UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Telecommunications Engineering

# Balancing data with SMOTE variants using supervised machine learning algorithms to predict churn rate.

Master's Thesis

Master's Degree in Telecommunication Engineering

AUTHOR: Martínez Cerdá, Luis José

Tutor: González Ladrón de Guevara, Fernando Raimundo

Cotutor: Fernández Diego, Marta

ACADEMIC YEAR: 2021/2022

## Resumen

Este proyecto incluye el desarrollo de un modelo de inteligencia artificial para un conjunto de datos donde la distribución de las clases a predecir no es del todo simétrica. Este hecho, conocido como Imbalance, supone un gran problema si no se tiene en cuenta esta irregularidad de las muestras.

Se analiza un caso de Churn Rate (Porcentaje de abandono de clientes de un servicio) en una empresa de banca digital donde hay mayoría de personas que conservan su cuenta en el banco, lo que denominamos clase mayoritaria, mientras que hay muy pocos clientes que eliminan su cuenta, la clase minoritaria.

Se propone la utilización de algoritmos de balanceo de la familia de SMOTE (Synthetic Minority oversampling Technique) para hacer que se generen más muestras de la clase con menor representación y así resolver este problema. Para probar estas técnicas utilizaremos algoritmos de boosting para predecir. Del mejor resultado, obtendremos nuestro modelo óptimo acorde a las métricas seleccionadas.

El tratado y elección de las métricas es de facto importante en este proyecto, se pretende darle de más profundidad a lo que supone la métrica score elegida, por eso, se ha hecho un análisis de proyectos anteriores con esta fuente de datos de Kaggle para ver posibles patrones en la elección de métricas y estudiar la validez de sus resultados comparados con los nuestros.

Se ha comprobado que en este caso concreto optimizar la métrica de exhaustividad nos puede llegar a dar los mejores resultados para cumplir el objetivo de frenar el mayor número de cierre de cuentas posibles.

## Resum

Aquest projecte inclou el desenvolupament d'un model d'intel·ligència artificial per a un conjunt de dades on la distribució de les classes s'han de predir no és del tot simètrica. Aquest fet, conegut com a Imbalance, suposa un gran problema si no es té en compte aquesta irregularitat de les mostres.

S'analitza un cas de Churn Rate (Percentatge d'abandon de clientes d'un servici) en una empresa de banca digital on hi ha una majoría de persones que conserven el seu compte bancari, el que denominem clase majoritaria, mentres que hi ha molt pocs clients que eliminen el seu compte, la clase minoritaria.

La proposta es la utilització d'algoritmes de balanceig de la família de SMOTE (Synthetic Minority oversampling Technique) per fer que es generen més mostres de la classe amb menor representació i així resoldre aquest problema. Per provar aquestes tècniques utilitzarem algorismes de boosting utilitzats per a predir. Del millor resultat, obtindrem el nostre model òptim d'acord amb les mètriques seleccionades.

El tractat i elecció de les mètriques és un factor important en aquest projecte, es pretén donar més profunditat a la que es suposa la mètrica score triada, per això, s'ha fet un anàlisi de projectes anteriors amb aquest datasource de Kaggle per analitzar possibles patrons en l'elecció de mètriques i estudiar la validesa dels seus resultats comparats amb els nostres.

S'ha comprobat que en aquest cas concret optimitzar la metrica d'exhaustivitat ens pot donar els millors resultats per a cumplir l'objectiu de frenar el major numero de tancaments de comptes possibles.

## Abstract

This project includes the development of an artificial intelligence model for a group of data where the distribution of the classes to be predicted is not completely symmetrical. This fact, known as Imbalance, is a major problem if this irregularity of the samples is not considered.

A case of Churn Rate (Percentage of customer abandonment of a service) is analysed in a digital banking company where there is a majority of people who keep their account in the bank, which we call the majority class, while there are very few customers who delete their account, the minority class.

We propose the use of balancing algorithms from the SMOTE (Synthetic Minority oversampling Technique) family to generate more samples of the underrepresented class and thus solve this problem. To test these techniques, we will use boosting algorithms to predict. From the best result, we will obtain our optimal model according to the selected metrics.

The treatment and choice of metrics is de facto important in this project, it is intended to give more depth to what the chosen score metric means, therefore, we have made an analysis of previous projects with this Kaggle data source to see possible patterns in the choice of metrics and study the validity of their results compared to ours.

It has been proven that in this specific case, optimising the Recall metric can give us the best results to meet the objective of stopping as many account closures as possible.

# Index

## Figures

## Equations

## Tables

# Introduction

The work on the different sample balancing algorithms has grown in the last decades along with the rise and recognition of machine learning as a candidate for solving problems that may be too complex to be solved directly by a human [1].

When we talk about imbalanced learning, we refer to the set of techniques used to combat the inequality of the classes that represent the label of the dataset, that is, the information that we want to obtain or predict using machine learning techniques.

A dataset is imbalanced when it has significantly more samples of one class than of another or others that it must predict. The most used metric for the calculation of imbalance is the IR (Imbalance Rate) and it is nothing more than the majority class divided by the sum of the majority and minority classes. When IR>0.7 we call it an unbalanced dataset, when IR>0.9 then it is a highly unbalanced dataset.

In addition, the rise of available data in cybersecurity sectors, where breaches and frauds are studied, and in the medical sector, where the existence of diseases can be studied with computer vision, have made it even more evident that it is exceedingly difficult in some areas to get data that is balanced (in the aforementioned cases the samples of fraud and the samples of people with diseases will be much smaller than the opposite cases). Also note that in most cases where we have imbalanced data, the class with less data is the one that is relevant for us and the one we want to be able to detect more accurately.

Therefore, for machine learning algorithms that are not able to detect those more rare or unusual cases where unbalanced data can be a big problem, other algorithms, if they have enough knowledge, make up for this ability to find hidden patterns changing parameters that switches the algorithm's operation.

The range of possible data balancing algorithms is very wide, containing both techniques that increase the samples available to us and techniques that can decrease and eliminate samples from the class with more data [2].

A factor to be also considered is also the metrics used to analyse the performance of the chosen algorithm itself, in cases where we have a remarkably high imbalance ratio it is important to know how to select the metrics to use to meet the objectives of our project. For this purpose, not only one metric is analysed in this research project.

One of the most used techniques for data balancing is SMOTE (Synthetic Minority Oversampling Technique). SMOTE is an oversampling technique where synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the samples that are close to each other.

This paper is organized as follows: Objectives, where the aims of the project are defined. State of the art that gives an overview for related works on resampling methods and boosting algorithms; in the Background we present details about the current work on the selected dataset and some outcomes of this research; The next chapter describes the methodology implemented and presents experimental results for several algorithm combinations. Concluding remarks and some perspectives are addressed in the last section.

## Objectives

In this work we propose several oversampling techniques based on SMOTE to provide a solution to a use case where we will try to compute the churn-off rate of an online bank where very few of the users cancel their account. The bank intends to implement a service that will use machine learning to obtain information on which users, depending on the characteristics of their accounts, are likely to cancel their account or stay. Clearly, the bank is interested in knowing who is leaving to offer them something to make them stay. But in the samples, they have collected there are very few users who have cancelled their accounts compared to those who continue to use the service.

Our main objective is to identify the problem and be able to find a model that is efficient and provides a function for the bank to find customers who are thinking of closing their account. it is a perfect case to check the achievements that we can get when trying to balance the data. We want to demonstrate how the use of oversampling methods combined with appropriate prediction algorithms can take us much closer to the true nature of the data.

As a secondary objective, we can include the correct analysis of the available metrics and the correct application of them. With this we will be able to know how the conventional metrics adapt to use cases like this one where ideal data condition are not given.

Another secondary objective would be the correct exploration of the data and the correct treatment of the data to be fed into the models. Thus, to be able to find the best way to operate the data we have.

## State of the Art

To fully understand the nature of the work, we will concisely address the issues involved in the execution of the project. This chapter briefly summarizes the algorithms and mathematics related to the subject matter of this document. Topics such as metrics, imbalance, oversampling and boosting will be discussed.

