision boundary
Use SMOTE to oversample clusters with IR > 1, generating more samples in sparse clusters
$IR_2 = \frac{1}{48}$

Figure 3.5: K-means SMOTE oversampling [18]

Algorithm 4 KMeans-SMOTE

```
1:  procedure ADASYN($\alpha, k_1, k_2, \beta$)
2:      ▷ $\alpha$ is the proportion of minority class respect to majority after resampling
3:      ▷ $k_1$ is the number of clusters to be used for K-Means
4:      ▷ $k_2$ is the number of nearest neighbors
5:      ▷ $\beta$ is imbalance threshold within cluster
6:      ▷ Initialization:
7:      $Q \leftarrow$ number of minority class samples
8:      $M \leftarrow$ number of majority class samples
9:      $P = \{ p_1, p_1, ..., p_Q \}$                              ▷ Minority class $P$
10:     $N = \{ n_1, n_1, ..., n_M \}$                              ▷ Majority class $N$
11:     $numattrs \leftarrow$ number of features
12:     ▷ **Step 1:**
13:     Apply K-Means($k_1$)                                       ▷ **Clustering**
14:     ▷ **Step 2:**
15:     For each cluster $c \in [1, k_1]$
16:     ▷ Compute $IR = \frac{Q_c}{M_c}$, with $Q_c$ and $M_c$ respectively the number of minority
        and majority class examples in each cluster $c$
17:     ▷ **Step 3:**                                               ▷ **Filtering**
18:     Select those cluster for which $IR \geq \beta$
19:     ▷ Set of selected cluster $F = \{f_1, f_2, ..., f_j\}$, $j \leq k_1$
20:     ▷ **Step 4:**
21:     For each selected cluster $f \subseteq F$      ▷ Determine the number of generated
        samples $\theta_f$
22:     ▷ Compute the Euclidean distance between all the minority samples in $f$
23:     ▷ Compute $averageMinorityDistance(f)$ , the mean distance within each
        cluster's minority samples in $f$
24:     ▷ $density(f) = \frac{Q_f}{averageMinorityDistance(f)^(numattrs)}$
25:     ▷ $sparsity(f) = \frac{1}{density(f)}$
26:     ▷ $w_f = \frac{sparsity(f)}{\sum_{l \in F} sparsity(l)}$
27:     ▷ **Step 5:**                                               ▷ **Oversampling**
28:     ▷ In each selected cluster $f$, the number of generated sample:
29:     $\theta_f = w_f \times \alpha \times M$
30:     ▷ For each selected cluster $f$ use the SMOTE **Generative Procedure**
        $\theta_f$-times
31: end procedure
```

ADASYN:

The **Adaptive Synthetic** (ADASYN) [19] is a sampling approach that aims to generate minority data samples adaptively based on their distributions. It focuses on generating more synthetic data for minority class samples that are harder to learn, as opposed to those that are easier to learn. Specifically, ADASYN generates additional synthetic data for minority class points that have a larger number of majority class points in their neighborhood.

The main benefit of ADASYN is that it not only reduces the learning bias introduced by the original imbalanced distribution but also dynamically adjusts the decision boundary to concentrate on the challenging samples that are harder to learn.

Differently from SMOTE, in which each minority class example has equal probability of being selected for generating a new synthetic data, ADASYN exploits the density of majority examples around the minority class r_i to determine the number of synthethic samples required. The density distribution r_i is used to assign weights to each different minority class example, based on the level of learning difficulty. Consequently, the resulting dataset achieves a balanced representation of the data distribution according to the defined balance level α, prompting the learning algorithm to focus on challenging examples.

By reducing the original imbalance dataset bias, and tuning the balance level α the algorithm can enhance the classification model's performance, reducing errors and improving results.

Algorithm 5 ADASYN

```
1:  procedure ADASYN($\alpha, k$)
2:      ▷ $\alpha$ is the proportion of minority class respect to majority after resampling
3:      ▷ $k$ is the number of nearest neighbors
4:      ▷ Initialization:
5:      $Q \leftarrow$ number of minority class samples
6:      $M \leftarrow$ number of majority class samples
7:      $P = \{ p_1, p_1, ..., p_Q \}$                    ▷ Minority class $P$
8:      $N = \{ n_1, n_1, ..., n_M \}$                    ▷ Majority class $N$
9:      $d = \frac{Q}{M}$                                  ▷ Degree of class imbalance $d$
10:     $G = M \times \alpha$                              ▷ Num synth examples to generate
11:     ▷ Execution:
12:     while $d \leq \alpha$ do
13:         For every point $p_i$ in the minority class
14:         ▷ calculate its $k$ nearest neighbours
15:         ▷ Calculate the ratio $r_i = \frac{\Delta_i}{k}$ , $i = 1...Q$, with $\Delta_i$ the number of majority
    class examples in the k nearest neighbours of $p_i$
16:         ▷ Normalize $r_i$
17:         $\hat{r_i} = \frac{r_i}{\sum_{i=1}^{Q} r_i}$
18:         ▷ For each $p_i$ retrieve the number of synthetic example to generate
19:         $g_i = \hat{r_i} \times G$
20:         For each $p_i$ generates $g_i$ synthetic examples using the SMOTE Generative Procedure
21:     end while
22: end procedure
```

3.2.2 Under-sampling

This family of techniques focuses on reducing the number of instances in the majority class. Data can be either randomly extracted or selected using specific methods to balance the class distribution. One of the main benefits of this approach stands in the reduction in training time for models, as the size of the training data is decreased, however, impacting the initial information contained in the dataset [20].

As for Over-sampling discussion, it is possible to distinguish several kinds of Under-Sampling methodologies:

- **RUS**
- **NearMiss**

- **Tomek-Links** (Discussed in Over Sampling + Under Sampling)

Typically, this family of methodologies by itself does not improve model's performances, however, their implementation in conjunction with Oversampling, depending on the characteristic of the dataset involved, is able to present important results, respect to the single techniques applied alone.

Random Undersampling (RUS):

Random undersampling involves randomly selecting examples from the majority class and removing them from the training dataset. Like ROS, it is considered a "naïve resampling" technique.

This kind of approach is suitable for datasets where a class imbalance is present, but the minority class is sufficiently represented by a substantial number of examples. Moreover, the main limitations of this undersampling methodology stands in the potential loss of majority class examples that could have been valuable or crucial for fitting robust decision boundaries. This can be problematic as it makes it more difficult to learn the decision boundary between minority and majority instances, leading to a decrease in classification performance [20].

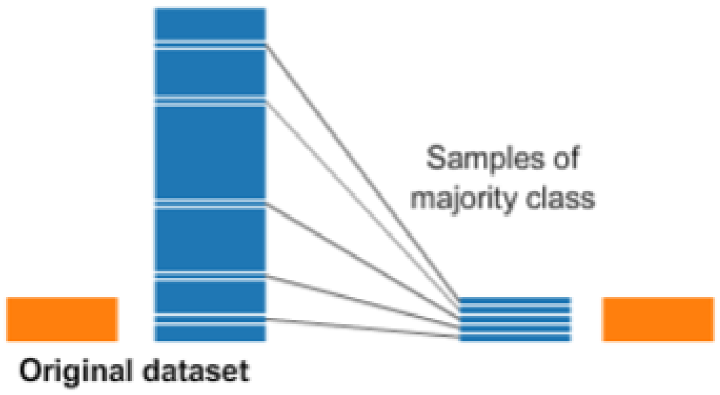


Figure 3.6: Random Undersampling [10]

NearMiss:

The Near Miss [21] undersampling methods are a set of techniques based on the KNN approach. The key idea is to increase the separation between the minority and majority classes, particularly when instances from different classes are in close proximity.

To achieve this, three different Near Miss methods can be employed:

- NearMiss-1: This method selects negative examples (majority class) that are close to some positive examples (minority class). Specifically, it chooses the majority class samples for which the average distances to the K closest minority class samples are the smallest.
- NearMiss-2: This method selects negative examples that are close to all positive examples. It identifies the majority class samples for which the average distance to the K farthest minority class samples is the smallest.
- NearMiss-3: This method involves a two-step algorithm. First, for each minority sample, the M nearest neighbors are retained. Then, among the selected majority samples, the ones with the largest average distance to the K nearest neighbors are chosen. This guarantees that every minority class instance is surrounded by examples from the majority class.

Compared to other undersampling methods, NearMiss-3 is anticipated to have a higher precision and lower recall. Its objective is to maintain a balance between retaining pertinent information and diminishing the dominance of the majority class during training.

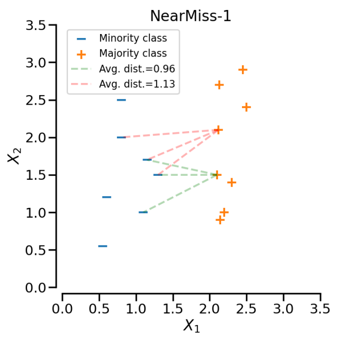


Figure 3.7: NearMiss-1

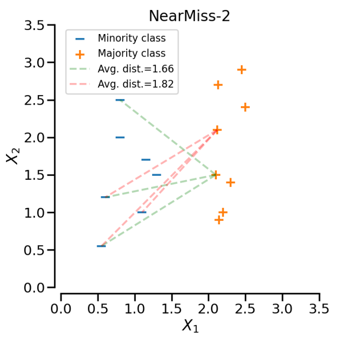


Figure 3.8: NearMiss-2

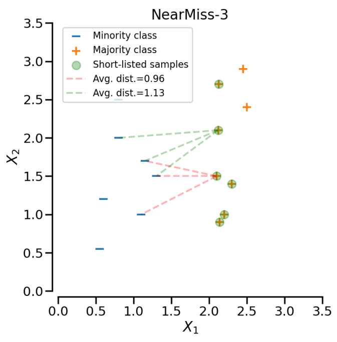


Figure 3.9: NearMiss-3

3.2.3 Over Sampling + Under Sampling

In order to derive benefits from both approaches, different studies have proposed to sequentially apply over and under-sampling methodologies to properly re-balance the initial data distribution.

SMOTE-Tomek:

SMOTE-Tomek is a hybrid sampling technique that combines the SMOTE (Synthetic Minority Over-sampling Technique) and Tomek Links methods to address the issue of class imbalance in datasets. It aims to improve the classification performance by simultaneously oversampling the minority class using SMOTE and undersampling the majority class using Tomek Links [22].

Tomek Links identifies pairs of very close instances, each belonging to a different class, and eliminates the majority class instances from these pairs. This process increases the separation between the two classes, which can facilitate the classification process.

A Tomek's link is defined as a pair of instances that are nearest neighbours to each other. Specifically, the two samples are the closest instances among their respective classes. By removing the majority class instances from Tomek Links pairs, the space between the minority and majority classes is expanded, enhancing the distinction between them.

This technique is derived from a modification of the Condensed Nearest Neighbors (CNN) undersampling technique developed by Tomek [23]. Unlike the CNN method, which randomly selects samples along with their K nearest neighbours from the majority class, the Tomek Link method applies specific rules for data selection.

As a matter of fact, a pair of observations is considered a Tomek Link if they satisfy two properties:

- Both observations are neighbours of each other
- The observations belong to different classes.

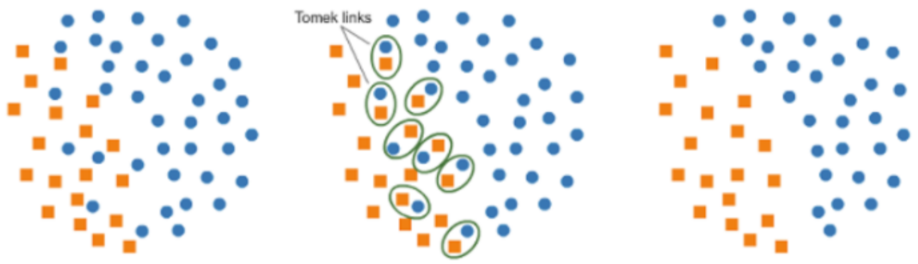


Figure 3.10: Tomek Links [24]

The objective of using Smote together with Tomek Links is to improve the minority class representation while removing majority class instances that are in close proximity to the minority class. By eliminating these samples, which represent potential borderline or overlapping instances, the resulting dataset aims to have a greater emphasis on the minority class and achieve a clearer separation between the targets.

3.3 Generative Methods:

Recently, generative methods have been proposed as a class of deep learning models able to learn the sample data distribution and exploit it to generate complex, high-dimensional data. Basically, they can be divided into two typologies:

- **Explicit Likelihood Models**: Being trained using a maximum likelihood approach, they are able to learn the data distribution from the sample and generate new types of data (Mixture Models, WaveNet, Autoregressive Language Models, Variational autoencoders, etc).
- **Implicit Likelihood Models**: Follow a more generalized approach able to generate new samples of data without depending on the probability distribution, but rather learn the data's statistical properties.

Based on the idea that traditional data-level re-sampling techniques rely on concepts that might fare poorly on high-dimensional datasets like nearest neighbours and simple linear interpolations. Recently, Generative Adversarial Networks have been proposed to generate complex, high-dimensional data and could, in principle, be used to generate additional minority examples.

3.3.1 Generative Adversarial Network: Background

Initially described in a 2014 paper by *Ian Goodfellow* [25] and mainly based on the computer vision domain, GANs are implicit likelihood models that generate data samples from the statistical distribution of the data. They are used to resemble variations within the dataset, combining two networks, **Generator** and **Discriminator**.

The Generator $G(z)$ takes random noise vector z as input and produces new data. The Discriminator D, on the other side, is a classifier that has to determine if the sample is real data x or generated by $G(z)$.

The two models are trained together in a zero-sum game (**adversarial**) until the generator model generates plausible examples able to fool the discriminator. Based on the Discriminator's capability of recognizing real and synthetic images, the weights of the two neural networks are updated, and the process continues iteratively to the point where the zero-sum game reaches the so-called Nash equilibrium in which the Generator produces images with such fidelity that the Discriminator cannot tell the difference between a forgery and a true copy.

Both Generator and Discriminator have their own loss function, but differently one from the other, their aim lies in quite opposite objectives:

- **Discriminator:** Maximize the chance of recognizing real data with respect to generated ones
$$\underset{D}{max} V(D) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{3.1}$$
- **Generator:** Generates data with the highest possible value of $D(G(z))$ to fool the Discriminator
$$\underset{G}{min} V(G) = \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{3.2}$$

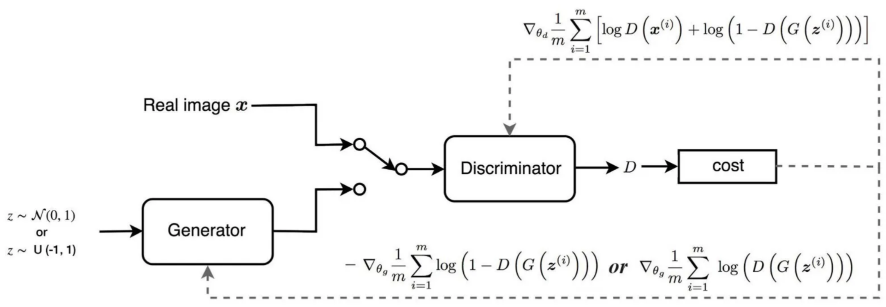

$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$
Real image $\boldsymbol{x}$
Discriminator
D
cost
$z \sim \mathcal{N}(0, 1)$
or
$z \sim$ U (-1, 1)
Generator
$-\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$ or $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$

Figure 3.11: Vanilla Generative Adversarial Network Learning Framework[26]

Despite the initial idea, over the years, the GAN basic structure has been re-engineered to adapt to different domains' specific tasks and to overcome many initial limitations, including vanishing gradients, mode collapse, and failure convergence. The modifications applied tend to come in two forms: loss variants and architecture variants.

In the following is presented a short description of the related works which have involved the development of generative adversarial networks in different domain settings regarding the generation of synthetic tabular data addressing also the class imbalance.

3.3.2 GAN for Tabular Data generation:

The heterogeneity characteristics of most structured data representations have made the development of a general-purpose GAN able to reliably work on tabular data a non-trivial job. Over the years, several approaches have been proposed with the aim of generating realistic privacy-preserving synthetic structured data:

- **table-GAN** [27] has been one of the first approaches highlighting the need for a design architecture specifically for tabular data. It consists of a readaptation of the synthetic **Deep Convolutional GAN** [28] architecture designed for synthetic image generations to structured data. The approach is based on transforming each row from the structured data into a square two-dimensional matrix which can be processed by a 2D-convolutional layer. To manage categorical features, a simple label encoding pre-processing is applied, and the generated output is just rounded to the nearest integer. However, due to its instrinsic nature of adaptability in those data domains in which relevant patterns can appear anywhere in an image, the arbitrary features order of structured data and their heterogeneous information content fully discourage the proposal.

- Many synthetic data generation approaches have been exploited inside the medical field generating different GAN architectures, among which major ideas have been developed in the **medGAN** [29] model, considered as a baseline in literature. Initially focused on the generation of discrete medical records adopting of a pretrained autoencoder structure , study involving the replacement of the vanilla GAN loss function with the Wasserstein GAN Gradient Penalty loss has been able to show improvements in the data generation process[30].

- **TGAN**[31] and **CTGAN**[32] proposed synthetic data generation providing differentiation between attribute typologies. The former uses LST-Cells with learned attention-weights in the generator to generate columns one-by-one, while the latter abandons the LSTM-Cells architecture in favour of fully-connected layers, which use a one-hot encodings representation and SoftMax activation function with added uniform noise to the output for categorical columns and a gaussian mixture model-based approach for numerical columns.

3.3.3 GAN for Tabular Data Imbalance Oversampling:

Despite the recognized importance of GAN architectures for data augmentation exploited in the context of unstructured data, such as images. During those years, a small literature on using GANs for generating minority class instances by oversampling structured data has also emerged:

- *Fiore et all* [33] proposed to exploit the basic GAN framework to train the model only in the minority samples. By doubling the number of minority instances it has been able to outperform the SMOTE methodology results, however, without providing a proper model evaluation criteria.

- *Yu-Jun Zheng et all* [34] proposed a complex classification framework which used a denoising autoencoder in conjunction with a GAN and a Gaussian Mixture Model as discriminator and classifier. Moreover, to convert categorical variables to numerical format it uses a weight-of-evidence transformation, but without exploiting GAN categorical variables generation capacity.

- *Georgios Douzas* and *Fernando Bacão* [35] have been the first to adapt the **Conditional GAN**(*cGAN*) to manage class imbalance. However, the benchmarking dataset exploited just numerical features.

- **TabGAN** [36] proposed a resampling behaviour coordinating the *CTGAN* application together with adversarial training evaluating both training and test sets. The approach consists of the generation of synthetic data from the initial training dataset exploiting the *CTGAN*. Those new data are concatenated together with the training dataset and developed an adversarial boosting to obtain data resembling the test set, without taking into consideration the original ground truth. The final resampled dataset is composed of the top rows from the training set and the synthetic dataset sorted by correspondence to the test set.

- **c-WGANGP** [37] adapts GAN concepts of synthetic data generation to the oversampling task. Having estimated the data distribution through a pre-trained Auxiliary Classifier, the GAN architecture is conditioned on generating minority class samples. To optimally solve the GAN's min-max game, the Wasserstein GAN Gradient Penalty loss function is provided enabling both Discriminator and Generator to train to optimality, ensuring that the generated data resemble the original data distribution by improving training stability, providing more gradient info to train the Generator, which leads to the creation of samples that are more spread out across the data distribution, rather than focusing on just a few modes. Moreover, some proposed adjustments are found in the introduction of the Gambel-softmax activation function in combination with embedding layers to model categorical columns while adding noise to the numerical data to avoid the Discriminator detection of trivial patterns.