### Metrics

A performance metric is nothing more than a numerical value used to evaluate the model obtained. It is applied once the predictions have been obtained after training and prediction of the model.

There are many evaluation metrics for classification training [3], only the ones related to this document will be explained. Being positives and negatives the 2 classes of the binary label to predict, we will consider TP as true positives instances, TN as true negatives instances, FP as false positives instances and FN as false negatives instances.

#### *Accuracy*

Accuracy is a metric used in classification problems used to tell the percentage of accurate predictions. We calculate it by dividing the number of correct predictions by the total number of predictions:

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

**Equation 1. Accuracy Formula**

It is not recommended to use this metric in problems in which the data are not balanced since it does not consider the distribution of the classes.

***Precision***

The precision metric is used to determine what percentage of values that have been classified as positive are actually positive. As shown in the equation 2, it is calculated by dividing the true positives by all the instances classified as positive.

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

**Equation 2. Precision Formula**

***Recall***

The recall metric, also known as the true positive ratio, is used to find out how many positive values are correctly classified. It is computed by dividing the true positives by all the instances that are actually positive (False negatives + True positives):

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

**Equation 3. Recall Formula**

This metric helps us to evaluate the positive instances independently of the class distribution.

***F1-Score***

This metric combines precision and recall, to obtain a much more objective value. it is calculated as the harmonic mean of these two:

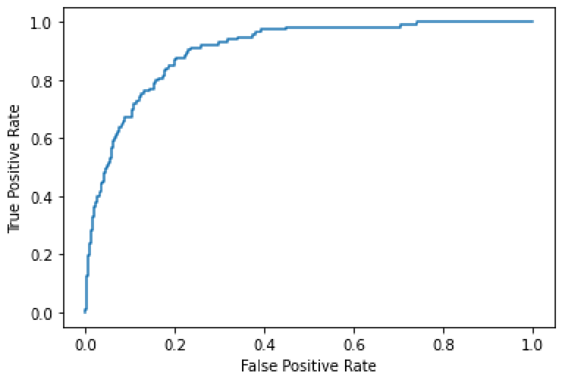
$$F1 = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

**Equation 4. F1 Formula**

This is a metric widely used in problems where the data set to be analysed is imbalanced.

***ROC Curve***

A Receiver Operating Characteristic (ROC) curve is a graph widely used to evaluate Machine Learning models for classification problems. The graph represents the percentage of true positives (True Positive Rate), also known as Recall, against the ratio of false positives (False Positive Rate). The difference with the other metrics is that in this case, the threshold at which an element is classified as 0 or 1 is modified, in order to generate all the points of the graph.



## Imbalance

A classification dataset with skewed class proportions is called imbalanced. Classes that make up a large proportion of the dataset are called majority classes. Those that make up a smaller proportion are minority classes. Imbalanced datasets exist in many real-world domains like fraud detection and in the field of disease detection [4]. The Techniques to resample the datasets are applied to improve the prediction performance.

These solutions include many different forms of re-sampling such as random oversampling with replacement, random undersampling, directed oversampling (in which no new examples are created, but the choice of samples to replace is informed rather than random), directed under-sampling (where, again, the choice of examples to eliminate is informed), oversampling with informed generation of new samples, and combinations of the above techniques.

We will apply oversampling techniques to reduce the imbalance in this project.

Oversampling can be performed by increasing the amount of minority class instances or samples creating new instances or repeating some instances [2], random over-sampling is a non-heuristic method that aims to balance class distribution through the random replication of minority class examples. In Figure 1 we could observe how we have added data to the minority class to balance both classes and thus obtain a balanced dataset.

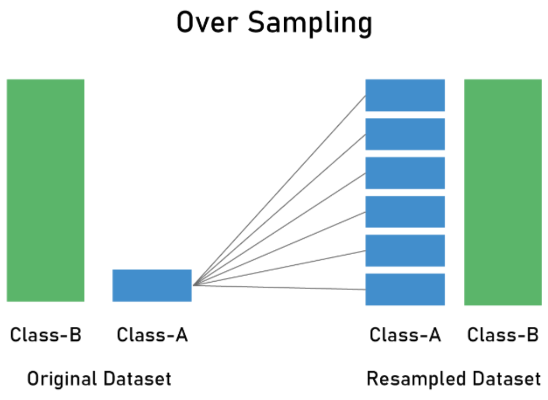


**Figure 1: Oversampling[2]**

One of the most famous techniques of oversampling is SMOTE [5], the one we will use to balance the data.

***SMOTE***

SMOTE is the preferred algorithm to perform oversampling, to introduce it we must bring up the k-nearest neighbour rule algorithm [6]. The basic idea of the algorithm is to assume that instances close to each other are more likely to belong to the same class. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbour. The only weakness or drawback of the algorithm is its high computational cost because whenever the k-nearest neighbour looks for the most similar instances, the algorithm searches through all the data set.[7].

SMOTE (Synthetic Minority Over-sampling Technique) [3] is a family of oversampling techniques that generates "synthetic" or artificial instances to balance the data sample based on the nearest neighbour rule on the minority class. The generation is performed by extrapolating new instances. For each of the minority instances, the neighbouring (nearest) minority instances are searched, and N instances are created between the line connecting the original instance and each of the neighbours. The value of N depends on the desired oversampling size. For a 200% case, for each instance of the minority class, two new generic instances must be created.

In the Figure 2 we can see the process to create a new instance:

- Step 1: choose an instance ($X_{\beta i}$).
- Step 2: select a neighbor of the minority class selected by 5 neighbors ($X_{zi}$).
- Step 2: Compute the difference between those 2 points and forms a vector.
- Step 3: Multiplies this difference by a random number between 0 and 1 and generate new instance.

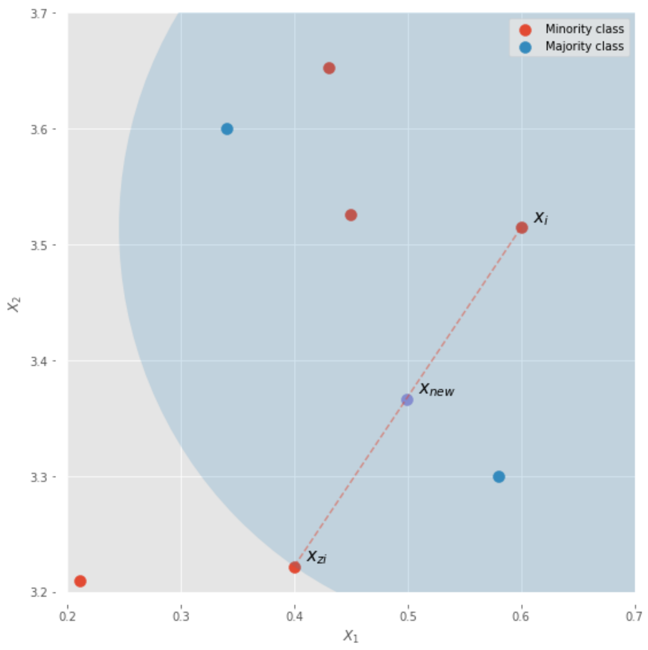


**Figure 2. SMOTE creation of a sample**

### *SMOTE variants*

Despite the substantial number of minority oversampling algorithms that exist, open-source implementations are available for only a handful of techniques. The package *smote-variants* provides a Python implementation for 85 binary oversampling techniques to boost the applications and development in the field of imbalanced learning [8].

There is a competition [9] with a ranking of algorithms in which an empirical comparison of the 85 variants of minority oversampling techniques with 104 unbalanced data sets is presented and discussed for evaluation [10].

Since we are going to use more than two features in our dataset, we will have to select those that allow multiclass oversampling.

We choose some of the best performing algorithms in the above-mentioned competition [5], such as Polynom-fit-SMOTE (first place overall), ProWSyn (second place overall) and LEE (third place overall). We will also use the SMOTE algorithm in its original variation and then choose algorithms that have worked best when the Imbalanced Rate (the percentage of imbalance) is not exceptionally large (< 9) in classification datasets such as the one we have chosen. Among the latter algorithms we select Supervised SMOTE (sixth place when IR< 9).

### *Polynom-fit-SMOTE*

Polynom-fit-SMOTE [11] refers to 4 fairly different over-sampling strategies controlled by the topology parameter of the technique. The common behaviour of using the 'bus', 'star ', 'mesh' and 'polynomic' topologies are that each of them generates instances along line segments between both near and distant samples of the minority class. These methods use samples distributed all over the space of the minority class, unlike SMOTE which uses two samples that must be relatively close to each other to generate an instance. This makes the samples have a better distribution over the whole plane than in SMOTE.

The generation of instances is computed differently:

Star topology generates instances starting from a minority sample and other nearby minority samples. It then creates a line with its neighbours and randomly places synthetic data between those lines.

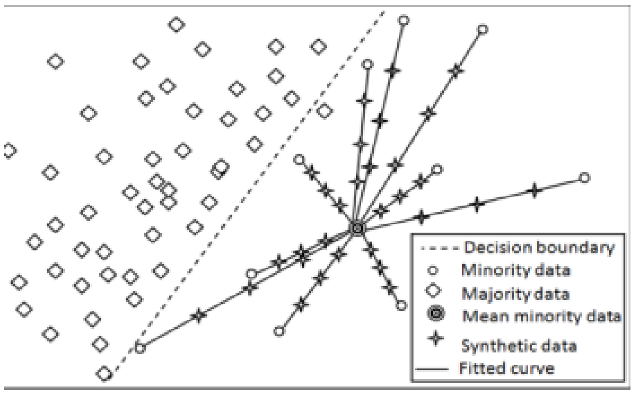


**Figure 3. Star topology [7]**

Polynomial topology finds a polynomial expression that fits the curve to include most minority samples, then it creates synthetic data along that curve.

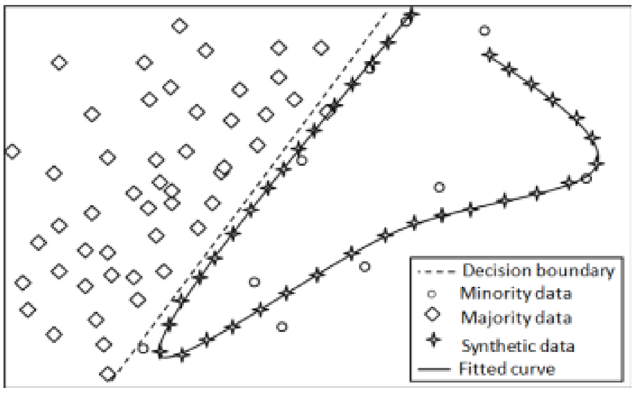


**Figure 4. Polynomial curve [7]**

Bus topology start from a single sample and generates a line from instance to instance creating a path through the samples (hence the Bus naming), then creates synthetic data on this path.

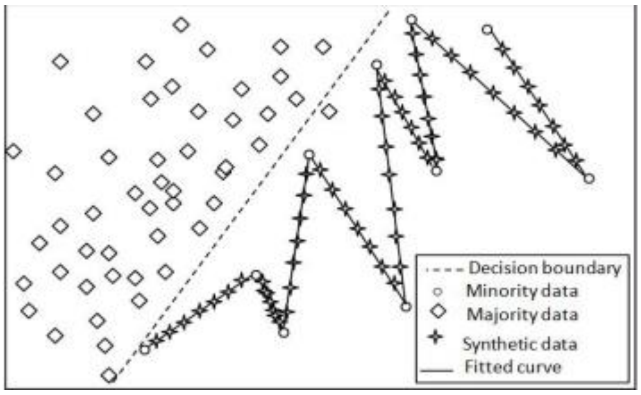


**Figure 5. Bus topology [7]**

Mesh topology creates a web between different samples of the minority class. It then randomly creates synthetic samples only on links contained in the created mesh. Computationally it is the most demanding as it needs to create one link for each pair of samples.

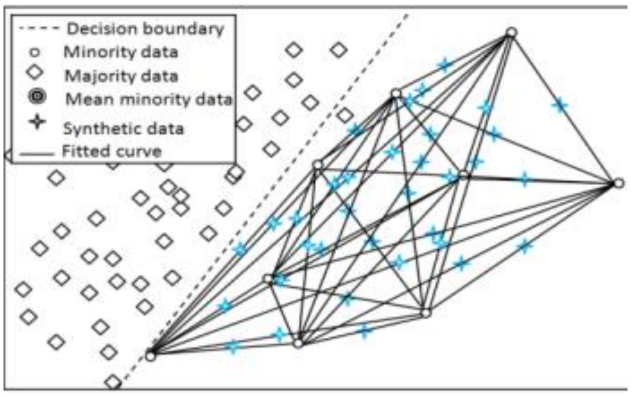


**Figure 6. Mesh topology [7]**

The number of lines generated is higher in mesh and star topologies, with 10 lines and 5 for 5 samples respectively. Covering a larger space of the set generates the best results.

### *ProWSyn*

In Proximity Weighted Synthesis (ProWSyn) [12] the number of instances generated about a minority sample is inversely proportional to their distance from the majority instances. What makes ProWSyn unique among other sampling density-based techniques is that it generates new instances by sampling the line segments between the minority instances that have similar distances to the majority instances. This property makes ProWSyn like the SMOTE technique in that samples are generated between distant minority instances, which appears to be an efficient oversampling approach, even though the assumption made about the data distribution is stronger than that of SMOTE.

### *Smote Lee*

The SMOTE method can avoid the overfitting problem by making the decision boundaries of the minority class larger and more extended in space than those of the majority class. However, SMOTE encounters the problem of overgeneralization and noise data generation. It blindly generalizes the region of a minority class without considering the majority class. This is especially problematic in the case of a highly skewed class distribution since the minority class is very sparse relative to the majority class. In this case, SMOTE results in a higher probability of class mixing.

The proposed method generates synthetic data considering their location. If the generated synthetic data are considered noise data, it is decided not to create them. With this, it is not necessary to know how many nearest neighbours should be used to create the synthetic instances. Finally, with Lee, overfitting problems and noisy data generation problems can be avoided.

Lee assigns each generated synthetic instance its rejection level before finally generating synthetic instances. Rejection level is defined as how many positive instances are located with its 5 nearest neighbours. If the value of rejection level is bigger than or equals to 3, the algorithm decides the synthetic data is appropriately generated. Otherwise, the algorithm rejects to generate the synthetic data.

### *Supervised Smote*

The goal of the supervised oversampling algorithm is to obtain a relatively balanced data set by synthesizing additional minority class samples under a supervised process. Let β>1 be the oversampling coefficient parameter, which is a scalar quantity that measures the ratio of the size of the minority class sample set after oversampling to that of the original minority class sample set. In other words, b controls how many additional minority samples will be generated. More additional minority samples will be synthesized with larger values of β.

The process is described as follows:

Step 1: Training an initial classifier model on the original training dataset.
Step 2: Synthesizing an additional minority sample with SMOTE denoted as $\mathbf{x}_{min}^{(\text{new})}$.
Step 3: The confidence of the synthesized sample is predicted using the trained initial classifier model:

$$P\left(\mathbf{x}_{min}^{(\text{new})}\right) \leftarrow \text{Predict}\left(C_{\text{model}}, \mathbf{x}_{\text{min}}^{(\text{new}}\right)$$

**Equation 5. Confidence new sample**

The validity of the synthesized sample depends on its confidence lies within the prescribed confidence interval:

$$P\left(\mathbf{x}_{\text{min}}^{(\text{new}}\right) \in \left[T_{low}, T_{\text{high}}\right]$$

**Equation 6. Confidence margin**

- Step 4: repeat step 2-3 until the $(\beta - 1) \cdot N_{min}$ valid minority class samples have been synthesized.

## Boosting algorithms

Before complex classifiers algorithms, it is important to understand what classification algorithms, or simply classifiers, are. A classifier is an algorithm that, receiving as input certain information about an object, is able to indicate the category or class to which it belongs from a limited number of possible classes.