3.3.4 c-WGANGP:

Among the presented approaches for managing class imbalance in a structure data scenario using a generative adversarial network framework, the interesting results achieved by cWGAN in managing tabular data oversampling on non-linear datasets [37] have brought us to analyze the general model's structure and readapt the model into our work.

However, despite the Vanilla GAN composition, to shift the model's goal into the generation of new tabular data constrained to the minority class, the selected architecture presents many modifications of the original generative adversarial network structure from both architectural and objective points of view.

Wassertein Gradient Penalty Loss:

The architecture proposed by Ian Goodfellow was driven by an optimization requirement only achievable through minimization of the Jensen-Shannon Divergence (JSD) between real and synthetic data, possible only in a situation in which p_g, the generator's distribution over the data space X , corresponds to the real data distribution p_{data}.

However, as suggested by the same author, the approach did not ensure convergence to equilibrium, with the losses of both Generator and Discriminator not necessarily correlated with sample quality, allowing for mode collapse [25]. This phenomenon is a quite common issue in generative modelling, meaning that the Generator fails to capture the full diversity of the data distribution, by focusing on a limited subset of samples it often produces very similar or even identical outputs for different inputs, which results in solving the max-min game. Therefore, the Generator only learns to generate a few specific patterns or modes from the data distribution, bringing a lack of diversity in the created samples, resulting in poor-quality output and limited variability in the data.

In order to provide the generative network with sufficient guidance to explore diverse modes in the data distribution, the WGAN [38] architecture proposed the optimization of the Wasserstein-1 distance (Earth-Mover's distance). Defined as the "minimum cost" of the optimal transport plan to move the probability mass of one distribution until it matches another distribution. The adoption of such a strategy smooths the generative process to not completely mimic the original data distribution, but instead, find something close to it, providing:

- **Improved Gradient Flow**: Smoother gradients compared to traditional GAN loss functions
- **Mode Diversity**: Encourages the Generator to capture a wider range of modes in the data distribution, focusing on sample quality.
- **Training Stability**: Improved training stability compared to traditional GANs, being less susceptible to vanishing gradients, while contributing to avoiding mode collapse.

The achievement of such distance metrics is approximated by changing the GAN objective to:

$$\underset{G}{min}\underset{D}{max}\ \mathbb{E}_{x\sim p_{data}}\left[D(x)\right] - \mathbb{E}_{z\sim p_z}\left[D(G(z))\right] \tag{3.3}$$

To measure the quality of the generated samples and real data, the Discriminator is constrained to be 1-Lipschitz continuous. This is achieved by:

- Clipping the Discriminator's weights within a range [-c,c], avoiding exploding gradients.
- Elimination from the Discriminator final layer of Sigmoid Activation function.

In this way, the predictions no longer fall in the range [0,1], but instead, the network outputs a scalar score between $(-\infty, \infty)$, interpreted as how much the input data resembles reality. For this reason, the Discriminator in a WGAN is usually referred to as a Critic [38].

	Discriminator/Critic	**Generator**
GAN	$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))\right]$	$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\log(D(G(z^{(i)})))$
WGAN	$\nabla_{\theta_w}\frac{1}{m}\sum_{i=1}^{m}\left[f(x^{(i)}) - f(G(z^{(i)}))\right]$	$\nabla_{\theta_w}\frac{1}{m}\sum_{i=1}^{m}f(G(z^{(i)}))$

Table 3.1: Loss Functions comparisons in GAN and WGAN architectures

Since weight clipping constraints the discriminator's capacity [38], being able to provide a good performance of the critic is fundamental for the WGAN. Without accurate gradients flow, the Generator cannot learn how to update the weights to create better samples over time. To overcome the weight clipping, in the WGAN-GP [39] architecture, the 1-Lipschitz constraint is enforced directly during the training with the adoption of a gradient penalty term.

A differential function f is 1-Lipschitz if it has gradients with a magnitude of at most 1 everywhere in the input space. To enforce such constraint, the gradient penalty method introduces a regularization term into the Discriminator's loss function. The idea is to use linear interpolation to create a smooth path between real and synthetic data points, and then calculate the gradient of the discriminator's output along this path, obtaining information on how the discriminator's response changes as we move from real to synthetic data. By penalizing the norm of the gradient, the Discriminator is encouraged to have smooth gradients and be Lipschitz continuous:

$$\underset{G}{min}\underset{D}{max}\ \mathbb{E}_{x\sim p_{data}}\left[D(x)\right] - \mathbb{E}_{z\sim p_z}\left[D(G(z))\right] - \lambda\mathbb{E}_{\hat{x}\sim p_{\hat{x}}}\left[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2\right] \tag{3.4}$$

With:

- $\lambda =$ Penalty Coeffiecient
- $\hat{x} = t\tilde{x} + (1-t)x =$ Interpolated point, $0 \leq t \leq 1$

Moreover, to ensure consistent gradient magnitudes, batch normalization is avoided for the Critic, since it can introduce scaling factors that vary across layers and mini-batches, making it challenging to ensure the Lipschitz constraint, impacting the effectiveness of the gradient penalty.

Being employed together with the Wasserstein GAN framework, the gradient penalty prevents mode collapse and vanishing gradients, by promoting the generation of samples that are more spread out across the data distribution, rather than focusing on just a few modes [40].

Conditional GAN with Auxiliary Classifier:

To further improve the training stability and specifically generate data belonging to the interesting class label, the GAN structure is ulteriorly modified by combining the idea proposed in two major works, cGAN and AC-GAN [[41],[42]].

Conditional GAN (cGAN) is a simple variation of the GAN objective by conditioning both Generator and Discriminator on the class labels, achieved by appending the class label y to both the networks' inputs. By letting the Generator to estimate the distribution $p_{X|y}$ and the Discriminator to learn to estimate $D(X,y) = P(fake|X,y)$, the Vanilla framework objective function can be reformulated:

$$\underset{G}{min}\underset{D}{max}V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x|y)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z|y)))] \quad (3.5)$$

Conditioning makes it possible for the Generator to sample only outputs belonging to a specific class while the Discriminator checks if the generated output is real or fake while conditioning the data generation process on matching the class label.

The Auxiliary Classifier GAN (AC-GAN) is a further extension of the GAN architecture based on ideas exploited in the cGAN configuration. Like conditional GAN, the generator model is provided with a point in the latent space and the class label as input, while differently the Discriminator is not conditioned, but instead is able to discern the difference between real and fake data while also predicting the class label of the given sample $D(X) = P(y|X)$, working as an "Auxiliary Classifier".

c-WGANGP objective:

Having synthesized the different approaches exploited in the designing of the GAN architecture objective function, the proposed configuration tries to properly combine all of them. The c-WGANGP architecture departs from the original AC-GAN formulation, presenting two separate network for both Auxiliary Classifier and Discriminator facilitating the independent utilization of the AC loss while simultaneously conditioning the Discriminator on the class label:

$$\underset{G}{m}in\underset{D}{m}ax\mathbb{E}_{x\sim p_{data}}\left[D(x|y)\right]-\mathbb{E}_{z\sim p_z}\left[D(G(z|y))\right]-\lambda_{GP}\mathbb{E}_{\hat{x}\sim p_{\hat{x}}}\left[(\|\nabla_{\hat{x}}D(\hat{x}|y)\|_2-1)^2\right]+\lambda_{AC}\mathbb{E}_{z\sim p_z}\left[AC(G(z|y)))\right] \tag{3.6}$$

With:

- AC = The Binary Cross Entropy of the Auxiliary Classifier between the True class label and the Predicted class probability y_{pred}.

Differently from the Vanilla framework in which the Generator's ability of creating new data is dependent only the Discriminator performances, the adoption of the Wassertein Gradient Penalty loss together with the adoption of an Auxiliary Classifier enforces the generative process by constraining the Generator's to provide synthetic data affine to real ones both in distribution and relative class.

Categorical Data Management:

To optimize the Generative Adversarial Network objective function through gradient learning, both the Generator and Discriminator need to be fully differentiable. Therefore, discrete outputs generation can be a bottleneck for neural network training. To backpropagate the gradients through discrete random variables, approximating the sampling from a discrete categorical distribution, the Generative model leverages the Gumbel-Softmax [43] function. Such activation function allows the model to sample from a discrete distribution during the forward pass, respecting at the same time the differentiality constraint:

$$\textbf{Gumbel-softmax}(x_i)=\frac{e^{(x_i+g_i)\tau}}{\sum_{j=1}^{k}e^{(x_j+g_j)\tau}}\quad\text{for } i=1,...k$$

The Gumbel-Softmax is a variation of the Softmax function, in which noise sampled independently from a Gumbel Distribution (0,1) is added to the logits, with the entire process parametrized by a temperature parameter τ controlling how closely the Gumbel-Softmax distribution approximates the categorical distribution. If τ is very small, we get very close to a quantized categorical sample, and conversely, the Gumbel-Softmax distribution becomes more uniformly distributed as τ increases.

Discriminator's categorical inputs are are encoded through an one-hot representation, which strongly differs from the "soft" one-hot encoding provided by the Gumbel-Softmax Generator's activation function. To make this distinction harder to recognize and to reduce the high-dimensional feature space representation achieved by one-hot encoding large feature levels, Embedding layers are employed to convert sparse data into a dense vector of real values:

$$x_{\text{embed},i} = W_{\text{embed},i}\ x_i$$

Where:

- $x_{\text{embed},i}$ is the embedding vector.
- x_i is the binary input of the i-th feature.
- $W_{\text{embed},i} \in \mathbb{R}^{n_e \times n_v}$ is the embedding matrix optimized with the network with n_e and n_v respectively the embedding and feature vocabulary sizes.

Numerical Data Management:

In order to represent numerical values, a slight amount of Gaussian noise (0, 0.01) is introduced to the numerical columns after being rescaled through min-max normalization. Independent Gaussian noise is applied to both real and generated samples, preventing the Discriminator from simply rejecting values that don't exactly match frequent modes in the real data, meanwhile, incentivizing the Generator to produce values that align with frequently occurring modes in the real data.

Moreover, to manage possible correlations within numerical and categorical columns, the author proposes a “self-conditioning” approach allowing the generator to capture relationships between columns more easily. By concatenating the categorical outputs to the numerical output layer, the numerical variables generation process is constrained to be dependent on the sampled categorical output.

Cross Layers:

To better model the relationships between variables, both Generator and Discriminator architectures present Cross layers [44]. Particularly used when dealing with complex relationships or dependencies between features in different parts of the input data, Cross Layers/Cross Network, are architectural pattern designs exploited in neural networks to explicitly capture feature interactions between different parts of the network, facilitating the information flow. The cross network is composed of cross layers, with each layer addressing the following formulation:

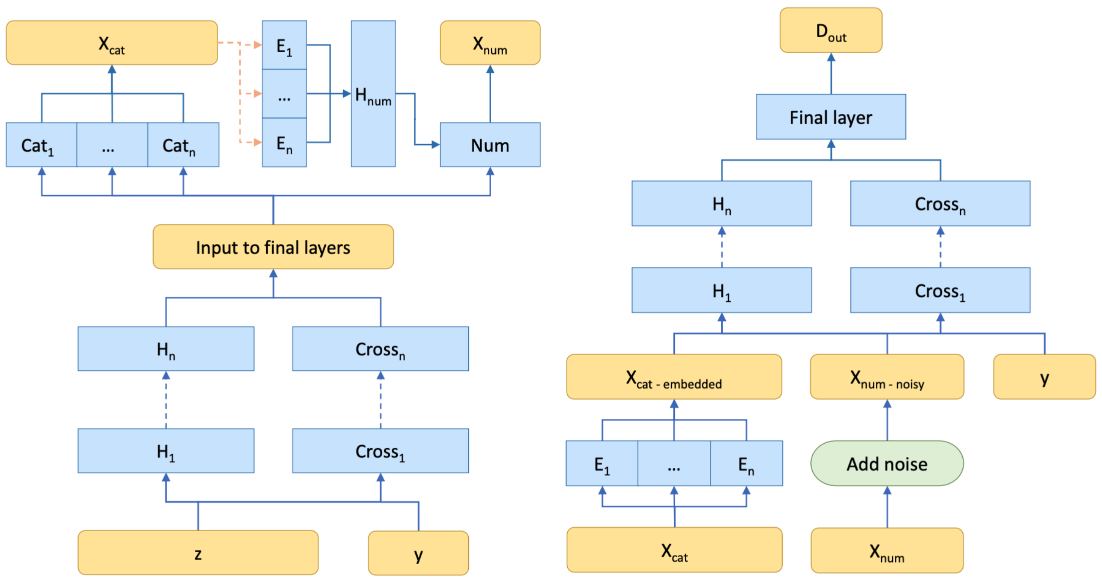
$$x_{l+1} = x_0 x_l^T w_l + b_l + x_l = f(x_l, w_l b_l) + x_l$$

Where:

- $x_l, x_{l+1} \in \mathbb{R}^d$ are column vectors denoting the outputs from the l-th and $l+1$-th cross layers.
- $w_l, b_l \in \mathbb{R}^d$ are the weight and bias parameters of the l-th layer.

Due to the limited model's capacity provided by the small parameters number of the Cross Networks, to capture high non-linear interactions and improve the representational capacity of the model, all the networks are composed of Cross Layers in parallel with a fully connected feed-forward neural network. Enforcing the Generator's noise variation while improving the discriminative power of both AC and Critic.

Network Architecture:



(a) Generator **(b)** Discriminator

Figure 3.12: Generator (left) and Discriminator (right) structures [37]

Described the improvements provided by the *c-WGANGP*, the general architecture provides the following network structures:

- **Discriminator:** Received a sample of data provided with the corresponding class labels, the Discriminator adds gaussian noise to the numerical columns while embedding the encoded categorical features. The embedding vector along with the dense numerical feature and the class label are stacked into one vector and passed through both hidden layers, provided of layer normalization, and crosslayers in parallel. Both outputs are passed through a final layer providing a final scalar value $Dout$ representing a score of data reality.

- **Auxiliary Classifier:** Similar Discriminator's structure but with some modifications:
  - No gaussian noise is added to numerical features.
  - AC not conditioned on the class labels.
  - Last layer present a Sigmoid Activation function for estimating the given data class probability.
- **Generator:** Received as input a sample of latent noise z and the minority class label, both passed through hidden layers and cross layers. The output of both networks are stacked becoming the input of the final output layers. Each categorical output layer Cat_i outputs a vector of length k_i equal to the number of levels of the i-th categorical column, while the numerical output layer outputs a vector lenght equal to the number of numerical columns. The categorical outputs X_{cat} are embedded and passed through a further hidden layer H_{num} reducing the embeddings of all the categorical columns to a single 16-dimension dense vector and furtherly concatenated to the input of the numerical output layer allowing for numerical feature conditioning.

Chapter 4

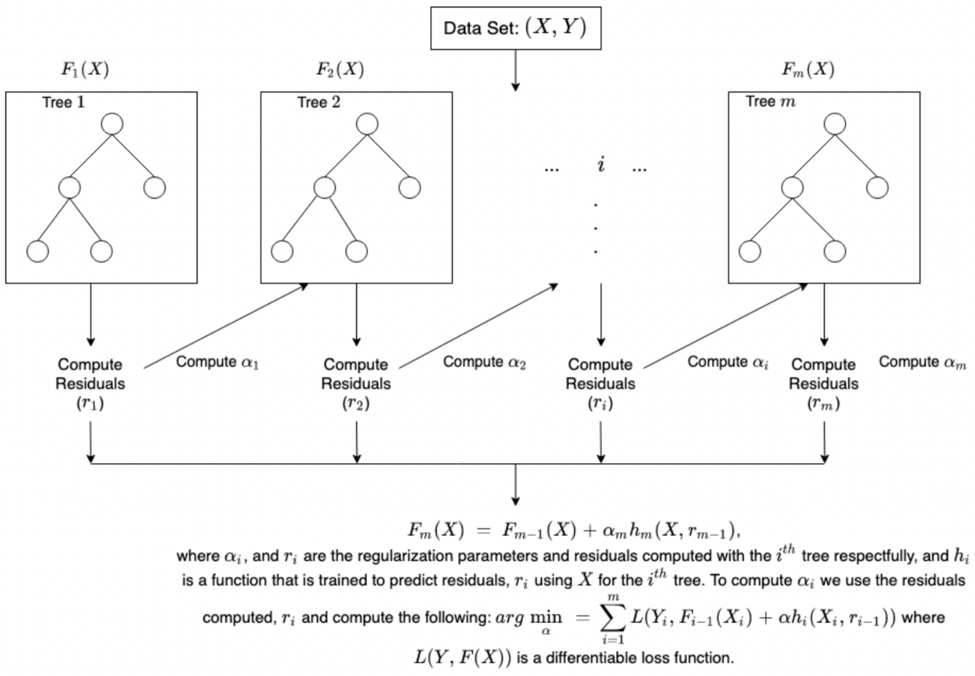
Boosting for Tabular Data

In recent years, within the field of machine learning, boosting algorithms have gained prominence as state-of-the-art models for efficiently handling tabular data. Several studies have compared the performances of Deep Learning approaches to Boosting frameworks across various tasks, dataset sizes [45],[46] and solutions proposed in Kaggle competitions [47], and according to them it has been observed that gradient-boosting algorithms, particularly XGBoost, consistently outperform deep learning techniques, delivering superior performances in these scenarios.

4.1 Boosting: In a Nuthsell

Boosting is an ensemble learning method that involves creating subsequent classifiers that are increasingly focused on instances misclassified by the previously generated classifiers. The fundamental concept behind boosting is to correct prediction errors. The models are fit and added to the ensemble sequentially in such a way that the next model aims to correct the predictions of the previous model, sequentially refining the ensemble.

Boosting often utilizes very simple decision trees that make only a single or a few decisions, often referred to as stumps. The predictions of these weak learners are combined using simple voting or averaging methods, although their contributions are weighted proportionally to their performance or capability. The objective is to construct a *Strong Learner* by leveraging the collective knowledge of many *Weak Learners*, resulting in an ensemble model that demonstrates improved predictive power and generalization. Typically, the training dataset is left unchanged, while the learning algorithm is adjusted to give more or less attention to specific examples based on their correct or incorrect predictions by previously added ensemble members. The concept of combining weak learners into strong learners was initially proposed theoretically, but early algorithms showed limited success. However, with the development of the Adaptive Boosting algorithm (**AdaBoost**) [48], boosting was demonstrated as an effective ensemble method. Since the introduction of AdaBoost, numerous boosting methods have been developed, among which Stochastic Gradient Boosting stands out as one of the most effective techniques for classification and regression tasks on structured data.



$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

Figure 4.1: Boosting Paradigm [49]

4.2 XGBoost

Extreme Gradient Boosting (XGBoost) is a scalable stochastic gradient boosting algorithm for regression and classification tasks, introduced by Tianqi Chen [50]. It is an ensemble learning method that uses gradient-boosting concepts to combine the predictions of multiple weak learners (usually classification and regression decision trees [51]) into a strong learner.