To predict the results, we will use boosting algorithms. Robert Schapire introduced the method in 1990 [9], boosting algorithms started to develop since then, he also introduced the concept of weak learners or classifiers, computationally simple algorithms that performs relatively poor.

Boosting consists of combining the results of several weak classifiers to obtain a robust classifier. A weak classifier is defined to be a classifier which is only weakly correlated with the correct classification (it classifies better than a random classifier). On the other hand, a strong or robust learner is a classifier that performs better than a weak classifier, since its classifications are closer to the true classes.

When these weak classifiers are added, they are added in such a way that they have different weights depending on the accuracy of their predictions. After a weak classifier is added, the data changes its weight structure: cases that are misclassified gain weight and those that are correctly classified lose weight. Thus, the weak classifiers focus more strongly on the cases that were misclassified by the weak classifiers.

AdaBoost [13] is the most popular boosting algorithm and is the most relevant as it was the first formulation of an algorithm that was able to learn from weak classifiers

It's vital to an understanding of the boosting algorithms to first grasp the machine learning concepts and algorithms that AdaBoost, XGBoost or LightGBM builds upon supervised machine learning, hyperparameters, decision trees, decision stumps, Random Forest, ensemble learning, and gradient boosting.

Supervised machine learning algorithms uses algorithms to train a model to find relations in some data with label and features and then uses the trained model to predict the labels on the new data incoming or the dataset.

A hyperparameter (HP) is a parameter that is set before the learning process begins. These parameters are tuneable and can directly affect how well a model trains. Hyperparameters can have a direct impact on the training of machine learning algorithms. Thus, to achieve maximal performance, it is important to understand how to optimize them [14].

The most relevant hyperparameter optimization processes for this project are Grid Search and Random Search.

Grid search is the process of discretizing each HP and exhaustively evaluating every combination of values. Numeric HP values are usually equidistantly spaced in their box constraints. The number of distinct values per HP is called the resolution of the grid. For categorical HPs, either a subset or all possible values are considered.

Random search is the simplest form of search, values are drawn independently of each other from a pre-specified distribution for (box-constrained) numeric, integer, or categorical parameters.

Decision trees create a model that predicts the label by evaluating a tree of if-then-else true/false feature questions and estimating the minimum number of questions needed to assess the probability of making a correct decision. Decision trees can be used for classification to predict a category, or regression to predict a continuous numeric value.

For example, imagine a group of people in a room and we must guess whether they would have a risk of getting weighty. We could use their activity level, the kilocalories they eat on a normal basis, their metabolism, and their height. If a person eats too much and does no activity, the normal output its him/her to get fat. All relations we can make can be observable on this tree:

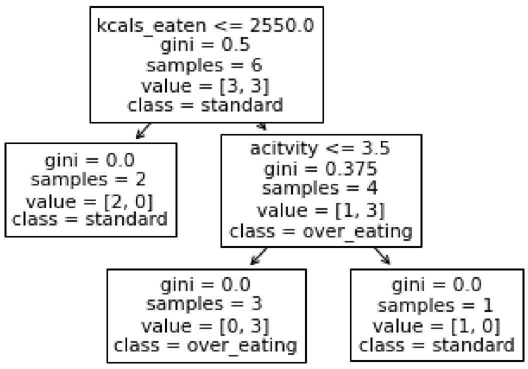


**Figure 7. Decision Tree**

Decision stump is a decision tree, which uses only a single attribute for splitting. This consist only of one interior node; these kinds of trees are way faster than normal decision trees and that's why its implementation on boosting suits perfect.

Ensemble learning combine multiple models into a new one with the objective of achieving a balance between bias and variance, thus achieving better predictions than any of the original individual models. The tree ensemble model consists of a set of classification and regression trees (CART). A CART is a bit different from decision trees, in which the leaf only contains decision values. In CART, a real score is associated with each of the leaves, which gives us richer interpretations that go beyond classification [15]. Two of the most used types of tree ensemble are:

**Bagging**: multiple models are fitted, each with a different subset of the training data. To predict, all the models that make up the aggregate participate by contributing their prediction. As a final value, the mean of all predictions (continuous variables) or the most frequent class (categorical variables) is taken. Random Forest models fall into this category.

**Boosting**: multiple simple models, called weak learners, are adjusted sequentially, so that each model learns from the errors of the previous one. In the case of Gradient Boosting Trees (GBDT), weak learners are achieved by using trees with one or a few branches. As a final value, as in bagging, the mean of all predictions (continuous variables) or the most frequent class (qualitative variables) is taken as the final value. Three of the most used boosting algorithms are AdaBoost, Gradient Boosting (XGBoost and LightGBM) and Stochastic Gradient Boosting.

Random forests[16] are a combination of tree predictors as seen in Figure 8, Each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. Instead of boosting it uses bagging, which is the process of creating a different training subset from sample training data with replacement & the final output is based on majority voting. In Random Forest algorithm the process goes:

1. N number of samples are selected from the dataset having K number.
2. Individual Decision trees are generated denoted as estimators.
3. Decision tree generates an output.
4. Majority voting is applied, and one estimator is selected.

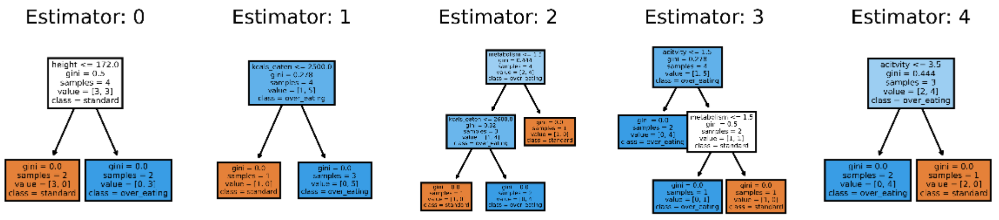

Estimator: 0
Estimator: 1
Estimator: 2
Estimator: 3
Estimator: 4
height <= 172.0 gini = 0.5 samples = 4 value = [3, 3] class = standard
gini = 0.0 samples = 2 value = [3, 0] class = standard
gini = 0.0 samples = 2 value = [0, 3] class = over_eating
kcals_eaten <= 2500.0 gini = 0.278 samples = 4 value = [1, 5] class = over_eating
gini = 0.0 samples = 1 value = [1, 0] class = standard
gini = 0.0 samples = 3 value = [0, 5] class = over_eating
metabolism <= 1.5 gini = 0.444 samples = 4 value = [2, 4] class = over_eating
kcals_eaten <= 2600.0 gini = 0.32 samples = 3 value = [1, 4] class = over_eating
gini = 0.0 samples = 1 value = [1, 0] class = standard
acitvity <= 1.5 gini = 0.278 samples = 4 value = [1, 5] class = over_eating
gini = 0.0 samples = 2 value = [0, 4] class = over_eating
metabolism <= 1.5 gini = 0.5 samples = 2 value = [1, 1] class = standard
gini = 0.0 samples = 1 value = [0, 1] class = over_eating
gini = 0.0 samples = 1 value = [1, 0] class = standard
acitvity <= 3.5 gini = 0.444 samples = 3 value = [2, 4] class = over_eating
gini = 0.0 samples = 2 value = [0, 4] class = over_eating
gini = 0.0 samples = 1 value = [2, 0] class = standard

**Figure 8. Random Forest**

***AdaBoost***

AdaBoost works with Decision Stumps [17] . Decision Stumps are like trees in a Random Forest, but not "fully grown." They have one node and two leaves. AdaBoost uses a forest of these stumps.

Stumps alone are not an effective way to make decisions. A full-grown tree combines the decisions from all variables to predict the target value. A stump, on the other hand, can only use one variable to decide [18].

It follows these steps:

1. A weak classifier is made on top of the training data based on the weighted samples. Here, the weights of each sample indicate how important it is to be correctly classified. Initially, for the first stump, we give all the samples equal weights.
2. We create a decision stump for each variable and see how well each stump classifies samples to their target classes.
3. More weight is assigned to the incorrectly classified samples so that they are classified correctly in the next decision stump. Weight is also assigned to each classifier based on the accuracy of the classifier, which means a high accuracy will end in a high weight.
4. Reiterate from Step 2 until all the data points have been correctly classified, or the maximum iteration level has been reached.