XGBoost is based on solid statistical models and a scalable learning system that can capture complex data dependencies, making it effective for a wide range of applications. For instance, it has produced state-of-the-art results on many classification benchmarks and real-world challenges involving structured data, such as store sales prediction, high customer behaviour prediction, and product categorization [45], [47].

One of the most important factors behind the success of XGBoost is its scalability. The system runs more than ten times faster than other popular solutions on a single machine and can scale to billions of examples in distributed or memory-limited settings. This scalability is due to several optimizations applied on both the algorithm and hardware sides.

Algorithm Optimizations:

- **Regularized Learning Objective**: Prevent Overfitting enhancing model's generalization capability.
- **Tree learning algorithm for handling sparse data**: During the Tree building process, missing values are handled.
- **Approximated Learning**: Improved algorithm computational efficiency by finding optimal points during the tree construction process.

Hardware Optimizations:

- **Parallel and Distributed computing**: Make advantage of multiple CPU cores to speed up the training process.
- **Cache Aware Access and Out-of-core computations**: Optimized access to processing and memory units.

All these improvements are combined into an end-to-end system able to scale into big data settings with the least amount of cluster resources.

4.2.1 Regularized Learning Objective

Considering a training dataset $\{X, Y\}$, with:

- n: Training Examples
- m: Number of Feature

Such that: $D = \{x_i, y_i\}\,(|D| = n, x_i \in \mathbb{R}^m, y_i \in \mathbb{R})$.

To predict the response variable, the tree ensemble model uses K additive functions:

$$\hat{y_i} = \sum_{k=1}^{K} f_k(x_i)\ , f_k \in \mathcal{F} = \left\{ f(x) = w_{q(x)} \right\} (q : \mathbb{R}^m \to \mathrm{T}, w \in \mathbb{R}^T) \tag{4.1}$$

With $\mathcal{F}$ being the space of regression trees, with each tree q mapping an example to the corresponding leaf index T. Each f_k corresponds to an independent tree structure q with leaf weights w, where the single component w_i represents the continuous score relative to the i-th leaf.

Therefore, given a new example, the decision rules learned by each tree q, are used to classify it. The final prediction is obtained as the sum of the scores produced by the corresponding leaves of each tree.

In order to achieve the set of functions learnt by the XGBoost model is necessary to minimize the following regularized loss function:

$$L(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \tag{4.2}$$

$$\text{where } \Omega(f) = \gamma \mathrm{T} + \frac{1}{2}\lambda \left\| w \right\|^2$$

with:

- $l(y_i, \hat{y}_i)$: Differentiable and Convex Loss function measuring the difference between the target y_i and the prediction $\hat{y}_i$
- Ω: Regularization term, penalizing each single tree complexity, achieved through the smoothing of leaf weights values.

The achieved final model consists of a combination of simple predictive functions in order to avoid overfitting. Moreover, by setting the regularization parameter to zero, the objective function corresponds to the one provided in the Gradient Tree Boosting learning.

4.2.2 Gradient Tree Boosting

In order to optimize the regularized loss reduction, the model is trained in an additive manner.

Letting ${\hat{y}_i}^{(t)}$ being the prediction of the i-th instance at the t-th iteration, the final prediction will be generated by greedily adding the f_t able to minimize at each iteration the following objective:

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}^{(t-1)}) + f_t(x_i) + \Omega(f_t) \tag{4.3}$$

$$\implies L^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{4.4}$$

Where, $g_i = \partial_{\hat{y_i}^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)})$ and $h_i = \partial^2_{\hat{y_i}^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)})$ are respectively first and second order gradient statistics on the loss function by applying the *Friedman* second-order approximation [52] on (4.3).

Defined as $\mathrm{I}_j = \{i|q(x_i) = j\}$ the instance set available at the corresponding leaf j. By applying, the proposed approximation, the optimal leaf weight w_j of leaf j in a fixed tree $q(x)$ is:

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda} \tag{4.5}$$

And assuming I_L and I_R respectively the instances of the left and right nodes after the split, such that $I_L + I_R = I$, then gain amount corresponding to the splitting, therefore, the loss reduction achieved through the split is:

$$GAIN_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] \tag{4.6}$$

Where, given a split in right and left nodes:

- $\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i+\lambda}$ is the impurity measure for the right node.
- $\frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i+\lambda}$ is the impurity measure of the left node.
- $\frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i+\lambda}$ is the impurity measure of the root node.

Moreover, each tree presents an intrinsic pruning strategy possible to be applied during or after the tree's construction and prevent the branches' full development. Based on a user pre-defined regularization parameter γ, the split is evaluated:

- If $GAIN_{split} \leq \gamma \rightarrow$ Prune the Branch
- If $GAIN_{split} \geq \gamma \rightarrow$ do not Prune the Branch

4.2.3 Shrinkage and Columns Subsampling

Besides the regularized objective, to further prevent the overfitting behaviour of the algorithm, two additional techniques are used:

- Shrinkage [53]: Newly added weights are scaled by a factor α after each step of tree boosting. Similar to learning rates applied in stochastic optimization, the influence of each individual tree is reduced, leaving space for future trees to further improve the model's prediction.

$$\hat{y_i} = \alpha \sum_{k=1}^{K} f_k(x_i)\ , f_k \in \mathcal{F} \tag{4.7}$$

- Column Subsampling/Feature Bagging [54]: Already exploited in Random Forest models, the method consists of randomly selecting a subset of columns from the set of columns chosen for the current tree, in order to prevent over-fitting and speed up parallel algorithm computations. The random sampling of feature subsets promotes the training of decorrelated trees, giving the possibility to different features to be selected as best attributes for the split.

4.2.4 Split Algorithms

Exact Greedy Algorithm:

The exact Greedy algorithm consists of enumerating all the possible splits according to the data distribution and finding at each step the one able to maximize the split gain over all the features. To do so, the algorithm must first sort the data according to the feature values and visit the database in sorted order to accumulate the gradient statistics for evaluating the split gain.

Since it is computationally demanding to enumerate all the possible splits when dealing with continuous features, the technique is strongly discouraged in those settings where the data do not fit entirely into memory.

Approximated Algorithm:

The approximate framework is divided into three categories:

- **Histogram-based Approximate Greedy Algorithm** [55]: During the training phase, for each feature, the XGBoost constructs a histogram representing the data points distribution for that feature. The algorithm bins the feature values into discrete bins, each having a similar number of points inside. For each bin boundary, gradient statistics are computed to evaluate the potential splits. The bin boundary that results in the highest gain is selected as the best split for that feature.

- **Quantile Sketch**: During the training phase, for each continuous feature quantiles are used as an approximated summary of the data distributions. Inside each quantile gradient statistics are computed, and the quantile bin that results in the highest gain is selected as the best split for that feature.

- **Weighted Quantile Sketch**: Similar to quantile sketch, feature values are bucket in quantiles, but in this case containing a similar amount of sum of squared gradients, evaluated for each data point as a weight. The newly defined quantiles are evaluated as potential candidate split and the best-split point able to greedily maximize the split gain is selected.

Moreover, for all the approximated algorithms the splitting proposal can be evaluated in two modalities:

- **Global Variant**: Propose the candidate split during the initial phase of tree construction and uses the same proposals for split findings at all levels.
- **Local Variant**: At each split the approximated algorithm is computed and a new proposal split is evaluated at each new level of the tree.

4.2.5 Sparsity-aware Split Finding

The model is able to handle sparse datasets by identifying a default direction for those instances presenting missing values. During the tree construction, when a candidate split feature contains missing values for some data points, for each candidate threshold the data points are assigned to both left and right child nodes and the gain for both possibilities is computed. The default direction is chosen as the one able to bring a major decrease in the loss function.

4.2.6 Parallel Learning

The most time-consuming part of tree learning is to get the data into sorted order. To reduce the cost of sorting, data are stored in a compressed format, called block (suggested 2^{16} examples per block to balance both cache and parallelization). This technique allows XGBoost to handle datasets that are too large to be loaded entirely in memory and distribute the workload across multiple CPU cores, exploiting both Feature-Level and Thread-Level Parallelism.

- **Feature-Level Parallelism**: Different dataset features are processed parallelly and independently by the different computational units available, takings advantage of the multi-core CPUs and speeding up the tree construction process.
- **Thread-Level Parallelism**: When evaluating potential split points for a single feature. XGBoost can divide the work among multiple threads running on the same core. Each single thread processes each block independently, and results are aggregated to construct the final decision tree model.

Chapter 5

Performance Evaluation

As already introduced, being able to choose the proper evaluation metric is crucial, especially when dealing with machine learning models applied to imbalanced datasets. In this scenario, proper identification of the right set of evaluation metrics helps to ensure that the model's performance is accurately assessed and avoids misleading conclusions.

Since the presented study is a binary classification problem, by mapping the minority and majority instances respectively to positive and negative outcomes, two-class metrics are evaluated. Indicating as TP and TN the number of positive and negative examples that are correctly classified, while FN and FP respectively the number of misclassified positive and negative examples, according to a taxonomy proposed by *Cesar Ferri* [56] is possible to divide the evaluation metrics into three groups:

- **Threshold Metrics**
- **Ranking Metrics**
- **Probability Metrics**

5.1 Threshold Metrics

Those family of metrics quantify the classification prediction error:

- **Accuracy**: Measures the percentage of correct predictions throughout the whole dataset.

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

- **Sensitivity/True Positive Rate/Recall:**

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

- **Specificity/ True Negative Rate:**

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

- **Precision:**

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

- **F$_\beta$-Score:**

$$F_\beta - score = (1 + \beta^2)\frac{Precision \times Recall}{(\beta^2 \times Precision) + Recall}$$

5.2 Ranking Metrics

Those family of metrics evaluates classifiers based on how effectively they can separate classes, without making assumptions on the class distributions, computing diagnostic plots to test the model's performance.

For design purposes, the classifier is required to predict for the different instances a class probability membership to which different threshold metrics are evaluated, identifying as good class separators those models able to maintain a good score across a range of thresholds. Typically, overall ranking information is evaluated considering the created plot's Area Under the Curve (AUC).

The literature presents different ranking metrics defined by evaluating different threshold metrics for different probability class levels, however, the most used are:

- **AUC-ROC:** Evaluates the area under the receiving operator characteristic curve, obtained by calculating the FPR and the TPR for a set of predictions by the model under different thresholds. Each threshold is a point on the plot and the points are connected to form a curve expressing the overall performances. A classifier that has no skill (e.g. always predicts the majority class under all thresholds) will be represented by a diagonal line from the bottom left to the top right. The AUC-ROC is always in the range [0,1], with an unskilled classifier having a score of 0.5.

- **AUC-PR:** Represents a summary of the precision-recall curve providing an estimate for all the evaluated thresholds a trade-off between precision and recall. Also, in this case, the generated curve expresses the overall performances of the classifier, with an unskilled classifier identified by a horizontal line, proportional to the number of positive examples in the dataset.

5.3 Probability Metrics

Those family of metrics quantify the classifier reliability based on the uncertainty the model has in predictions, penalizing those mistakes with high confidence.

Evaluating a model based on the estimated probabilities requires a calibration assessment of the classifier. While some classifiers are trained using a probabilistic framework (e.g. Logistic Regression and Naïve Bayes), many nonlinear classifiers require their probabilities to be calibrated via probabilistic metric.

The most common metrics for evaluating predicted probabilities are:

- **Binary Cross-Entropy:** Summarize the average difference between the empirical and actual probability distribution

$$\text{Cross-Entropy} = -\frac{1}{N}\sum_{i=1}^{N} y_i \times \log(\hat{y}_i) + (1 - y_i) \times \log(1 - \hat{y}_i)$$

- **Brier Score:** Mean squared error between the expected probabilities and the model's predicted probabilities of belonging to the positive class.

$$\text{Brier-Score} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

  where N is the number of observations, y_i is the true class label of the i-th instance and $\hat{y}_i$ is the predicted probability of belonging to the positive class for the i-th instance.

Chapter 6

Dataset EDA

6.1 Dataset Description

The dataset exploited for the pipeline development is the "Credit Card Customer Attrition” [57], collected by the platform Leaps by Analyttica and containing different information relative to the banking customers. Those features on which the identification of the possibility of customer churning has been built describe each instance highlighting two important customer characteristics shared with the credit company touching both demographical and financial domains.

The former aspect is identified by all those attributes falling in the category of sensitive information or personally identifiable information (*PII*), which refers to data that, if disclosed, can lead to an individual’s identification and possible discriminatory behaviour if not well managed during the model’s training phase. Those features describe a customer by its age, gender, family size, marital status, and education level. While the latter refers to all the remaining details involved in the individual’s financial account; for example credit card category, customer’s revolving balance and open-to-buy credit line.

The analysed dataset collects the information of 10,127 customers, without the presence of missing values, and of which 1627 of the instances are identified as churned customers. Each customer instance is represented by 21 features, summarized in the following Table 6.1:

Feature	**Description**
CLIENTNUM	Customer ID
Attrition_Flag	- 0: The customer did not churn - 1: The customer did churn
Customer_Age	Customer age
Gender	- M: Male - F: Female
Dependent_count	Customer's family size
Education_Level	Customer's educational level
Marital_Status	Customer's marital status
Income_Category	Customer's income category
Card_Category	Customer's card type
Months_on_book	Number of months on the bank's books in the preceding 12 months
Total_Relationship_Count	Total accounts the customer has with the bank
Months_Inactive_12_mon	Number of months of credit card inactivity
Contact_Count_12_mon	Amount of times a customer has contacted the bank in the preceding 12 months
Credit_Limit	Customer's credit limit on the card
Total_Revolving_Bal	Customer's total revolving balance
Avg_Open_To_Buy	Customer's open-to-buy credit line averaged on the last 12 months
Total_Amt_Chng_Q4_Q1	Change in transaction amount (Q4 over Q1)
Total_Trans_Amt	Total amount of transactions in the preceding 12 months
Total_Trans_Ct	Total count of transactions in the preceding 12 months
Total_Ct_Chng_Q4_Q1	Change in transaction count (Q4 over Q1)
Avg_Utilization_Ratio	Average Utilization Ratio

Table 6.1: "Credit Card Customer Attrition" Dataset Description

6.2 Funtional Exploratory Data Analysis:

The heterogeneous characteristic of the input dataset requires to evaluate properly the different feature distributions based on their respective data typology.

While reporting the different univariate feature distributions in **Appendix A**, the following exploratory data analysis focused on highlighting which features could be prematurely predicted as the main driver of our final classification task, based on how well they individually distinguish between attrited or existing customers.

By analysing the feature distributions with respect to the target variable (Attrition_Flag), it is possible to observe how those attributes able to differentiate the customer's churn possibility are the ones involved mainly in the financial domain (Fig 6.1)

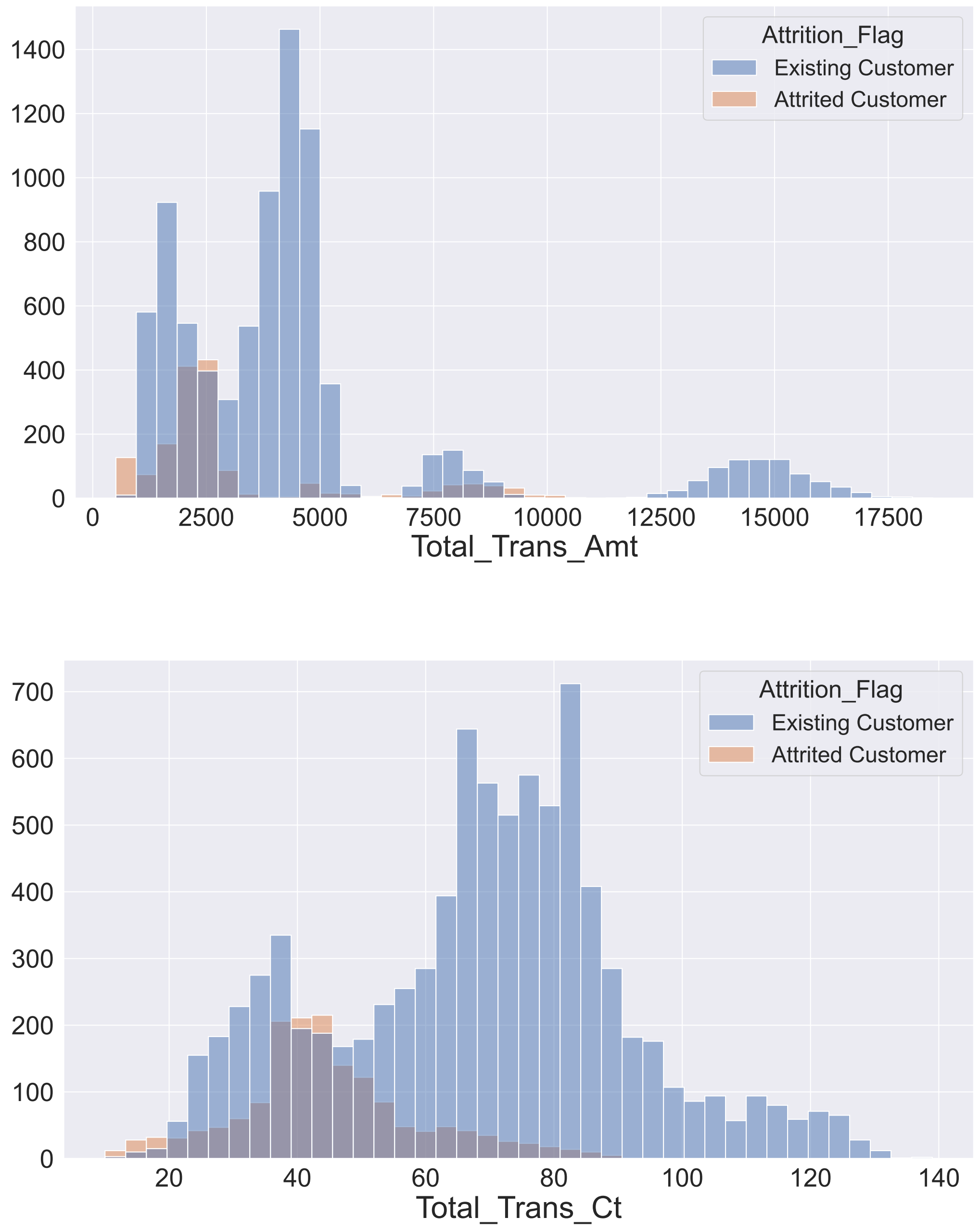


Figure 6.1: Histogram of the features Total_Trans_Amt, Total_Trans_Ct based on the response variable Attrition_Flag

As the images suggest, due to the strong class imbalance present in our dataset, most continuous feature value ranges are able to identify a customer as maintaining its credit card (existing customer). While, due to dataset class imbalance, just a few ranges express an expected reduction in customer engagement. Moreover, by analysing the different attributes' distributions it is possible to state how the ability to discern customer behaviour appears relevant in those features which characterised the individual credit card adoption exploiting different metrics such as total/variational amount/count of transactions.

6.3 Correlation Analysis:

In order to uncover relationships and patterns between the numerical variables in the dataset, and inspecting possible data multicollinearity, both linear and monotonic pairwise correspondences are exploited through Pearson and Spearman's correlation coefficients (Fig 6.2 and Fig 6.3).

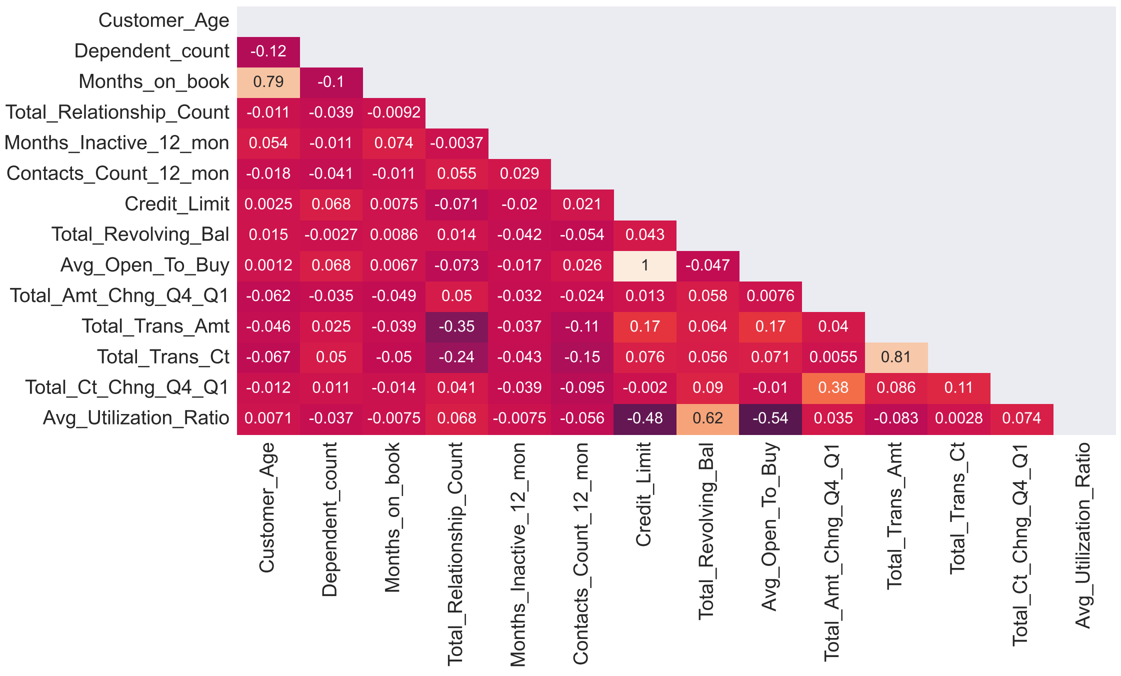


Figure 6.2: Pearson Correlation Heatmap

The obtained correlation's heatmaps highlight strong positive relationships between many numerical features. Especially, they suggest how practically both Avg_Open_To_Buy and Credit_Limit exploit the same information, whereas a monotonic relationship is found between Total_Trans_Amt and Total_Trans_Ct, which could be expected by the strong proportionality expressed by the fact that an increase in the number of transactions often leads to an improvement of the economical amount spent by the customer through its credit card. In addition, also a quite strong linearity is presented between the features Month_on_book and Customer_Age, also in this case mainly intuitively their relationship could be explained by the fact the older customer may posses their credit card from along time and thank to their accumulated experience and positive interaction with the bank they provide a longer credit card history, while younger people just due to their inner inclination to switch services they might have open their bank account recently.

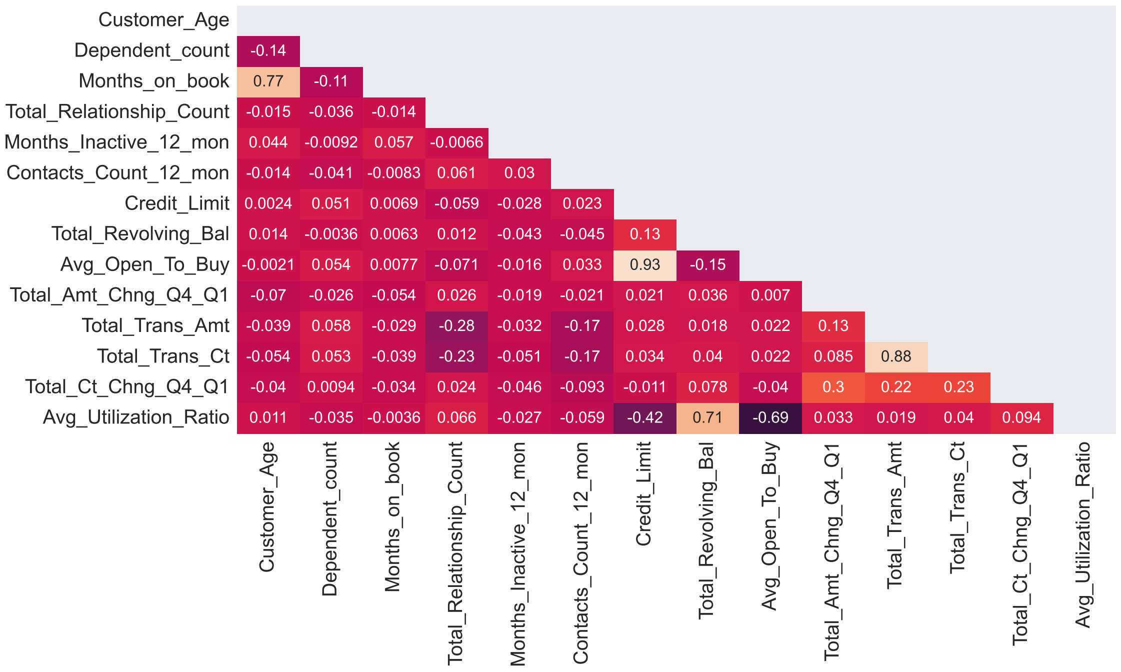


Figure 6.3: Spearman Correlation Heatmap

The exploratory data analysis (EDA) is an essential and indispensable step to get some domain-specific knowledge and develop hyphothesis through the dataset. Through the following analysis has been possible to obtain some valuable insights into the relationships between various features describing customers. Moreover, the reasoning effort behind the discussed major data distinction, related to fairness compliance, and they way in which different features are related will be helpful in the design and generation of the following machine-learning pipeline.

Chapter 7

Pipeline Design:

Being able to properly access churning behaviour becomes fundamental in the dynamic landscape of financial services. The ability to predict which customers are at risk of churning, in the realm of credit card services, allows businesses to take proactive measures towards smoothing such phenomenon. To give a specific tool to access credit customer churn, in this chapter, we try to leverage the discussed theoretical backgrounds and improve them with the creation of a robust machine-learning pipeline.

This process employs a series of specifically designed steps, each contributing to the development of an efficient data-driven predictive model.

7.1 Data Ingestion and Pre-processing: IBM DataStage and Feature Selection

In order to properly develop our machine learning pipeline, it has been necessary to find a way to collect, import and prepare the input data from the data sources to be used as input for training, validation and testing of machine learning models. To access the initial stage of the pipeline it has been exploited the technology provided by IBM DataStage, an industry-leading data integration tool that helps in designing, developing and running jobs that move and transform data, by supporting both ETL and ELT patterns.

7.1.1 DataStage Environment:

Despite the on-premises deployment, DataStage is also integrated into the Cloud Pak for Data as a Service (**CPDaaS**), a fully managed cloud and data fabric solution offered by the IBM Cloud Pak for Data Platform; a modular platform running integration data and AI services in a hosted environment containing different microservices running on a multi-node Red Hat OpenShift cluster.

CPDaaS provides all the capabilities and features of the on-premises version of IBM Cloud Pak for Data, but in a cloud-native and fully hosted environment, without the need for infrastructure provisioning, maintenance, and management. As a matter of fact, IBM takes care of the underlying infrastructure, ensuring scalability, availability, and security, while leaving the users to focus on utilizing the platform for their data and analytics needs.

DataStage on Saas provides a graphical framework for developing the jobs that move data from source systems to target systems. The transformed data can be delivered to data warehouses, data marts, operational data stores, real-time web services and messaging systems. Moreover, DataStage exploits parallel processing, the parallel engine (PX), to run jobs on multiple nodes in the IBM Cloud and uses enterprise connectivity to provide a fully scalable platform.

The DataStage graphical interface for designing and building data integration and ETL process workflows is the DataStage Flow. The DataStage flow consists of a set of interconnected stages, each representing a specific action or transformation to be performed on the data. The basic building blocks of the flow are:

- **Jobs**: The overall data integration workflow, consisting of one or more stages connected to define the data flow and transformations.
- **Stages**: Divided into Stages and QualityStage, they are specific operating modules used during the DataStage job regarding functions like extraction, transformation, filtering, aggregation or conflict and ambiguities resolutions and data standardization. Moreover, they also supported customizable stage development.
- **Connectors**: Allow to connect DataStage with different data sources and targets, facilitating reading data from and writing data to various external data sources (both structured and dynamic), providing data connectivity and metadata integration, or files (Lookup file set, Flat file, Data set, File set or Sequential File).
- **Transformations**: Enable to apply transformations to the data within the data flow, including cleansing, mapping, aggregating, and enrichment.
- **Control Element**: Manage the execution of the job within the data flows, providing conditional statements, loops, branching and error-handling mechanisms.

- **Parallel Processing**: The parallel processing supports, allowing the job's execution to be distributed across multiple processing units. DataStage offers six PX environments with prefixed computing and memory capacity (default 1cCPU and 4GB RAM) to choose to run your ETL job based on dataset size and number of stages involved.
- **Partitioning Stage**: Data are divided into smaller partitions allowing for data-parallel processing and distributing the workload across multiple processing resources to achieve better performance and scalability.
- **Collection Stage**: Reverse of the partitioning process, which involves consolidating data from multiple partitions into a single dataset, typically performed after a parallel processing operation to merge the results from different partitions into a final output or for further processing.

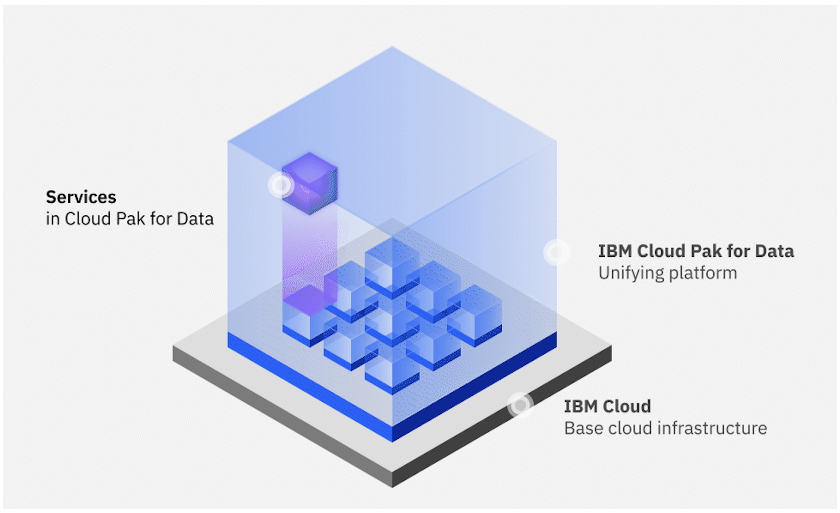


Figure 7.1: IBM Cloud Pak for Data Platform: IBM DataStage service for data ingestion [58]

7.1.2 Data Flow Design:

The original dataset is split into training and test sets, following a structured 80:20 ratio. The stratified split ensures that both subsets maintain the same distribution of classes of the original dataset, minimizing any bias in the training and evaluation process.

Through DataStage has been possible to design a data flow (Fig 7.2) able to shepherd the data through a series of crucial transformations, ultimately enhancing the model's predictive capabilities while prioritizing ethical considerations.

The designed flow collects the inputs and processed data into the IBM-optimized cloud storage solution, IBM Cloud Object Storage. Being highly scalable and distributed across multiple geographic locations, the storage architecture provides global accessibility and strong security measures like, encryption at rest and in transit and IAM (Identity and Access Management) services making it optimized for storing and retrieving data objects over the internet.

The initial phase of this flow revolves around a preliminary feature selection, focused on eliminating variables which has been shown carrying on the same information and those demographical features that could potentially lead the machine learning model to make discriminatory decisions , ensuring the model's fairness and equitable outcomes.

The subsequent phase delves into encoding and feature scaling, essential steps that enable the data to seamlessly integrate with both the generative architecture and classification model. The categorical feature ordinal encoding and target variable binary encoding are harnessed to transform the categorical attributes into a format that the model can comprehend and process effectively. The former exploits the inherent hierarchy of the different data levels of the categorical variables, while the latter labels the minority class as 1 ad the majority class as 0, to drive attention to the identification of the churning event.

Furthermore, min-max scaling ensures that each feature is compliant with the way in which the proposed GAN manages numerical data while mitigating the risk of biased outcomes during the classification model's training.

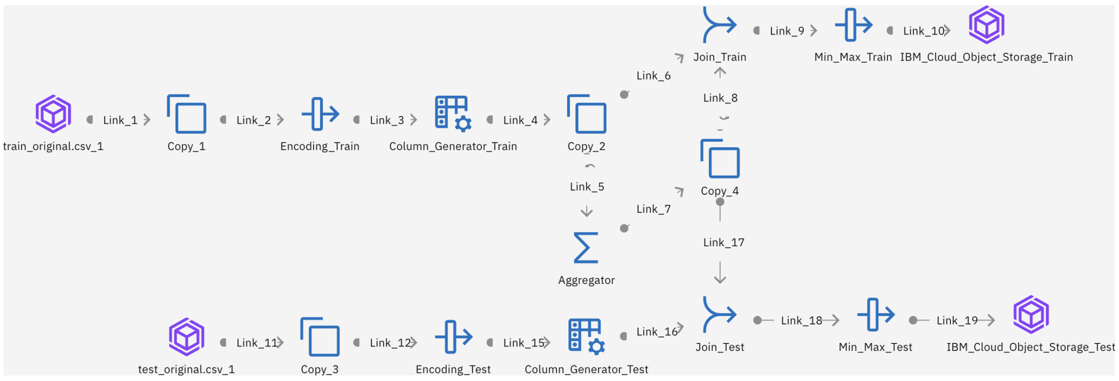


Figure 7.2: ETL DataStage Flow

To provide proper guidance on comprehension of the designed DataStage flow the following table summarizes the different stages and their relative functionality during the ETL process:

Stage	Function
Copy	
Copy_1	Generate a copy of the Train set, not considering the features: Avg_Open_To_Buy, Education_Level, Marital_Status, Gender
Copy_3	Generate a copy of the Test set, not considering the features: Avg_Open_To_Buy, Education_Level, Marital_Status, Gender
Copy_2	Generate a copy of the Train set passed to both Aggregator and Join stages
Copy_4	Generate a copy of the min-max statistics of the Train set to both Train and Test sets
Column Generator	
Column_Generator_Train	Add a column of 1s to the Train set
Column_Generator_Test	Add a column of 1s to the Test set
Transformer	
Encoding_Train	Ordinal encoding for categorical features and Binary encoding on the response variable of the Train set
Encoding_Test	Ordinal encoding for categorical features and Binary encoding on the response variable of the Test set
Min_Max_Train	Compute the min-max normalization on Train set features
Min_Max_Test	Compute the min-max normalization on Test set features
Aggregator	
Aggregator	Compute min-max summary functions for each Training set feature
Join	
Join_Train	Join the min-max statistics to the Train set
Join_Test	Join the min-max statistics to the Test set

Table 7.1: Stages functionality

7.1.3 Feature Selection:

After retrieving the transformed data from the developed ETL process, the next crucial step involves performing feature selection to extract the most salient features that aptly describe our dataset's characteristics. To accomplish this, two distinct yet complementary wrapping feature selection methodologies were employed: the sequential floating forward (SFFS) and floating backward (SFBS) feature selection techniques. The primary objective of the adoption of such methodology was to discern and retain the most impactful attributes, being sure of their contribution to the model's decision-making process.

Both methodologies, were employed to explore the dataset's dimensions comprehensively. These techniques iteratively find the best subset of features by considering how each individual feature affects the model's performance. Their key difference lies in the starting points and the direction of iterations. SFFS starts with a small set of features and adds features iteratively, while SFBS starts with all features and removes them iteratively. These methods strike a balance between exhaustively exploring all possible feature combinations (which is computationally expensive given the feature set size) and quickly reaching an acceptable subset of features.

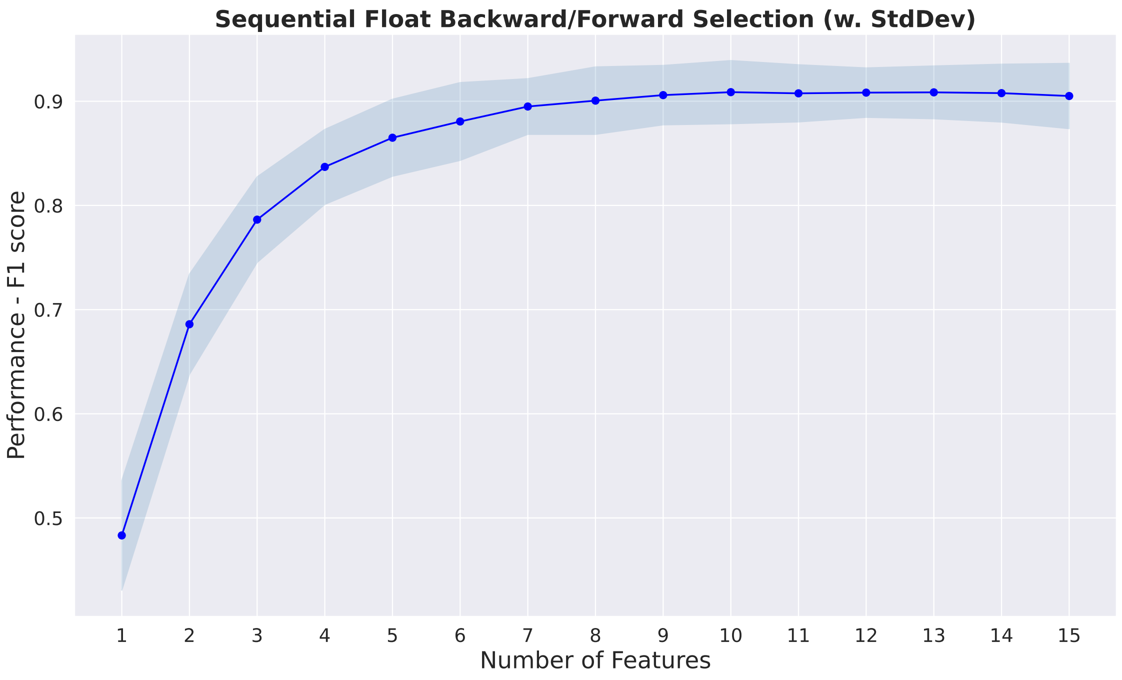


Figure 7.3: SFBS/SBBS results

The feature selection approaches converge to the same results. Therefore, Figure 7.3 represents just the sequential backward feature selection. The F1-score performance evolution of a default XGBoost classifier configuration showcases a commendable consistency in identifying the optimal features set. However, there is a relatively marginal performance variation when using a number of features over the plot's knee points. According to this, and since the default classifier configuration performance presents variability related to the feature order, leveraging both the obtained results and domain knowledge, a customized feature selection is exploited. This personalized variable section contributes as a contextual layer to the model's decision process, aligning the model's data interpretation more cohesively with real-world implications.