***XGBoost***

XGBoost [19] , which stands for Extreme Gradient Boosting, is a scalable and highly accurate implementation of gradient boosting that pushes the limits of computing power for boosted tree algorithms, being built for energizing machine learning model performance and computational speed. It is based on the gradient boosting algorithm [20] and dominates structured or tabular datasets on classification and regression predictive modelling problems and that is why it was selected.

Gradient Boosting [20] is a generalization of the AdaBoost algorithm that allows the use of any cost function, as long as it is differentiable. The flexibility of this algorithm has made it possible to apply boosting to a multitude of problems (regression, multiple classification...) making it one of the most successful machine learning methods. Although there are several adaptations, the general idea of all of them is the same: to train models sequentially, so that each model adjusts the residuals (errors) of the previous models. It can be used for regression and for classification.

Gradient boosting classifier follows these steps

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$. Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{\arg\ \mathrm{m}} \sum_{i=1}^{n} L(y_i, \gamma).$$

**Equation 7. initial model**

2 For $m = 1$ to $M$ :

2. Compute so-called pseudo-residuals:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

**Equation 8. residuals**

3. Fit a base learner (or weak learner, e.g., tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e., train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
4. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:

$$\gamma_m = \underset{\gamma}{\arg}\ \text{m} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

**Equation 9. error multiplier**

5. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

**Equation 10. Updated model**

6. Output $F_M(x)$.

In gradient boosting regularization is usually ignored. This was because the traditional treatment of tree learning only emphasized improving impurity, while the complexity control was left to heuristic. XGBoost uses regularization methods (highlighted) that penalize various parts of the algorithm and generally improve the performance of the algorithm by reducing overfitting. XGBoost uses the Taylor expansion of the loss function up to the second order:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} \left[ l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \omega(f_t) + \text{ constant}$$

**Equation 11. Object function XGBoost**

Where the $g_i$ and $h_i$ are defined as:

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right)$$
$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right)$$

**Equation 12. Taylor terms**

After we remove all the constants, the specific objective at step $t$ becomes:

$$\sum_{i=1}^{n} \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \omega(f_t)$$

**Equation 13. Objective function at step t**

In XGboost regularization gains importance and the model complexity $\omega(f)$ is defined formally:

$$\omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

**Equation 14. Model complexity in XGBoost**

Finally, if we put together those terms and derivate, we obtain the objective function to minimize:

$$\text{obj}^* = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

**Equation 15. Objective function XGBoost**

***Light Gradient Boosting Machine***

In AdaBoost, the sample weight serves as a good indicator for the importance of data instances. However, there are no native sample weights, and thus the sampling methods proposed for AdaBoost cannot be directly applied. Fortunately, we notice that the gradient for each data instance in GBDT (Gradient Boosting Decision Trees) provides us with useful information for data sampling. That is, if an instance is associated with a small gradient, the training error for this instance is small and it is already well-trained [21].

Light Gradient Boosting Machine, or LGBM, is based on 2 new ideas: GOSS and EFB [22]:

Gradient-based One-Side Sampling, or GOSS, is a modification to the gradient boosting method that focuses attention on those training examples that result in a larger gradient, in turn speeding up learning and reducing the computational complexity of the method.

Exclusive Feature Bundling, or EFB, is an approach for bundling sparse (mostly zero) mutually exclusive features, such as categorical variable inputs that have been one-hot encoded. As such, it is a type of automatic feature selection that reduces dimensionality of the train data.

Combination of both changes speed up de process up to 20 times, that is what makes LGBM a suitable option when you want to accelerate the training process.

## Other mentioned algorithms

There are other algorithms that are mentioned throughout the reading of the document; therefore, we will give a small description in those topics.

In the boosting family we also include CatBoost that introduces in CatBoost are the implementation of ordered boosting, a permutation-driven alternative to the classic algorithm, and an innovative algorithm for processing categorical features.

Other famous and recognized algorithm is an Artificial Neural Network (ANN). It tends to simulate how humans' neuronal systems works. Although, the first neural network model evidenced was created by Warren McCulloch and Walter Pitts in 1943 [23], the first recognized as an ANN applied to real world problems was developed to recognize binary patterns so that if it was reading streaming bits from a phone line, it could predict the next bit[24].

The processing units are organized in layers. There are typically three parts to a neural network: an input layer, with units representing the input fields; one or more hidden layers; and an output layer, with a unit or units representing the target field or fields. The units relate to the variable through connection strengths (or weights). The input data are presented in the first layer, and the values are propagated from each neuron to each neuron in the next layer. at the end, a result is sent from the output layer.

Support Vector machine is widely used for classification and regression. In SVM, two parallel hyperplanes are constructed on each side of the hyperplane that separate the data as seen in the figure below with class A and Class B. The separating hyperplane is the hyperplane that maximize the distance between the two parallel hyperplanes. An assumption is made that the larger the margin or distance between these parallel hyperplanes the better the generalization error of the classifier will be [25].

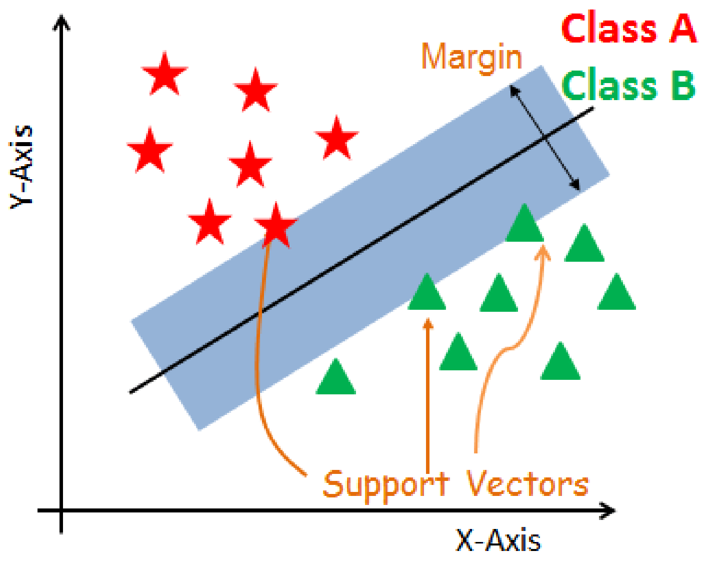


**Figure 9. SVM algorithm [26]**

Logistic regression (LR) is one of the most expressive and versatile statistical tools available for data analysis. Its origin dates to the 1960s [27], its use has been universalized and expanded since the early 1980s, especially due to the computer facilities available since then.

The purpose of the LR is to determine the probability of occurrence of an event in question as a function of certain variables that are presumed to be relevant or influential. Therefore, the LR consists of obtaining a logistic function of the independent variables that allows classifying individuals into one of the two subpopulations or groups established by the two values of the dependent variable.

The logistic function is that which finds, for each individual according to the values of a series of variables (Xi), the probability (p) that he/she presents the effect studied. A logarithmic transformation of this equation, called logit, consists of converting the probability (p) into odds.

This gives rise to the logistic regression equation, which is like the multiple linear regression equation [28].

## Background

Our selected contains information about a bank's customers and whether they have stopped using the service. Based upon data of clients we calculate whether they stand a chance to close their bank account or not. The main problem of the dataset is the imbalanced distribution of the classes. Having 3 instances of a customer that has stayed for 1 that has left throughout the dataset.

We have made an intensive study of the top 30 projects [29]–[58] (based on scoring by other users out of a total of 96) that have worked on it and have proposed diverse ways to solve the imbalance problem both in the treatment of the data and in the algorithms chosen. This will help us to place our results in a competitive framework in order to analyse how successful they are.

We assume competition as the subset of selected projects since the size of all the notebooks makes it very difficult to cover them all. Even so, these projects are the ones that receive the best votes from users and are the benchmarks when it comes to this dataset.