7.2 Data Oversampling

Hyperparameter	**Value**
Optimizer	Adam($\alpha = 5 \times 10^4, \beta_1 = 0, \beta_2 = 0.9$)
Epochs	300
Batch size	50
Gradient Penalty factor λ_{GP}	15
AC loss factor λ_{AC}	10% of generator Wasserstein loss on current batch
AC loss cut-off	0.3
Discriminator updates per generator update	3
Generator layer sizes	(128,64)
Number Generator crosslayers	1
Discriminator Layer sizes	(128,64,32)
Number of Discriminator crosslayers	2
AC layer sizes	(64,64)
Number of AC crosslayers	2
Activation function in hidden layers	Leaky ReLU
Numerical Activation	None
Noise Distribution	$\mathcal{U}^{30}$ [0,1]

Table 7.2: cWGAN hyperparameter setting

Based on the promising outcomes in the realm of strong non-linearity datasets of enhancing classification performances and accurately estimating complex data distribution achieved by the joint adoption of c-WGANGP architecture with gradient-boosting techniques [cWGAN], we have decided to exploit such generative methods in our pipeline.

Incorporating the cWGANGP architecture within the machine learning pipeline introduces a multi-faceted approach to tackle the challenges of imbalanced datasets. It combines the strength of accurate data distribution modelling from cWGAN with the robustness of Gradient Boosting's ensemble learning. This integration not only boosts the model's capacity to address class imbalance but also results in improved classification performance on both majority and minority classes. Moreover, facilitating generalization by allowing the model to recognize and adapt to complex non-linear relationships present in the data.

However, despite the original work, the architecture needs to be suited to be able to work the data types of the different features involved in the dataset after the feature selection step. Since the resulting dataset presents just numerical features, the generative adversarial network architecture has been modified to adapt to such data typologies. Despite the original structure, all the network layers involved in the management of categorical variables are not exploited leaving room for the new structure presented in Fig 7.4.

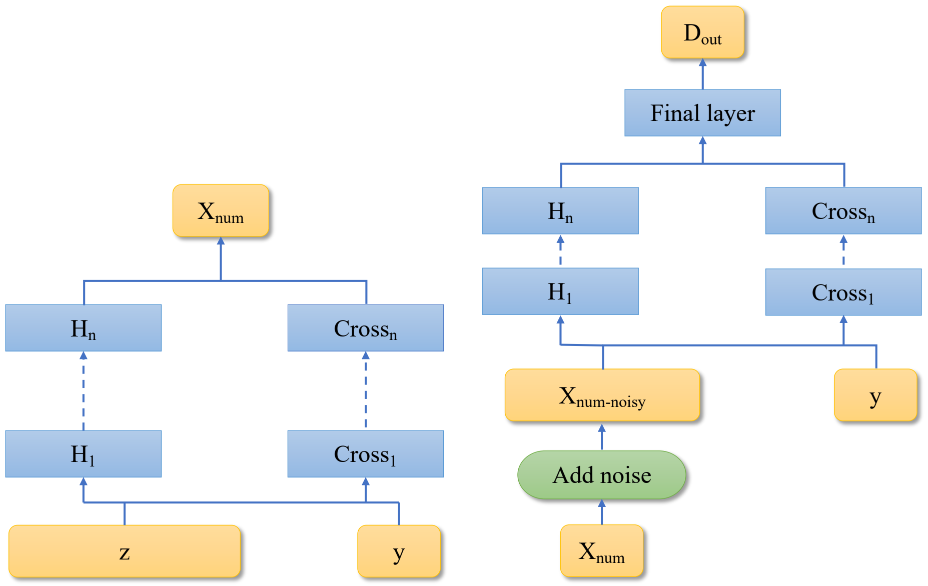


Figure 7.4: Generator (left) and Discriminator (right) structures

The cWGAN model has many hyperparameters that can be chosen. However, proper tuning for the selected structure has been guided by the recommendations laid out in the original paper and the specific context of the dataset under consideration. The suggested hyperparameter values that yield optimal performance on the UCI adult dataset have been utilized as a foundational basis for tuning the GAN within the context of the dataset interest.

Grounded by the extensive experiment and tests on the different hyperparameter combinations that brought to effectively balance the adversarial training process, stabilize convergence, and yield high-quality synthetic samples, the hyperparameters setting of Table 7.2 is obtained by leveraging the expertise and insights of the architecture's developers, while also accounting for the unique characteristics of the dataset used.

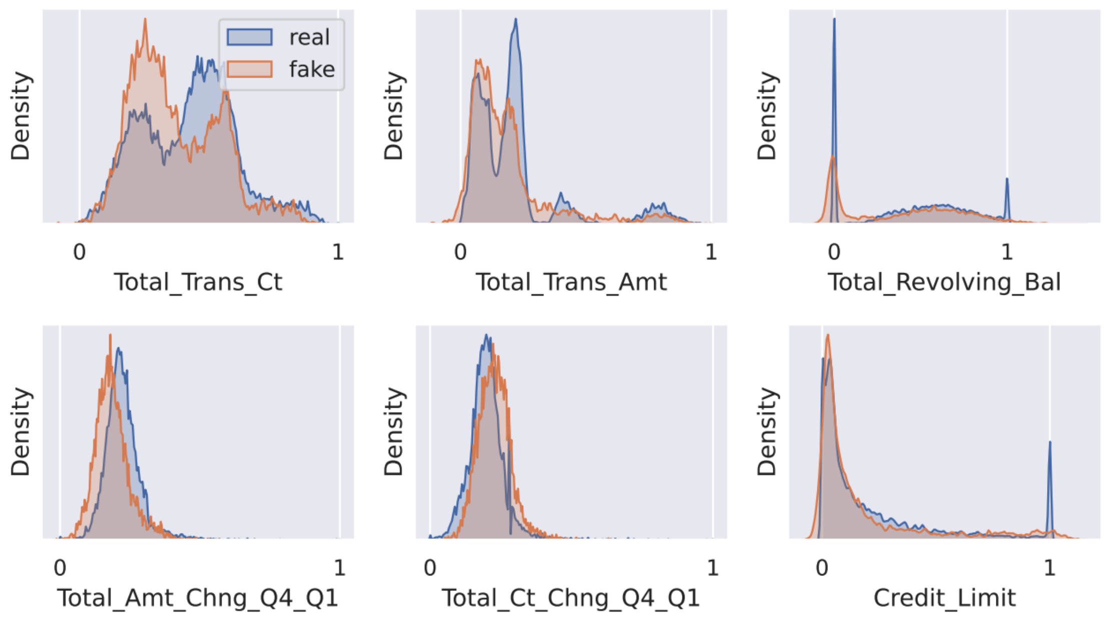


Figure 7.5: Univariate distribution plots comparing the real and generated distributions of the continuous numerical columns.

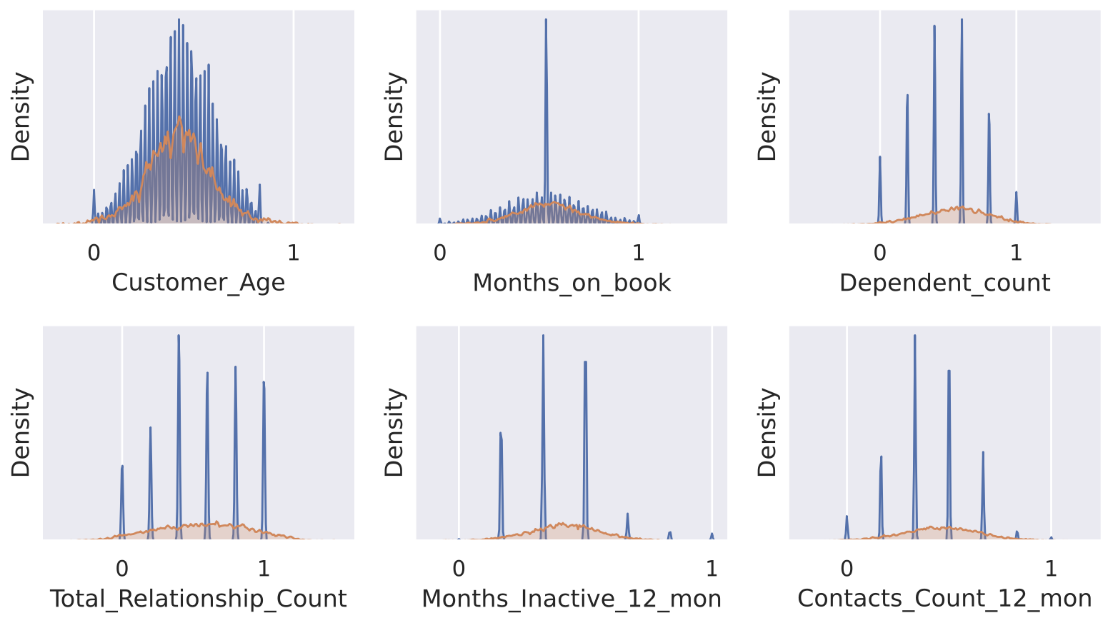


Figure 7.6: Univariate distribution plots comparing the real and generated distributions of the discrete numerical columns.

Furthermore, in order to assess the extent to which the generative architecture effectively learns to produce authentic synthetic data, we have conducted an evaluation of the model's generative performance on the input dataset.

Comparing the univariate distributions of each variable between the synthetic and real data, using a kernel density estimate (KDE) plot with Gaussian kernels and bandwidth of 0.02, some noteworthy observations have surfaced. For continuous numerical variables, the architecture generally succeeds in approximating the distribution of individual variables and demonstrates an ability to capture multiple modes within the distribution, however, encountering challenges in distinguishing between closely neighbouring modes variations (Fig 7.5). While, when it comes to discrete numerical variables, the architecture adeptly emulates distributions exhibiting high cardinality, presenting some difficulty for those quantitative discrete variables with lower cardinality (Fig 7.6). However, generating data that emulates an empirical Gaussian-shaped continuous distribution properly approximates the mean and standard deviation of these variables. To quantitatively assess the generative performance concerning univariate distributions, a scatterplot representation that reported the dimension-wise mean and standard deviation comparisons of synthetic data against real data is used (Fig 7.7 left and middle). The deviation from the ideal identity line is measured using the root mean squared error and reinforced through the Pearson correlation coefficient. Our findings indicate that generative learning closely aligns with both the means and standard deviations of the original dataset, avoiding the mode collapse phenomenon.

Moreover, the evaluation also extends to assess whether the generator adequately models the relationships between variables. We visualize a dimension-wise prediction performance (Fig 7.7, right), which determines how predictable a column is based on the remaining columns—a measure of the strength of association between variables. Ideally, this performance should be consistent between real and synthetic data. Also in this case, the presented scatterplot comparison between real and generated values is summarized by the root mean squared error and the Pearson correlation coefficient with reference to the identity line.

Despite the challenges faced in predicting performances for discrete numerical variables, the architecture effectively estimates the distribution of a complex, tabular dataset by generating synthetic data that is both realistic and coherent. This evaluation underscores the architecture's capacity to handle intricate relationships and produce synthetic data that mirrors the complexities of the original dataset.

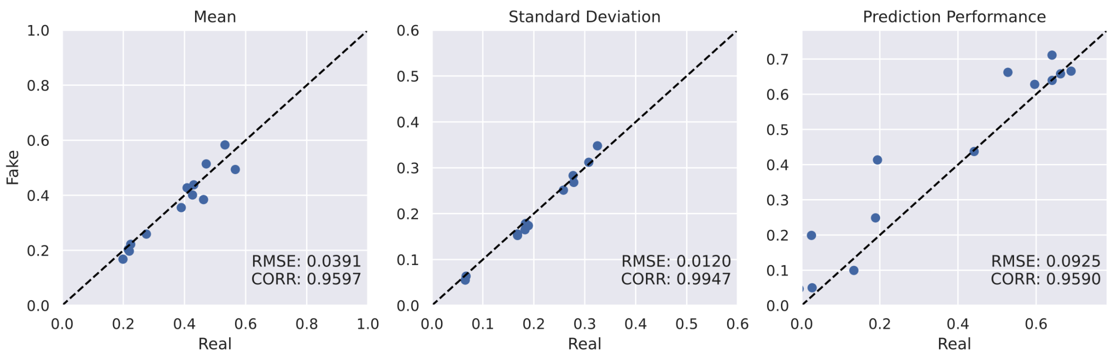


Figure 7.7: Dimension-wise performance metrics: dimension-wise means (left), standard deviations (middle), and prediction performance(right).

7.3 Model Deployment: H2O AutoML

Having highlighted XGBoost remarkable ability to handle complex data relationships and deliver outstanding predictive performance. The exploitation of its true prowess can only be fully harnessed with a proper hyperparameter optimization able to match the unique characteristics of the provided input dataset, composed of both real and synthetic tabular data.

In the landscape of machine learning, the pursuit of correct settings configuration plays a pivotal role in the achievement of optimal model performance. However, tuning hyperparameters manually is a formidable challenge that demands extensive experimentation and expertise. In the last few years, automation has emerged as a game-changing paradigm. Designed to simplify and automate the process of building and deploying high-performing machine-learning models, it has been able to make complex and time-consuming typical tasks of the ML pipeline like data preparation and ingestion, feature engineering, hyperparameter optimization and model selection accessible to data scientists and analyst with varying levels of expertise.

Following those ideas, the technology-specialized company H2O.ai provides an open-source and commercial platform composed of a suite of tools for machine learning, data analysis and predictive modelling. To automate the model's deployment workflow one of its key features is the AutoML framework [59].

Being accessible through a user-friendly interface, H2O Flow web UI, or R and Python H20 libraries, the H20 AutoML interface is designed to have as few parameters as possible, in which the user just needs to point the dataset of interest (schema structure guesses by the H20 data parser), identify the response columns and optionally specify a time constraint or limit on the number of total models to train. Its accessibility, combined with the provided cutting-edge and distributed implementation of many predictive algorithms and its automation capabilities, democratizes machine learning by allowing both newcomers and experienced practitioners to build high-quality models without being experts in every facet of the ML pipeline.

Identified the algorithm of interest or leveraging the full set of algorithms available in H20, the AutoML performs a hyperparameter search in order to deliver the best model. In the case of XGBoost, H2O implements the algorithm communicating with native XGBoost libraries via JNI API, providing all the necessary REST API definitions to expose the XGBoost model builder to the client. Based on the environment in which the H20 cluster is initialized, the AutoML framework trains and performs the model's hyperparameter grid search exploiting the provided resources.

H2O Cluster characteristics	**Values**
H20_cluster_uptime:	2 secs
H2O_cluster_version:	3.42.0.3
H20_cluster_name:	H2O_from_python_unknownUser_j2ueyl
H2O_cluster_total_nodes:	2
H2O_cluster_free_memory:	12.75 Gb
H2O_cluster_total_cores:	8
H2O_connection_url:	http://127.0.0.1:54321
Python_version:	3.10.12 final

Table 7.3: H2O Cluster characteristics

In our setting, based on the resources allocated to the Google Colab environment, the H2O cluster provides the characteristics summarized in Table 7.3, while the obtained XGBoost hyperparameter's best configuration is presented in Table 7.4.

XGBoost Parameters	**Description**	**Values**
Booster	Typology of base model used in the boosting process	gbtree
Col_sample_rate	Feature sampling rate for each split in each level	1.0
Col_sample_rate_per_tree	Features subsampling rate per tree	0.7
Max_depth	Maximum tree depth	6
Min_rows	Minimum number of observations for a leaf	1.0
Ntrees	Number of trees to build the additive model	30
Reg_alpha	Value for L1 regularization on feature weights	0.5
Reg_lambda	Value for L2 regularization on feature weights	0.01
Sample_rate	Row sampling ratio (without replacement) of the training instance	1.0

Table 7.4: XGBoost Hyperparameters

7.4 Model's Evaluation

Having examined the possible model's evaluation metric, the ability to translate the problem's assessment and estimate the classification algorithm's performance into mathematic formulation becomes of paramount importance.

Given that our endeavour revolves around providing each customer with a score that indicates the likelihood of churning, while simultaneously assessing the permissible level of error reduction that the model can exhibit for the minority class, a multifaceted approach to evaluate the model's performance becomes fundamental.

Consequently, the translation of these requisites into mathematical expressions is accomplished by opting for the following classification performance evaluation metrics the Area Under the Receiving Operator Curve (AUC-ROC), the Area Under the Precision Recall Curve (AUC-PR) and the Brier score.

While traditional threshold-based metrics such as accuracy and f1-score necessitate of a binary decision threshold, being not able to capture the complete performance spectrum of the classifier, AUC-ROC and AUC-PR provide a more holistic understanding, particularly well-suited for scenarios involving imbalanced datasets. AUC-ROC showcases the model's proficiency in distinguishing between classes across a range of thresholds, addressing the model's overall discriminative capacity. Conversely, AUC-PR accentuates the trade-off between precision and recall for the minority class. This combined evaluation approach sidesteps the need for arbitrarily selecting a threshold point, enabling an equitable comparison of overall classification performance.

Moreover, in order to gauge the suitability of the model's predicted churning probabilities as a viable scoring metric, a calibration assessment of the model becomes necessary. Poor calibration can lead to misleading probability estimates, thereby influencing the consistency behind the decision-making process and therefore impacting the reliability of the model's predictions. For these purposes, the Brier score serves as a metric to quantify calibration quality, effectively measuring how accurately predicted probabilities align with the actual outcomes.

Having tuned different XGBoost algorithms through the H20 AutoML framework, our best configuration (Table 7.4) provides the results presented in Table 7.5. Moreover, **Appendix B** provides a comparison of the extreme gradient boosting with respect to the different data imbalance management techniques and the baseline performance on the original dataset.

Model: XGBoost__grid1__model__377	**Cross-Validation (5 folds)**	**Test set**
AUC-ROC	0.9970 ± 0.0002	0.9922
AUC-PR	0.9971 ± 0.0125	0.9703
Brier Score	0.0192 ± 0.0027	0.0221

Table 7.5: XGBoost chosen configuration performance over the 5-Fold Cross-Validation and Test set

Upon reviewing Table 7.5, it becomes evident that there is a major decline in the performance of the ROC-PR metric from the results obtained through Cross-validation to those from the Test phase. Such a trend may raise concerns about potential model overfitting to the training data. To delve deeper into this pattern, we can inspect the model's learning curve (Fig 7.8).