Both the projects and the dataset have been obtained from Kaggle and it contains information about churn-off rate of an online bank account.

Do not forget that we also want to know how well our dataset is related to the use of SMOTE, so we will also test its use in all these projects.

Table 12 shows the algorithms employed by the different users for the most optimal resolution of the problem. Apart from these algorithms, an identifier ID of each project is also included.

The third column represents the metrics used for tracking and performance of each algorithm within the project.

The fourth column identifies the algorithm chosen by the author as the best based on either the title of the notebook or the conclusion of the notebook itself. With this chosen model (*BestModel*) we can observe the values of its performance of the metrics used in the fifth column, *Best Values*. *Best Values* represents for the author the best values he can obtain to solve the problem, using the best model he can do with the *BestModel* algorithm.

The sixth column, *MetricsGoal*, shows the metric on which the author has based the title of the project or which he has concluded is the one that can most closely approximate the nature of the problem.

Finally, the seventh column considers whether SMOTE has been employed to improve the values of these metrics. F represents False and T represents True.

| ID | Algorithms | Metrics | Best Model | Best Values | MetricGoal | Smote |
|---|---|---|---|---|---|---|
| 1 | Logistic Reg, SVC, Random Forest, XGB | Accuracy, Recall, Precision, F1 | XGB | 0.63, 0.88, 0.58, 0.73 | Recall | T |
| 2 | XGB | Accuracy | XGB | 0.87 | Accuracy | T |
| 3 | KNN, Gaussian Naive Bayes, Decision tree, SVC, Random Forest, AdaBoost, XGB | Accuracy, Precision | XGB | 0.86, 0.88 | Accuracy | T |
| 4 | Logistic Reg, Decision Tree, | F1 | XGB | 0.81 | F1 | F |

| | | | | | | |
|---|---|---|---|---|---|---|
| | Random Forest, Extra Trees, XGB, ANN | | | | | |
| **5** | Logistic Reg, SVC, KNN, Gaussian Naive Bayes, Gradient Boosting, Random Forest | Accuracy | Gradient Boosting | 0.87 | Accuracy | F |
| **6** | KNN, Gradient Boosting, Random Forest, ANN | Accuracy, Recall, Precision, F1 | Random Forest | 0.67, 0.85, 0.86, 0.87 | Accuracy | F |
| **7** | ANN | Accuracy, Precision, Recall, F1, ROC | ANN | 0.80, 0.51, 0.75, 0.61, 0.78 | Accuracy | T |
| **8** | ANN | Accuracy | ANN | 0.866 | Accuracy | F |
| **9** | ANN | Accuracy, Precision, Recall, F1 | ANN | 0.86, 0.75, 0.76, 0.76 | Accuracy | F |
| **10** | ANN | Accuracy | ANN | 0.86 | Accuracy | F |
| **11** | KNN, Logistic Reg, SVC, Random Forest, XGB | Accuracy | Random Forest | 0.86 | Accuracy | F |
| **12** | ANN | Accuracy | ANN | 0.85 | Accuracy | F |
| **13** | Logistic Reg, Decision Tree, Gaussian Naive Baye, Random Forest, Extra Trees, XGB, ANN, MLP, CatBoost | Accuracy | CatBoost | 0.86 | Accuracy | F |
| **14** | ANN | Accuracy | ANN | 0.86 | Accuracy | F |
| **15** | ANN | Accuracy | ANN | 0.86 | Accuracy | F |
| **16** | XGB | Accuracy | XGB | 0.86 | Accuracy | F |
| **17** | ANN | Accuracy | ANN | 0.84 | Accuracy | F |
| **18** | ANN | Accuracy | ANN | 0.86 | Accuracy | F |
| **19** | SVC, ANN | Accuracy | ANN | 0.85 | Accuracy | F |
| **20** | ANN | Accuracy, Recall, Precision, F1 | ANN | 0.80, 0.69, 0.7, 0.69 | Accuracy | F |
| **21** | KNN, Logistic Reg, XGB | Accuracy, Recall, Precision, F1 | XGB | 0.81, 0,71, 0.57, 0,58 | Accuracy | F |
| **22** | Logistic Reg, SVC, Random Forest, ANN | Accuracy, Recall, Precision, F1 | Random Forest | 0.86, 0,71, 0.81, 0,75 | Accuracy | F |
| **23** | Random Forest, ANN | Accuracy, Recall, Precision, F1 | Random Forest | 0.85, 0,72, 0.78, 0,74 | Accuracy | F |
| **24** | ANN | Accuracy, Recall, Precision, F1 | ANN | 0.85, 0.35, 0.82, 0.45 | Accuracy | F |
| **25** | KNN, Random Forest | Accuracy | Random Forest | 0.87 | Accuracy | F |
| **26** | Logistic Reg, XGB, LGBM | Accuracy | XGB | 0.86 | Accuracy | F |

| | | | | | | |
|---|---|---|---|---|---|---|
| **27** | Logistic Reg, Decision Tree, Random Forest | Accuracy, Recall, Precision, F1 | Decision Tree | 0.85, 0.34, 0.81, 0.66 | Accuracy | F |
| **28** | Logistic Reg, Boosted Logistic Reg, Gradient Boosting | F1 | Gradient Boosting | 0.59 | F1 | F |
| **29** | Random Forest, ANN | Accuracy | Random Forest | 0.86 | Accuracy | F |
| **30** | ANN | Accuracy | ANN | 0.79 | Accuracy | F |

**Table 1. Study of other projects on our dataset [29]–[58]**

At first glance, we can already see from the table that the combination of accuracy, precision, recall, F1 is by far the most used. This is because these metrics are closely related to each other and on many occasions the same way of obtaining them has been used, by means of the scikit-learn library function, classification report[59].

This function allows, with the prediction and the actual values as input, to provide a matrix containing these values for each class.

It also calculates the overall of both by counting the distribution of the classes (weighted average).

Also noteworthy is the emergence of ANNs (Artificial Neural Network), in many cases with no other algorithm to compete with.

For a more comprehensive analysis, we will visualize the results in a more graphical and compensable way. In this way, we can also obtain minute details that are not so easy to identify in a table.

This project will be carried out in a *Jupyter* notebook [60] , using the Python language [61] with the *Pandas* framework [62], using the *seaborn* library [63] for the visualizations.

To begin with, we will visualize a count (Y-axis) of the metrics (X-axis) used throughout this study (Figure 10), it is evident the supremacy in the choice of accuracy over other metrics, being present in 28/30 datasets.

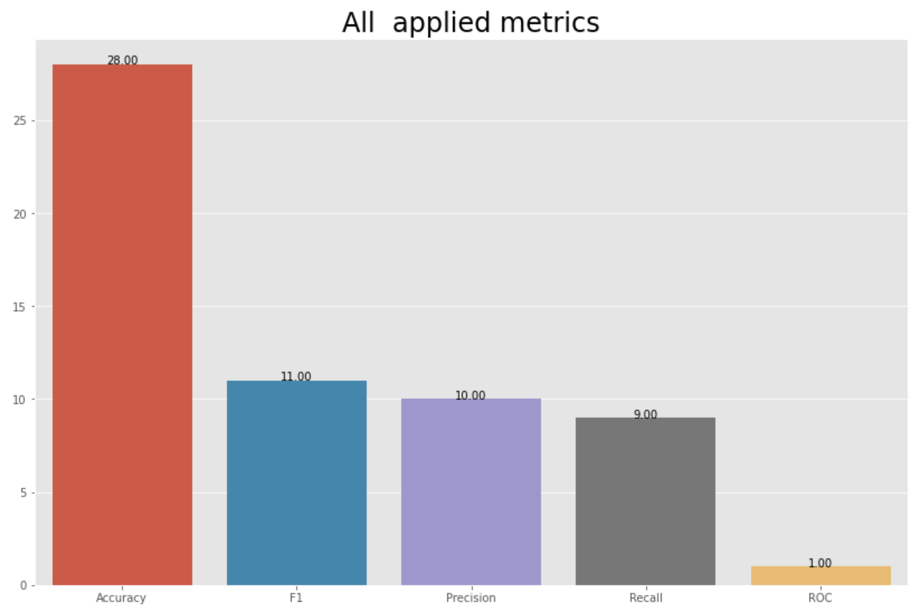


**Figure 10. Used metrics in the competition**

Accompanying this fact, when choosing a metric as a guide for model evaluation, we can observe in Figure 12 how the metric chosen to represent and choose the optimal model is accuracy in 9 out of 10 cases.