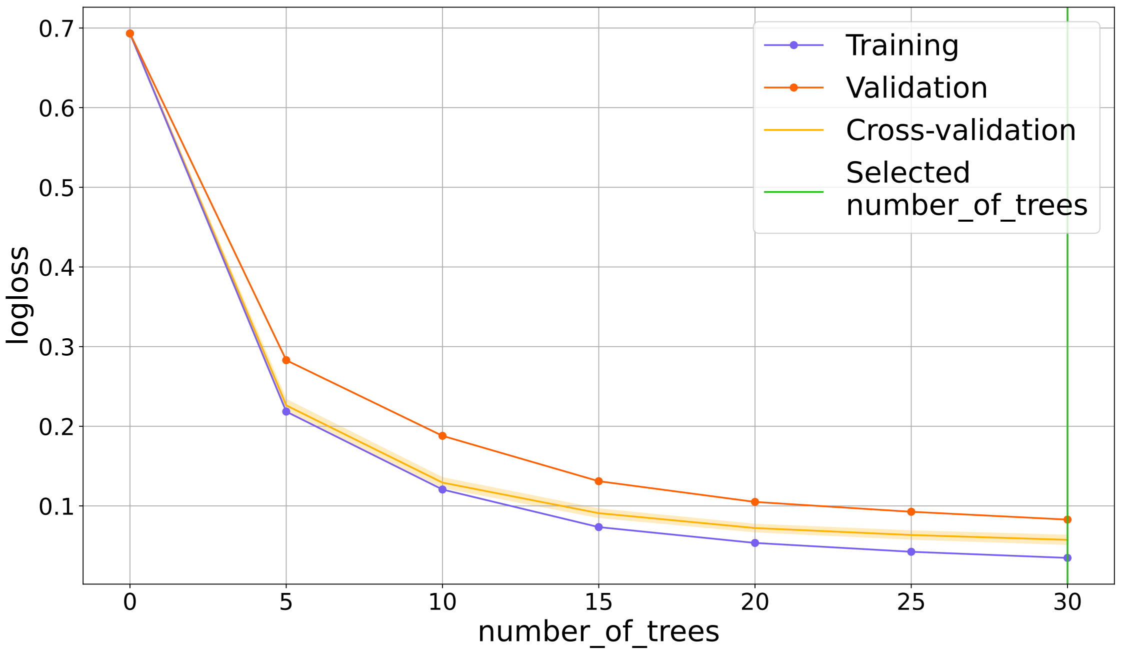


Figure 7.8: XGBoost Learning Curve: logloss vs. number of trees trained

By analyzing the learning curves, which depict the logloss dependence on the number of trained regression trees, particularly across the "Training" (the complete oversampled dataset), "Cross-Validation" (employing 5-fold Cross-Validation), and "Validation" (Test set) scenarios, we gain insight into this phenomenon. Interestingly, the performance degradation, while not statistically significant, cannot be attributed to an overfitting tendency of the model to the training data. Indeed, all the curves follow a consistent trajectory as the model's complexity increases, without any signs of divergence in behavior.

Chapter 8

Results

In the pursuit of addressing the model's effectivity and reliability, the analysis of performance is undoubtedly a critical phase. It provides insights into the model's predictive capabilities and areas that warrant improvement. However, beyond the performance metric lies the imperative to manage the model's fairness and explainability. This chapter delves into the significance of this crucial post-evaluation stage, elucidating the model's training and results in such a way as to prevent the possibility of biased decisions, enhancing transparency and accountability.

8.1 Model's Admissibility: Fairness

After having examined the performances of the developed model, ensuring that the algorithm's decisions are free from bias and discrimination is not merely a technical concern, but an ethical imperative. This aspect gains even greater importance within the specific domain of analysis. In this case, the act of treating an individual unfairly due to characteristics like gender, race or age can lead to legal liabilities and societal biases.

To effectively tackle this aspect of analysis, we have directed our attention towards demographical variables. Those variables are some of the information collected in the ingested dataset giving insight into the churn behavior of the customers under analysis. Therefore, our study unfolds two main ideas of study on these "sensible" features:

- **Exploring the significance of excluded predictors**: Considering those demographical features excluded during the Data ingestion phase, we aim to understand the existence of a possible significant correlation between these attributes and the feature identified by the model as the main driver of its decision-making process.

- **Understand their impact on the response variable**: Examine the demographic feature upon which the model was trained, quantifying their contribution to the response variability.

Our initial examination seeks to determine whether pertinent information from the excluded sensitive features is effectively carried on within the prominent feature guiding the selected model's customer churning identification. To achieve this, we leverage the model's feature importance metrics (Fig 8.1). Due to the qualitative and quantitative nature of the respectively sensitive and most relevant features, we employ a violin plot (Fig 8.2) representation to showcase the distribution of the continuous variables across diverse categories.

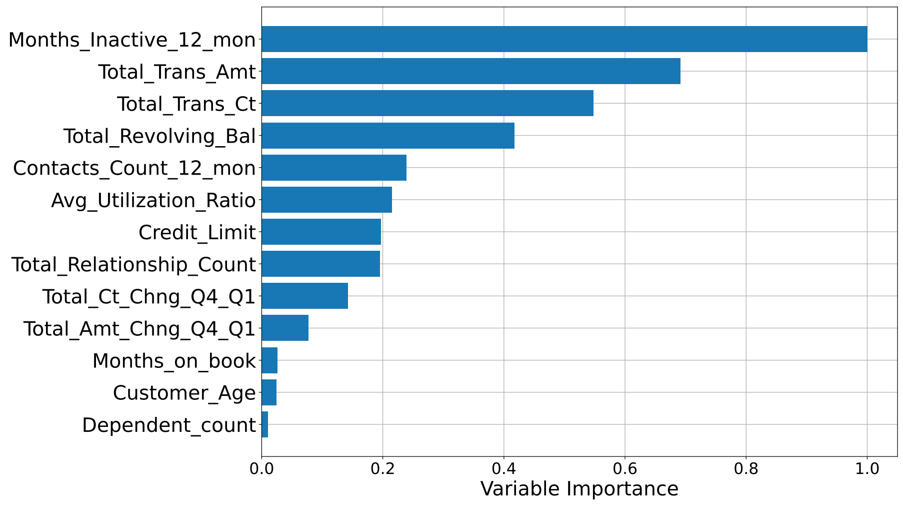
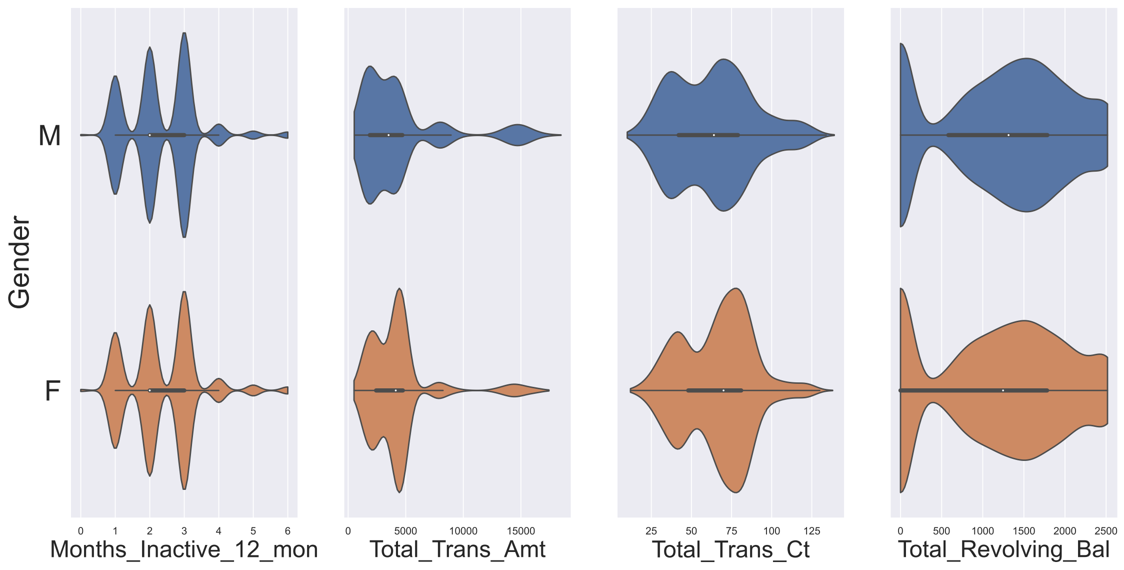
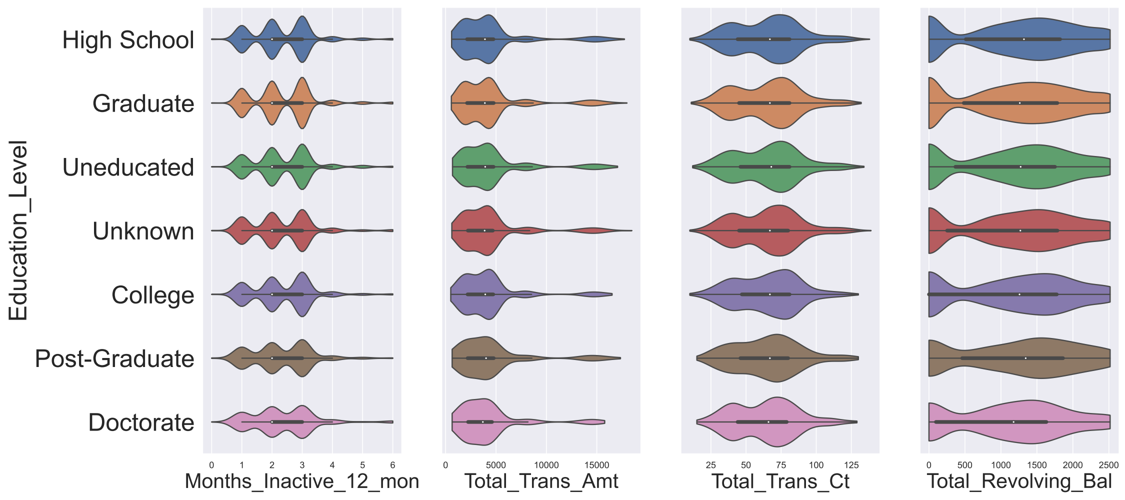


Figure 8.1: XGBoost Features Importance





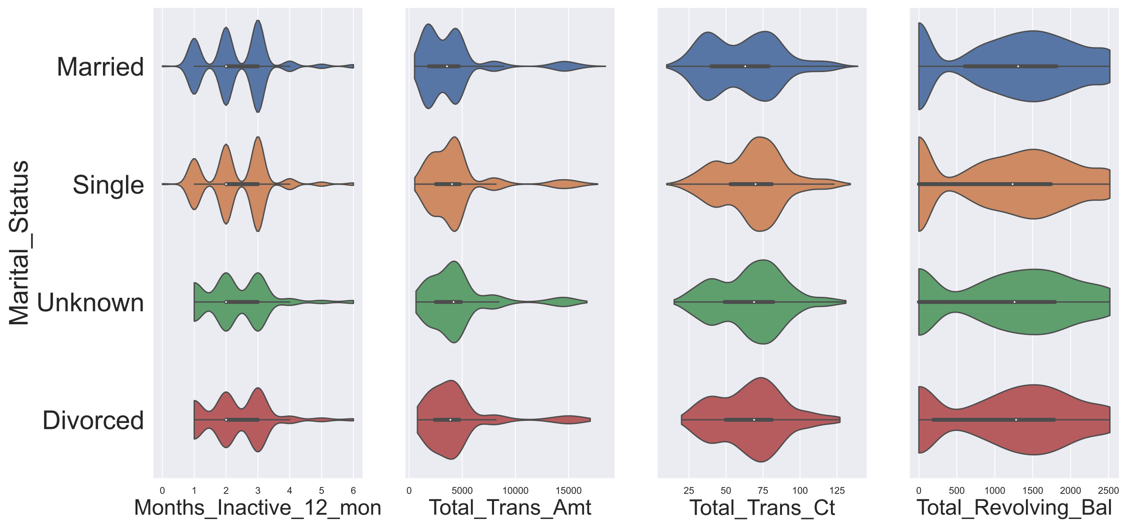
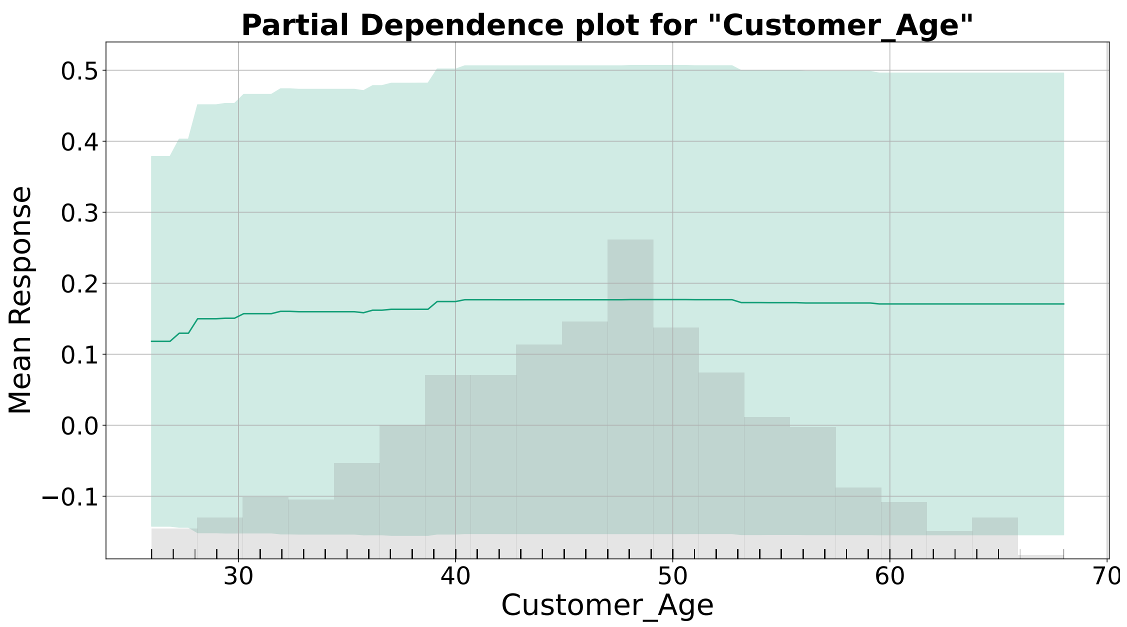


Figure 8.2: Violin Plots of sensitive features: "Gender", "Education_Level", "Marital_Status"

By evaluating the divergences in data distribution across the different categorical variables, it is evident how different shapes have a minimal difference as the sensitive feature level shifts. However, it is important to note that those superficial differences, since within the sensitive feature levels the continuous variables' essential characteristic is preserved, can be attributed to the inherent imbalance present inside the dataset, which is not only with respect to the class labels but also in the representation of the diverse categorical feature levels.



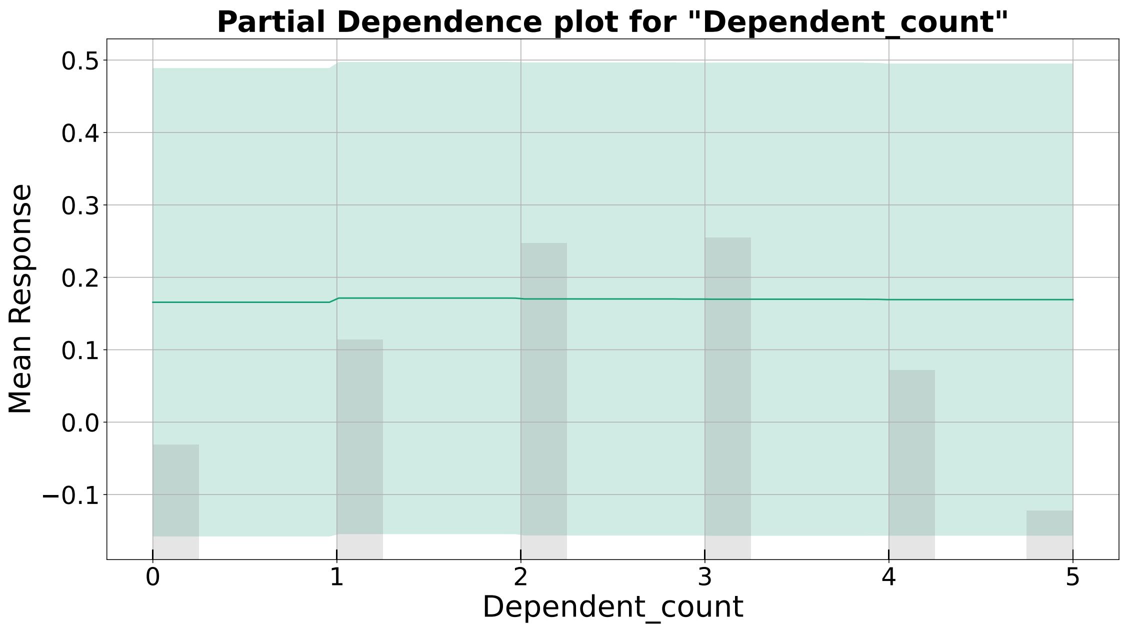


Figure 8.3: Partial Dependency Plots of the features: "Customer_age" and "Dependent_count"

Furthermore, by closely examining the developed model's feature importance metrics, it becomes evident that the demographic features "Dependent_count" and "Customer_Age" do not significantly contribute to the prediction of customer churning behaviours. To enforce this observation, we conducted a more in-depth analysis using partial dependency plots for these two variables. These plots reveal the feature influences, both linear and non-linear, of these variables on the model's predictions while holding other features constant. Carefully looking at Fig 8.3, it is possible to observe a slight mean response variation identified by decreasing customer ages, however not able to identify a discriminatory algorithm behaviour by itself. Therefore it is possible to confirm how both sensitive predictors involved in the model's training do not have a substantial impact on the response variable, suggesting that the chosen classification algorithm remains unbiased in relation to these specific features' value variations.

By embracing a comprehensive approach that not only examines isolated feature importance but also delves into the complex interactions among features, we are able to demonstrate robustness and trustworthiness in the developed analytical framework.

8.2 Model's Explainability

Explainability, together with fairness, is vital for users who require insights into why a model arrived at a particular decision. Especially in the financial contest, and precisely in the tracking of customer churning from their credit card usage, it offers a clear window into the decision-making process of the model, shedding light on why a particular customer is predicted to churn. This transparency is essential for financial institutions to comprehend the factors influencing churn predictions, enabling them to take informed actions to retain customers Moreover, in a domain as sensitive as finance, being able to provide explainability behind the model's decisions instils trust, helping avoid biases and to facilitate the alignment of model predictions with business strategies. By providing insights into the "why" behind predictions, explainability empowers stakeholders to make strategic decisions, improve customer retention efforts, and ensure ethical and equitable treatment of all customers.

Typically, most high-performing machine learning models are exploited due to their ability to capture intricate relationships and interactions within data. Indeed, extreme gradient boosting falls into the so-called complex/black-box model category, presenting extremely good predictive behaviour but at the same time a not easy-to-decode decision-making process.

In order to make this process transparent, Shapley values, a concept from cooperative game theory is used. When applied to machine learning models, they

help to provide a comprehensive understanding of how each feature contributes to a particular prediction [60]. Despite, feature importance scores, indicating the magnitude of a feature's influence on predictions, Shapley values fill the gap left on the lack of perspective on feature interactions, considering the average marginal contribution of a specific feature value across all possible predictors coalitions. While feature importance typically ranks features based on their individual contributions to the model's predictive power, Shapley values consider the collaborative impact of features in different combinations.
Shapley value can provide the model's interpretability under two different aspects:

- **Global Interpretability**: The collective SHAP values describe the expected behaviour of a machine learning model with respect to the whole distribution of values for its input variables. In this way, it can be shown how much each predictor contributes, positively or negatively, to the target variable prediction.

- **Local Interpretability**: It provides a tailored insight into the decision-making process by examining individual predictions, especially required in applications like the finance domain where singular decisions have high stakes. It sheds light on the model's reasoning behind a particular prediction based on how additively each feature contributes to that specific outcome. This provides a way to understand unexpected or counterintuitive model outputs, but also to establish clear lines of accountability on the final prediction. Furthermore, it provides a way for the model's validation, being analysed by the stakeholders it can be assessed where the model's decisions align with domain knowledge and expectations.

8.2.1 Model's Explainability: Global

To enhance the global interpretability of a model using Shapley values, we employ a summary plot that illustrates the relationship between feature values and their influence on the model's final prediction.