This makes sense, since there are many cases where only the accuracy metric is present, either because of ignorance of other metrics or a lack of willingness to implement them.

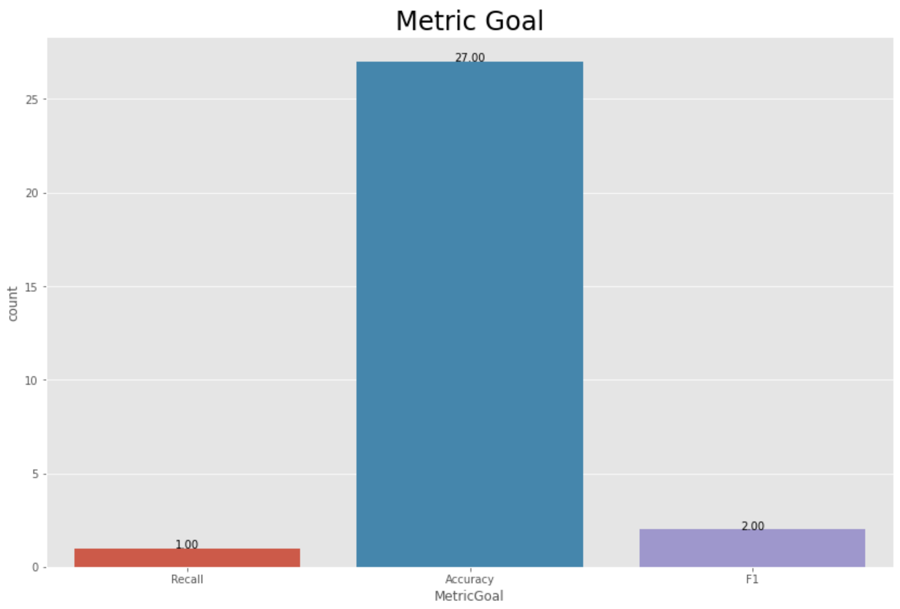


**Figure 11. Metric goal in the competition**

Looking deeper into this metric, in Figure 12, we have represented the values of each project. It is worth mentioning that acceptable accuracy values depend on the problem being solved, but a value between 0.7 - 1 would be within the usual range.

The Y-axis is composed of values from 0 to 1, value 0.7 means that we have correctly guessed 7 out of 10 instances of the dataset. The X-axis is composed of the Ids of the projects. We see that the values lie at a mean of 0.86, a considerably high value in a general case.

But this metric loses value when the data source used is imbalanced, so obtaining high values as in this case does not always ensure an optimal case without a correct accompanying metric.

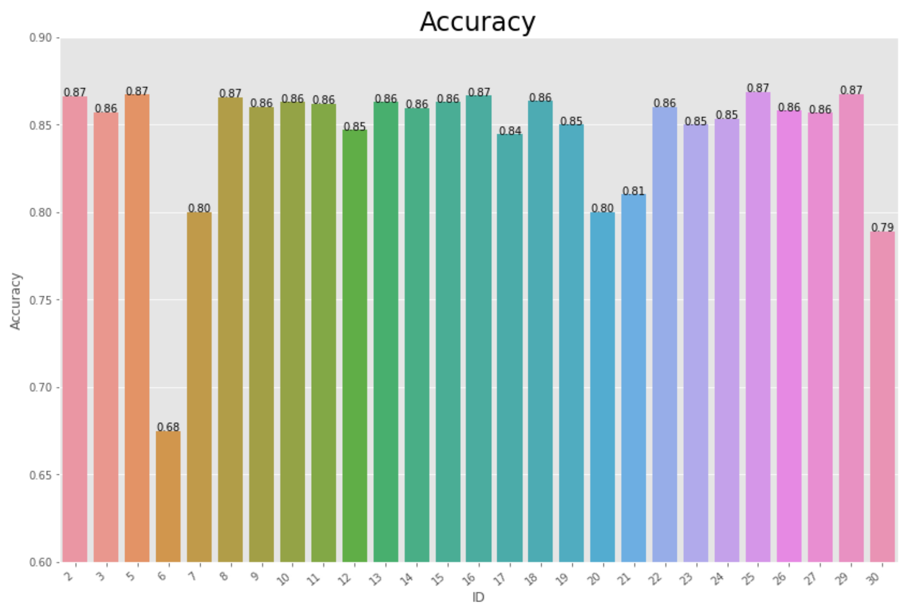


**Figure 12. Accuracy of each project in the competition**

Then we move to the analysis of all the algorithms used, we highlight the use of many Artificial Neuronal Networks (19 / 30) followed by Random Forest.

This graph does not catch us by surprise, the name, and the relevance that artificial neural networks have in machine learning is especially high, many times other supervised learning algorithms go unnoticed in these types of cases where a classifier could get a better result.

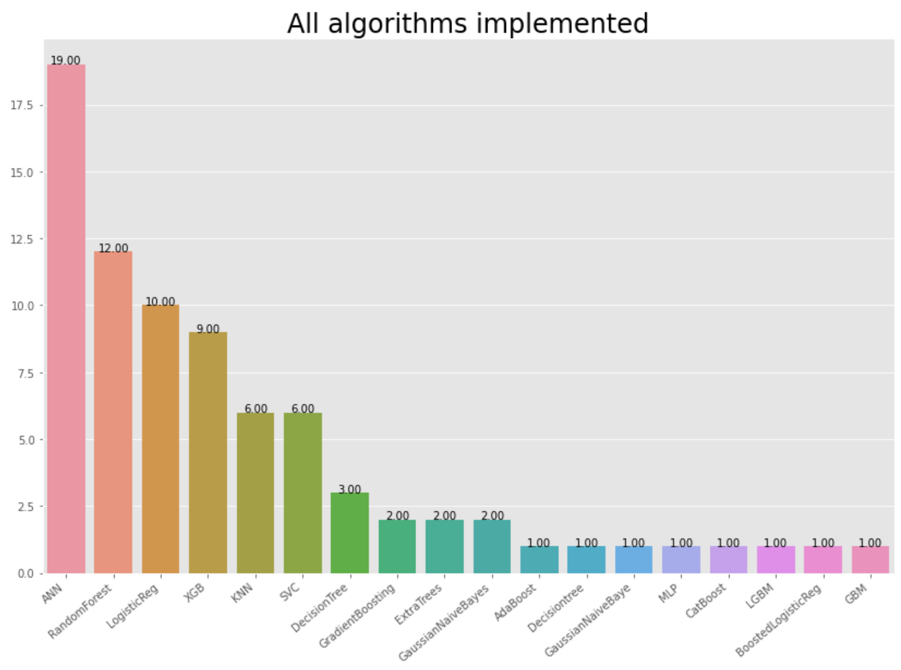


**Figure 13. All algorithms used in the competition**

Following these results of chosen models, we now look at the percentage of choice when there is competition (when more than one algorithm is used), in Figure 14 we can see in percentage (Y-axis) the choice ratio of each algorithm when one or more algorithms (X-axis) are competing with the same metrics.

Comparing Figure 14 with Figure 13 we can deduce that algorithms such as Gradient Boosting and CatBoost have been chosen but have been used very few times.

Perhaps most suggestive is to discover how little use the ANNs have. We see in Figure 15 that it has only a 14.29% selection rate while XGBoost obtains a remarkable 71.43%.

The drop in ANNs may suggest that users with more knowledge of other algorithms are leaning towards these in cases of imbalance.