In Fig 8.4, each data point represents a Shapley value associated with a particular feature and instance. The vertical positioning order along the y-axis is determined by the feature's importance. The horizontal x-axis reflects the direction of the relative impact of the Shapley value, indicating each feature's value contribution to the model's prediction. Additionally, the colour of the data points represents the normalized magnitude of the features, from low (in blue) to high (in red). To prevent overlap, the feature's points presenting the same Shapley values are jittered, offering insight into the distribution of the Shapley values for each feature. Through the figure, it becomes evident that the model's ability in the identification of potential churning behaviour is primarily influenced by factors such as a decreasing number of transactions in relation to the total amount spent, the duration of inactivity registered over the year and the revolving balance amount on the credit card. Conversely, most of the other features exhibit a lower impact, as evidenced by a narrower range of Shapley values registered.

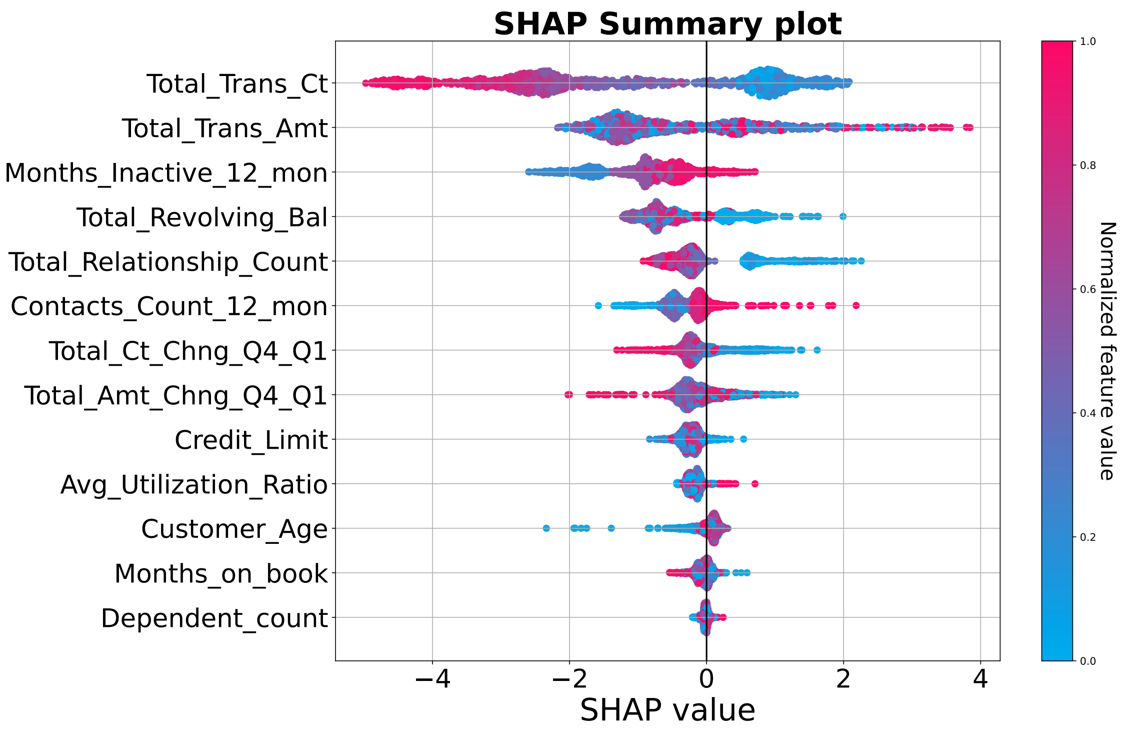


Figure 8.4: Shapley Summary Plot

8.2.2 Model's Explainability: Local

To enhance our understanding of why the developed model generates specific predictions and gain insights into the contributions of individual features to the model's classification, we have employed a horizontal barplot, providing a comprehensible visual representation of the feature contributions. Rather than providing complex mathematical equations, stakeholders can intuitively grasp the reasoning behind a specific prediction. This transparency not only improves trust in the model's decision-making process but also empowers businesses with the ability to implement tailored strategies for each individual customer, reducing churning behaviour and maximising the customer's lifetime value.

By examining the influence of feature values on the model's predictions for customers with significantly high and low churn rates (approximately 0.99 and 0.001, respectively), we can precisely identify those attributes that have a substantial customer-specific impact on increasing or decreasing the churning probability.

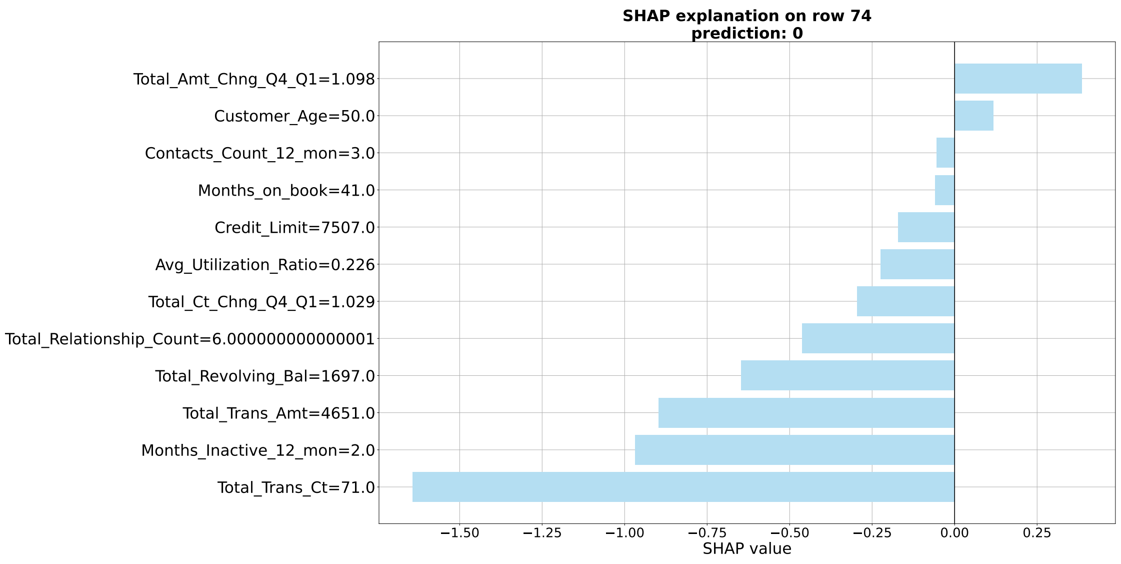


Figure 8.5: Features contribution barplot for not churning customer: Probability of Churning ≈ 0.01

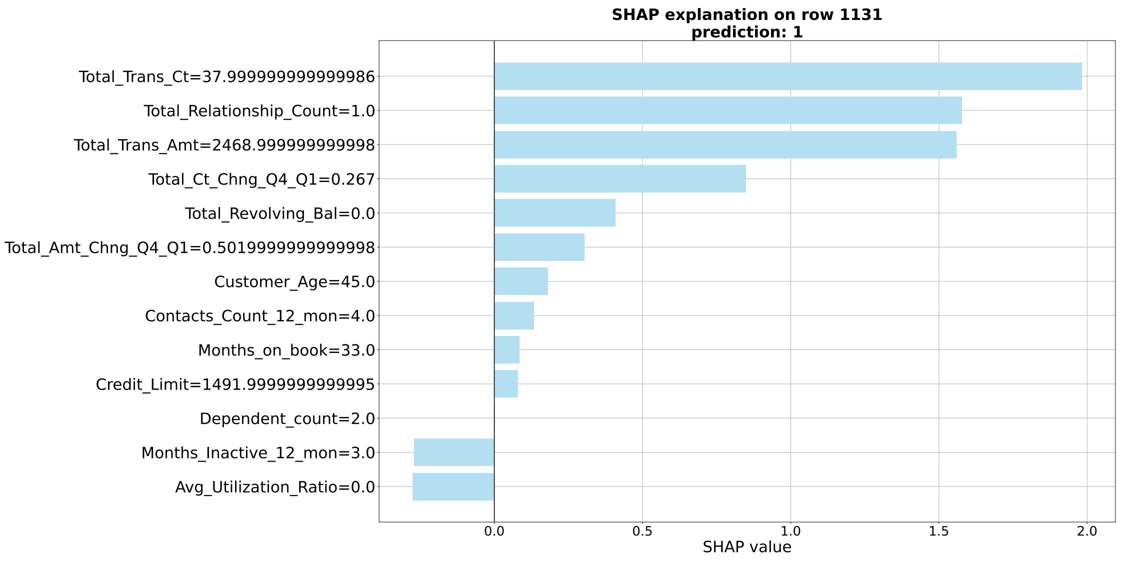


Figure 8.6: Features contribution barplot for churning customer: Probability of Churning ≈ 0.99

Fig 8.5 provides an explanation of the model's prediction of a particular non-churning customer. We can observe that a high transaction count, coupled with a low number of inactive moths and a relatively high total revolving balance, can help the model to classify a specific person as not likely to churn.

Conversely, in Fig 8.6, a reduced number of transactions and total amount spent, followed by zero debts on the credit card are indicative of the model of a customer who may be prone to churning, even in the presence of a low number of inactive months.

Chapter 9

Conclusions

The provided work tries to propose a possible way of addressing credit card customer churn but in a way that could be properly replicated, previous to a correct data analysis and goals identification, to different customer relationship management domains.

Our research underscores the significance of establishing a comprehensive end-to-end pipeline to address all the main aspects regarding data processing within an organization committed to the delivery of data-driven solutions. Beginning with a suited exploration of the relevant domain, it has been possible to leverage cloud solutions like IBM DataStage to design a well-suited data ingestion process. This flow has been able to provide a data schema structure appropriate for the pre-processing phase which has involved the generation of synthetic minority class data to balance the initial class distribution exploiting a literature generative approach, cWGANGP. Furthermore, leveraging industry state-of-the-art solutions like H2O AutoML to develop our final predictive model, we have ensured a robust and efficient model development process.

Despite the high performances achieved, our focus stands also towards the offering of a practical guideline for handling sensitive information about individuals, from both technical and ethical points of view, that can be shared in many other domains. This is a consequence of the awareness that data-driven business solutions are for most built upon personal information that should be treated with utmost care.

Hypothetically, in our work, we could have just eliminated personal users' information and provided a classification model. However, this straightforward approach hides a preliminary and fundamental inspection of the input dataset, in which possible correlated factors are not taken into consideration, providing a prediction framework concealing strong biases towards the decisions taken.

In our case, through a proper exploratory analysis, we have been able to provide insights into the way to navigate and uncover patterns with sensitive information, safeguarding individual's privacy. This is also enforced through a provided perspective on the decision-making model's interpretation, which has ensured good performances shedding light on both ethical and legal requirement assessment.

In essence, our machine-learning pipeline tries to set a standard for how data should be handled and interpreted in the nowadays data world, in which privacy and ethics are paramount, being not only design-specific for the analyzed dataset but also suitable for other domains.

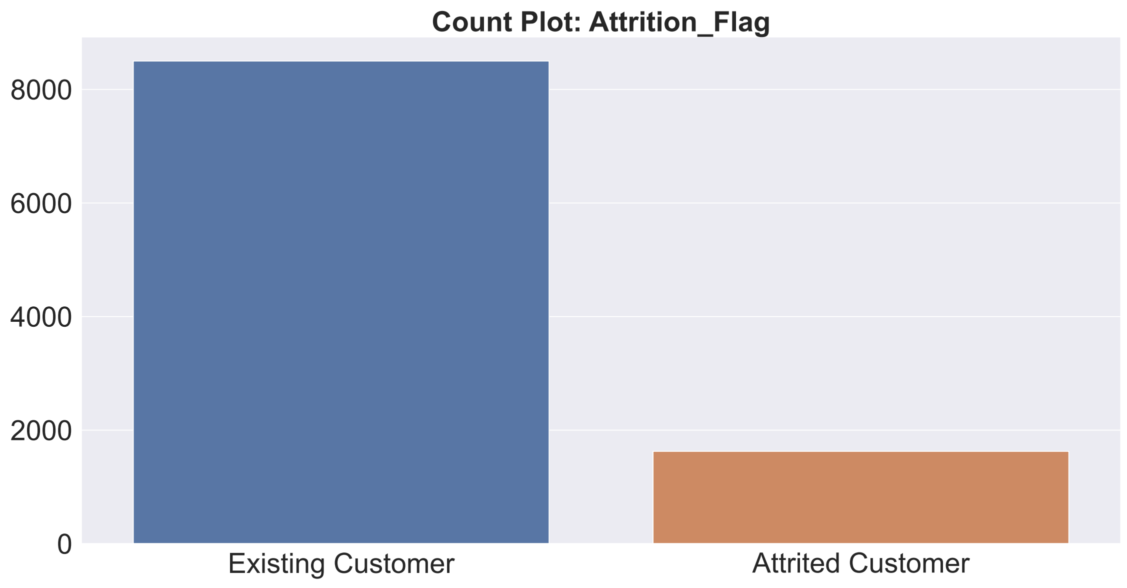
Appendix A

Data Distributions

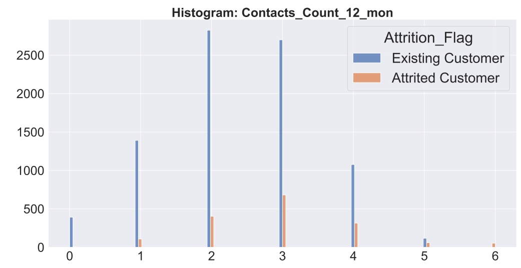
Differently from the "Functional Exploratory Data Analysis" provided, the following section enriches the analysis by showing the different features' univariate distribution.

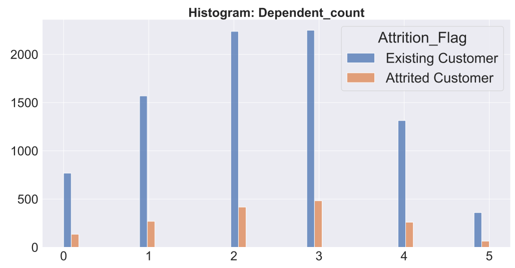
However, since the exploited dataset provides both numerical and categorical attributes, two different plots' typologies are used:

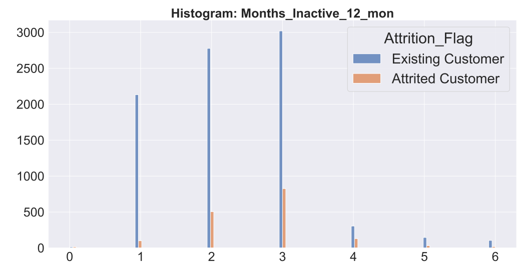
- Target variable "**Attrition Flag**" count plot:

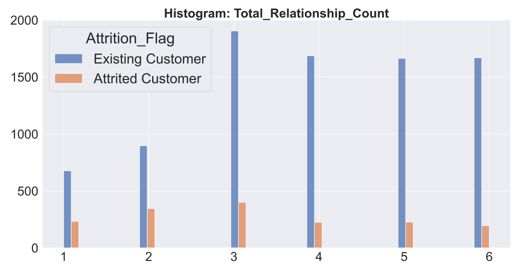


- Discrete numerical variables histograms:

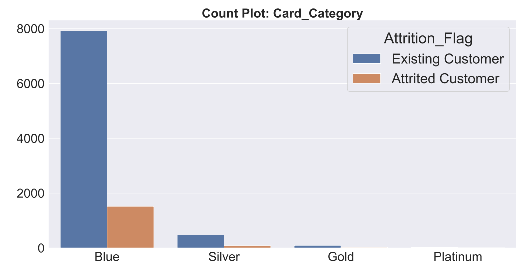


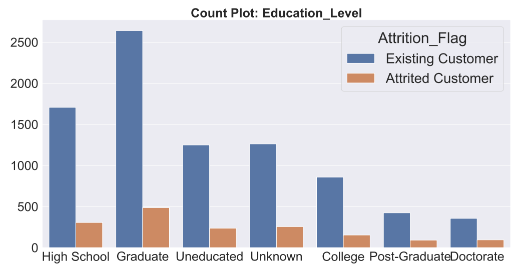


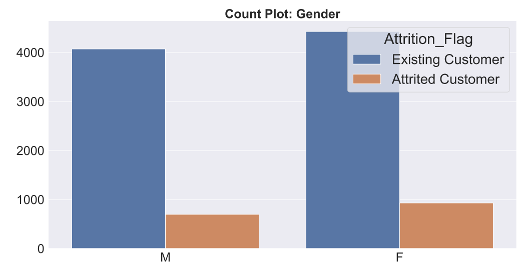


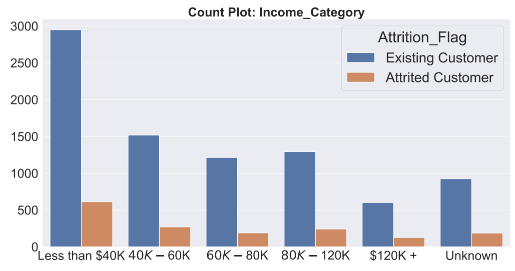


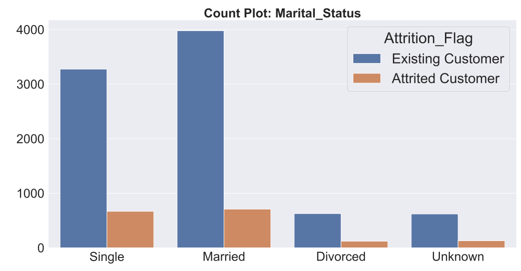
- Categorical variables count plots:



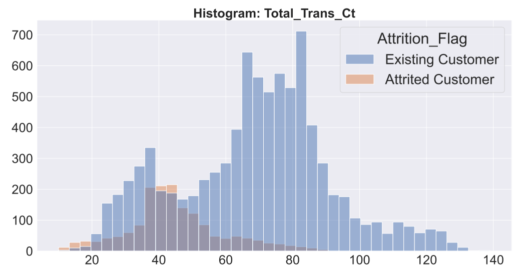


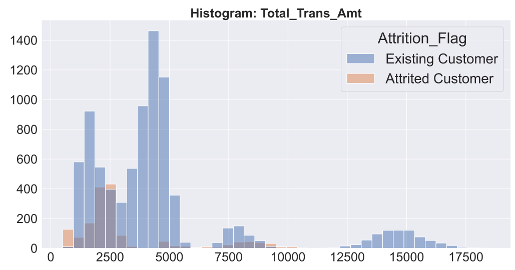


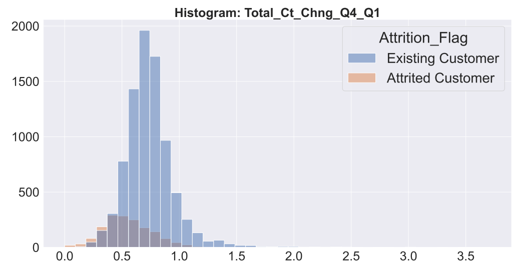


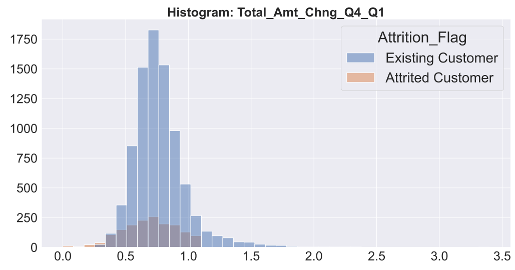


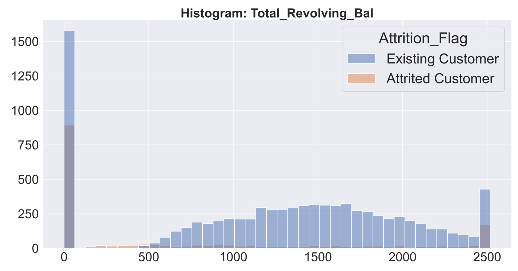
- Continuous numerical variables histograms:

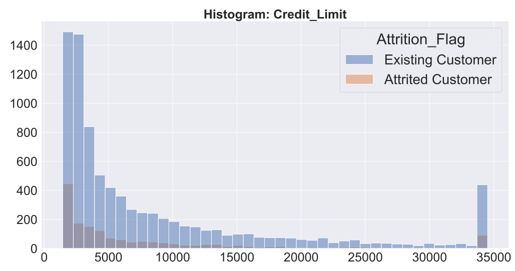


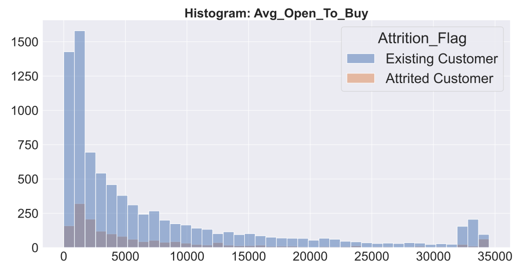


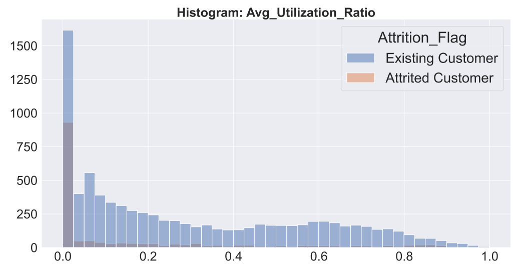


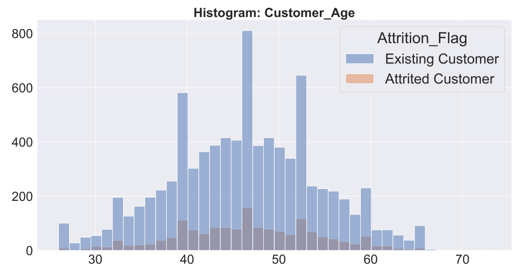


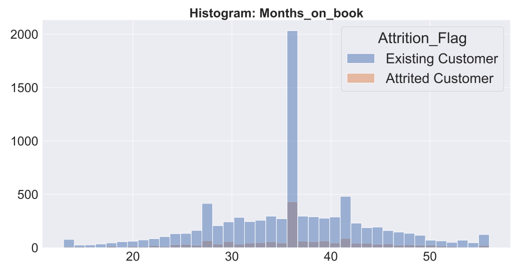












Appendix B

Imbalance Management Comparisons:

To provide a comparative assessment, the following section conducts an analysis of the data imbalance management techniques performance on the "Credit Card Customer Attrition" dataset.

To ensure a meticulous evaluation of performance, we fine-tuned the XGBoost algorithm for each dataset, employing the H2O AutoML Framework. We maintained consistent configurations that were leveraged throughout the development of the pipeline.

Furthermore, to provide a comprehensive evaluation of the impact of both algorithm-level and data-level methodologies, we explored different scenarios. In one scenario, the baseline, the classification algorithm was trained on the unprocessed original dataset (Imbalance Management Method: None). While, for the imbalance management methodologies is considered the dataset provided by the ETL process on which the customized feature selection is applied.

The reported results are expressed in terms of ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) and PR-AUC (Precision-Recall - Area Under the Curve) scores, all computed on the same test dataset that was used to evaluate the overall pipeline performance.

Imbalance Management Method:	**ROC-AUC**	**PR-AUC**
None	0.9917	0.9678
Sample Weight	0.9915	0.9687
ROS	0.9917	0.9678
SMOTE	0.9919	0.9654
Borderline-SMOTE	0.9914	0.9621
SVM-SMOTE	0.9912	0.9662
KMeans-SMOTE	0.9915	0.9604
ADASYN	0.9899	0.9580
RUS	0.9894	0.9486
NEAR-MISS	0.9886	0.9565
SMOTE Tomek	0.9922	0.9649

Table B.1: ROC-AUC and PR-AUC performance comparisons of the XGBoost classifier on the original dataset and on the imbalance management techniques.

Bibliography

[1] Sunil Gupta, Donald R. Lehmann, and Jennifer Ames Stuart. «Valuing Customers». In: *Journal of Marketing Research* 41.1 (2004), pp. 7–18. DOI: 10.1509/jmkr.41.1.7.25084. eprint: https://doi.org/10.1509/jmkr.41.1.7.25084. URL: https://doi.org/10.1509/jmkr.41.1.7.25084 (cit. on p. 1).

[2] Antreas Athanassopoulos. «Customer Satisfaction Cues To Support Market Segmentation and Explain Switching Behavior». In: *Journal of Business Research* 47 (Mar. 2000), pp. 191–207. DOI: 10.1016/S0148-2963(98)00060-5 (cit. on p. 1).

[3] Mark R. Colgate and Peter J. Danaher. «Implementing a Customer Relationship Strategy: The Asymmetric Impact of Poor versus Excellent Execution». In: *Journal of the Academy of Marketing Science* 28.3 (2000), pp. 375–387. DOI: 10.1177/0092070300283006. eprint: https://doi.org/10.1177/0092070300283006. URL: https://doi.org/10.1177/0092070300283006 (cit. on p. 1).

[4] Arpit Singh and Anuradha Purohit. «A Survey on Methods for Solving Data Imbalance Problem for Classification». In: *International Journal of Computer Applications* 127 (2015), pp. 37–41. URL: https://api.semanticscholar.org/CorpusID:356390 (cit. on p. 6).

[5] Joffrey Leevy, Taghi Khoshgoftaar, Richard Bauder, and Naeem Seliya. «A survey on addressing high-class imbalance in big data». In: *Journal of Big Data* 5 (Nov. 2018). DOI: 10.1186/s40537-018-0151-6 (cit. on p. 6).

[6] K Veropoulos, ICG Campbell, and N Cristianini. «Controlling the Sensitivity of Support Vector Machines». English. In: *Proceedings of the International Joint Conference on Artificial Intelligence, Stockholm, Sweden (IJCAI99)*. Other: Workshop ML3. 1999, pp. 55–60 (cit. on p. 7).

[7] Chris Drummond and Robert Holte. «C4.5, Class Imbalance, and Cost Sensitivity: Why Under-Sampling beats OverSampling». In: *Proceedings of the ICML'03 Workshop on Learning from Imbalanced Datasets* (Jan. 2003) (cit. on p. 7).

[8] Gary M. Weiss. «Mining with Rarity: A Unifying Framework». In: *SIGKDD Explor. Newsl.* 6.1 (June 2004), pp. 7–19. ISSN: 1931-0145. DOI: 10.1145/1007730.1007734. URL: https://doi.org/10.1145/1007730.1007734 (cit. on p. 8).

[9] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo Cristiano Prati, B. Krawczyk, and Francisco Herrera. «Learning from Imbalanced Data Sets». In: *Cambridge International Law Journal.* 2018, pp. 7–9. URL: https://api.semanticscholar.org/CorpusID:53046396 (cit. on p. 9).

[10] *Analytics Vidhya Undersampling and oversampling: An old and a new approach.* https://medium.com/analytics-vidhya/undersampling-and-oversampling-an-old-and-a-new-approach-4f984a0e8392. Accessed: 2023-03-20 (cit. on pp. 9, 20).

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. «SMOTE: Synthetic Minority Over-sampling Technique». In: *Journal of Artificial Intelligence Research* 16 (June 2002), pp. 321–357. DOI: 10.1613/jair.953. URL: https://doi.org/10.1613%2Fjair.953 (cit. on p. 10).

[12] Ronaldo Prati, Gustavo Batista, and Maria-Carolina Monard. «Learning with Class Skews and Small Disjuncts». In: Sept. 2004, pp. 296–306. ISBN: 978-3-540-23237-7. DOI: 10.1007/978-3-540-28645-5_30 (cit. on p. 10).

[13] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. «Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling TEchnique for Handling the Class Imbalanced Problem». In: vol. 5476. Apr. 2009, pp. 475–482. ISBN: 978-3-642-01306-5. DOI: 10.1007/978-3-642-01307-2_43 (cit. on p. 10).

[14] M. Aldraimli, Daniele Soria, Jim Parkinson, Elizabeth Thomas, Jimmy Bell, Miriam Dwek, and Thierry Chaussalet. «Machine learning prediction of susceptibility to visceral fat associated diseases». In: *Health and Technology* 10 (July 2020), pp. 925–944. DOI: 10.1007/s12553-020-00446-1 (cit. on p. 11).

[15] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. «Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning». In: vol. 3644. Sept. 2005, pp. 878–887. ISBN: 978-3-540-28226-6. DOI: 10.1007/11538059_91 (cit. on p. 11).

[16] X. Zheng. *SMOTE Variants for Imbalanced Binary Classification: Heart Disease Prediction.* University of California, Los Angeles, 2020. URL: https://books.google.it/books?id=0SQTzgEACAAJ (cit. on pp. 12, 15).

[17] Hien Nguyen, Eric Cooper, and Katsuari Kamei. «Borderline over-sampling for imbalanced data classification». In: *International Journal of Knowledge Engineering and Soft Data Paradigms* 3 (Apr. 2011), pp. 4–21. DOI: `10.1504/IJKESDP.2011.039875` (cit. on p. 13).

[18] Georgios Douzas, Fernando Bacao, and Felix Last. «Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE». In: *Information Sciences* 465 (Oct. 2018), pp. 1–20. DOI: `10.1016/j.ins.2018.06.056`. URL: `https://doi.org/10.1016%2Fj.ins.2018.06.056` (cit. on pp. 15, 16).

[19] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. «ADASYN: Adaptive synthetic sampling approach for imbalanced learning». In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. 2008, pp. 1322–1328. DOI: `10.1109/IJCNN.2008.4633969` (cit. on p. 17).

[20] Chris Seiffert, Taghi M. Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. «RUSBoost: A Hybrid Approach to Alleviating Class Imbalance». In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 40.1 (2010), pp. 185–197. DOI: `10.1109/TSMCA.2009.2029559` (cit. on pp. 19, 20).

[21] J. Zhang and I. Mani. «KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction». In: *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets*. 2003 (cit. on p. 20).

[22] Gustavo Batista, Ronaldo Prati, and Maria-Carolina Monard. «A Study of the Behavior of Several Methods for Balancing machine Learning Training Data». In: *SIGKDD Explorations* 6 (June 2004), pp. 20–29. DOI: `10.1145/1007730.1007735` (cit. on p. 22).

[23] «Two Modifications of CNN». In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6.11 (1976), pp. 769–772. DOI: `10.1109/TSMC.1976.4309452` (cit. on p. 22).

[24] *Kaggle Resampling strategies for imbalanced datasets*. https://www.kaggle.com/code/rafjaa/r strategies-for-imbalanced-datasets. Accessed: 2023-03-27 (cit. on p. 22).

[25] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014. arXiv: `1406.2661 [stat.ML]` (cit. on pp. 23, 27).

[26] *Jonathan Hui GAN — Wasserstein GAN WGAN-GP*. https://jonathan-hui.medium.com/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490. Accessed: 2023-04-10 (cit. on p. 24).

[27] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, and Sushil. In: 11.10 (June 2018), pp. 1071–1083. DOI: `10.14778/3231751.3231757`. URL: `https://doi.org/10.14778%2F3231751.3231757` (cit. on p. 25).

[28] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: `1511.06434 [cs.LG]` (cit. on p. 25).

[29] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F. Stewart, and Jimeng Sun. *Generating Multi-label Discrete Patient Records using Generative Adversarial Networks*. 2018. arXiv: `1703.06490 [cs.LG]` (cit. on p. 25).

[30] Mrinal Kanti Baowaly, Chia-Ching Lin, Chao-Lin Liu, and Kuan-Ta Chen. «Synthesizing electronic health records using improved generative adversarial networks». In: *Journal of the American Medical Informatics Association* 26 (2018), pp. 228–241. URL: `https://api.semanticscholar.org/CorpusID:54479855` (cit. on p. 25).

[31] Lei Xu and Kalyan Veeramachaneni. *Synthesizing Tabular Data using Generative Adversarial Networks*. 2018. arXiv: `1811.11264 [cs.LG]` (cit. on p. 25).

[32] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. *Modeling Tabular data using Conditional GAN*. 2019. arXiv: `1907.00503 [cs.LG]` (cit. on p. 25).

[33] 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». In: *Information Sciences* 479 (2019), pp. 448–455. ISSN: 0020-0255. DOI: `https://doi.org/10.1016/j.ins.2017.12.030`. URL: `https://www.sciencedirect.com/science/article/pii/S0020025517311519` (cit. on p. 25).

[34] Yu-Jun Zheng, Xiao-Han Zhou, Wei-Guo Sheng, Yu Xue, and Sheng-Yong Chen. «Generative adversarial network based telecom fraud detection at the receiving bank». In: *Neural Networks* 102 (2018), pp. 78–86. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/j.neunet.2018.02.015`. URL: `https://www.sciencedirect.com/science/article/pii/S0893608018300698` (cit. on p. 26).

[35] Georgios Douzas and Fernando Bacao. «Effective data generation for imbalanced learning using conditional generative adversarial networks». In: *Expert Systems with Applications* 91 (2018), pp. 464–471. ISSN: 0957-4174. DOI: `https://doi.org/10.1016/j.eswa.2017.09.030`. URL: `https://www.sciencedirect.com/science/article/pii/S0957417417306346` (cit. on p. 26).

[36] Insaf Ashrapov. *Tabular GANs for uneven distribution.* 2020. arXiv: 2010.00638 [cs.LG] (cit. on p. 26).

[37] Justin Engelmann and Stefan Lessmann. *Conditional Wasserstein GAN-based Oversampling of Tabular Data for Imbalanced Learning.* 2020. arXiv: 2008.09202 [cs.LG] (cit. on pp. 26, 32).

[38] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN.* 2017. arXiv: 1701.07875 [stat.ML] (cit. on pp. 27, 28).

[39] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. *Improved Training of Wasserstein GANs.* 2017. arXiv: 1704.00028 [cs.LG] (cit. on p. 28).

[40] Lilian Weng. *From GAN to WGAN.* 2019. arXiv: 1904.08994 [cs.LG] (cit. on p. 29).

[41] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets.* 2014. arXiv: 1411.1784 [cs.LG] (cit. on p. 29).

[42] Augustus Odena, Christopher Olah, and Jonathon Shlens. *Conditional Image Synthesis With Auxiliary Classifier GANs.* 2017. arXiv: 1610.09585 [stat.ML] (cit. on p. 29).

[43] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax.* 2017. arXiv: 1611.01144 [stat.ML] (cit. on p. 30).

[44] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. *Deep Cross Network for Ad Click Predictions.* 2017. arXiv: 1708.05123 [cs.LG] (cit. on p. 31).

[45] Vadim Borisov, Tobias Leemann, Kathrin Sessler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. «Deep Neural Networks and Tabular Data: A Survey». In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–21. DOI: 10.1109/tnnls.2022.3229161. URL: https://doi.org/10.1109%2Ftnnls.2022.3229161 (cit. on pp. 34, 36).

[46] Sheikh Amir Fayaz, Majid Zaman, Sameer Kaul, and Muheet Ahmed Butt. «Is Deep Learning on Tabular Data Enough? An Assessment». In: *International Journal of Advanced Computer Science and Applications* 13.4 (2022). DOI: 10.14569/IJACSA.2022.0130454. URL: http://dx.doi.org/10.14569/IJACSA.2022.0130454 (cit. on p. 34).

[47] Casper Solheim Bojer and Jens Peder Meldgaard. «Kaggle forecasting competitions: An overlooked learning opportunity». In: *International Journal of Forecasting* 37.2 (Apr. 2021), pp. 587–603. DOI: 10.1016/j.ijforecast.2020.07.007. URL: https://doi.org/10.1016%2Fj.ijforecast.2020.07.007 (cit. on pp. 34, 36).

[48] Yoav Freund and Robert E. Schapire. «A Short Introduction to Boosting». In: 1999. URL: `https://api.semanticscholar.org/CorpusID:9621074` (cit. on p. 35).

[49] Lien Rodríguez-López, David Bustos Usta, Lisandra Bravo Alvarez, Iongel Duran-Llacer, Andrea Lami, Rebeca Martínez-Retureta, and Roberto Urrutia. «Machine Learning Algorithms for the Estimation of Water Quality Parameters in Lake Llanquihue in Southern Chile». In: *Water* 15.11 (2023). ISSN: 2073-4441. DOI: `10.3390/w15111994`. URL: `https://www.mdpi.com/2073-4441/15/11/1994` (cit. on p. 35).

[50] Tianqi Chen and Carlos Guestrin. «XGBoost». In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016. DOI: `10.1145/2939672.2939785`. URL: `https://doi.org/10.1145%2F2939672.2939785` (cit. on p. 35).

[51] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. CRC Press, 2017. ISBN: 9781138469525. URL: `https://books.google.it/books?id=CmK4tAEACAAJ` (cit. on p. 35).

[52] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. «Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors)». In: *The Annals of Statistics* 28.2 (2000), pp. 337–407. DOI: `10.1214/aos/1016218223`. URL: `https://doi.org/10.1214/aos/1016218223` (cit. on p. 38).

[53] Jerome Friedman. «Stochastic Gradient Boosting». In: *Computational Statistics Data Analysis* 38 (Feb. 2002), pp. 367–378. DOI: `10.1016/S0167-9473(01)00065-2` (cit. on p. 38).

[54] L Breiman. «Random Forests». In: *Machine Learning* 45 (Oct. 2001), pp. 5–32. DOI: `10.1023/A:1010950718922` (cit. on p. 39).

[55] Stephen Tyree, Kilian Weinberger, Kunal Agrawal, and Jennifer Paykin. «Parallel Boosted Regression Trees for Web Search Ranking». In: Mar. 2011. DOI: `10.1145/1963405.1963461` (cit. on p. 39).

[56] C. Ferri, J. Hernández-Orallo, and R. Modroiu. «An experimental comparison of performance measures for classification». In: *Pattern Recognition Letters* 30.1 (2009), pp. 27–38. ISSN: 0167-8655. DOI: `https://doi.org/10.1016/j.patrec.2008.08.010`. URL: `https://www.sciencedirect.com/science/article/pii/S0167865508002687` (cit. on p. 41).

[57] LEAPS by Analyttica. *Predict Credit Card Customer Attrition - Application of Logistic Regression*. `https://enterprise.1leaps.com/cases/9574` [Accessed: (20-02-2023)]. 2022 (cit. on p. 44).

[58] *IBM Cloud Pak for Data Overview of Cloud Pak for Data as a Service.* `https://dataplatform.cloud.ibm.com/docs/content/wsj/getting-started/overview-cpdaas.html?context=cpdaas&locale=en`. Accessed: 2023-05-15 (cit. on p. 51).

[59] Erin LeDell and Sebastien Poirier. «H2O AutoML: Scalable Automatic Machine Learning». In: *7th ICML Workshop on Automated Machine Learning (AutoML)* (July 2020). URL: `https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf` (cit. on p. 59).

[60] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions.* 2017. arXiv: `1705.07874 [cs.AI]` (cit. on p. 68).