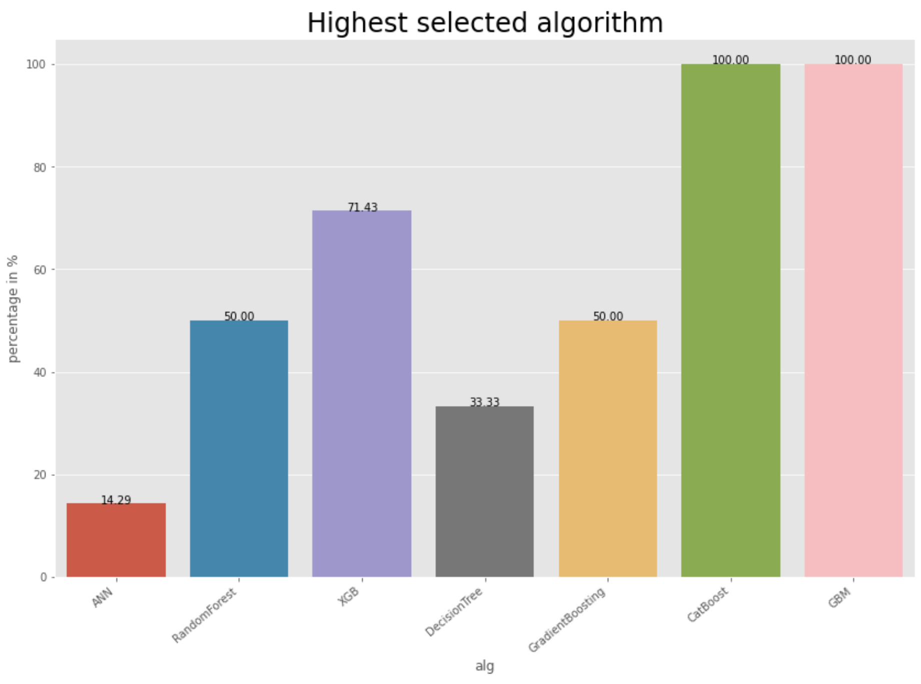


**Figure 14. Algorithms ' selection rate in the competition**

Smote is hardly an option in the entire competition, few people use any kind of oversampler, it may be a lack of knowledge or that they did not see the convenience of it, but only 4/30 have used Smote.

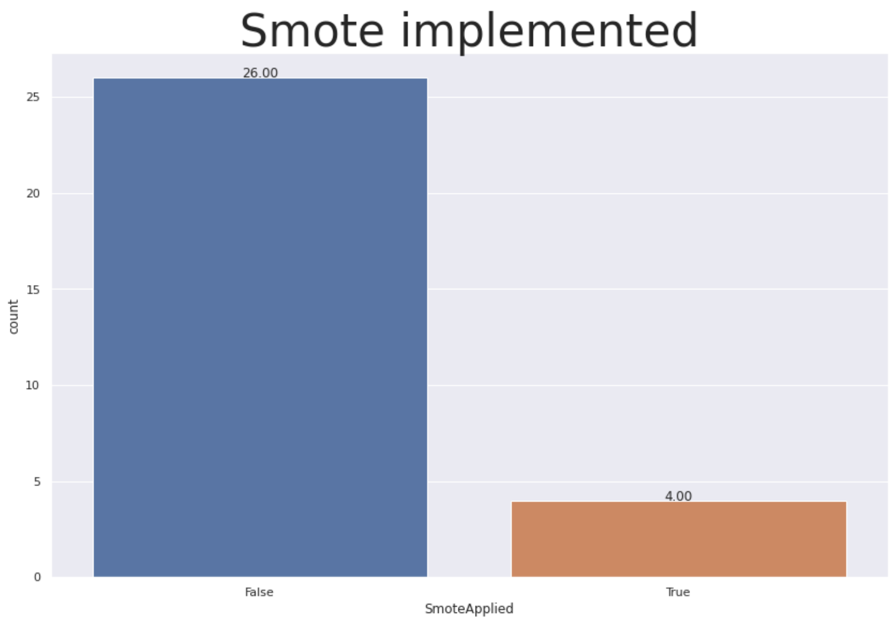


**Figure 15. SMOTE implementation in the competition**

# Methodology

## Data source

### *Kaggle and its competitions*

To achieve a good result, we need a trustable source of data. For this, we will draw on one of the largest databases on the Internet, Kaggle [64]. It is a web platform that gathers the largest Data Science community in the world, with more than 536 thousand active members in 194 countries, receiving more than 150 thousand publications per month. There are more than 50 thousand public datasets and 400 thousand public projects available to everyone. This platform allows data specialists and other developers to participate in Machine Learning competitions and data challenges, write and share code and save datasets.

Competitions and challenges are the backbone of the platform. They allow us to challenge ourselves by taking as a reference the rankings to measure our performance and use them as motivation [65].

### *Dataset*

Our dataset has 91 notebooks and 96 unique contributors. It is an unbalanced dataset containing information about a bank's customers and whether they have stopped using the service. Based upon data of clients we calculate whether they stand a chance to close their bank account or not.

This is known as Churn Rate, the possibility that a customer will or will not exit a business.

#### *Columns*

We are now evaluating and explaining all the variables present in this dataset [33], we first begin with the sensitive and personal information it contains, in this group we can classify these 5:

1. CustomerId: Client unique ID inside the bank.
2. Surname: Client’s surname.
3. Geography: Which Country the client belongs to.
4. Age: Age of the client.
5. Gender: Gender of the client.

It also contains a unique value in the dataset being this one Row Number, is no more than an ascending number showing the row number. NumOfProducts which define the number of assets the client has.

CreditScore is a variable created by the bank that defines how likely it is for the client to get credit. Tenure is the time of bond client-company.

The last variable that comes into play is Balance, this value expresses the money left at the time they exited the company. This is in fact the label of the dataset, value to be predicted is exited, which consist of a binary assignation: 0 no exited. 1 for exited clients.

## Procedure

All the code belongs in a jupyter python notebook. The processes on data preparation, data splitting, data oversampling and predicting stage have been used in this project. *Pandas* have been used as the main package for manipulating data. *Sklearn* package has been selected for the prediction and data splitting stage. Other libraries such as *matplotlib* and *numpy* are of less relevance in the project aswell. The stand-out package is *smote_variants*, which hosts many Smote variants for processing training data. We have used it to oversample the chosen Smote variants on our data.

We have followed a classic approach when treating the data for feeding it into the algorithms of predictions.

We have first check whether there's error with the data like null values or missing data, then a small visualization of all the fields contained in the dataset to notice any feature at first sight. It is also important to reject any sensible data from the dataset like ids or surnames.

Imbalance is the main matter of this document, so we have detected whether there's imbalance or not in our dataset.

Feature engineering has been applied in the next section, with categorical and numerical data receiving transformation to let the models have an easier job finding patterns.

A normal split into training data and test data has been applied to the cleaned dataset followed by the oversampling stage, where we have used the s*mote_variants* library on python to select the oversampling methods presented before and applying these methods to the train data. It is crucial we do not oversample our test data, because this data could be treated as unseen data by the model and so the oversampling techniques could falsify the data.

Then the prediction stage where we have 3 different boosting algorithms: AdaBoost, XGBoost and LGBM, where we have run all our different oversampled train sets and obtained conclusions.

### *Data preparation*

#### *Missing values*

The first thing we are going to do is check whether this dataset contains null or missing values. For that we would use native functions of Pandas library *isnull,* and the do the summatory of nulls. It is crucial to detect nulls values since most machine learning algorithms do not tolerate null values.

It is important as well to check whether there is any duplicated row, duplicated rows contain entirely the same information, it is important to remove those since the model could treat both rows as different samples and then it could spoil the algorithm selection protocol.

Humans perceive things best through graphic representation. Therefore, to identify any variable that contains something strange or that we can see that it does not maintain any relationship, we show each feature.

### *Imbalance*

We check the imbalance, we will divide majority class / (majority class + minority class), if this value exceeds 0.75, we could consider that our dataset is imbalance. This metric is the IR, it is the most used to determine imbalance in a data source, it could be misleading when there is more than one class since it is not our case, we are safe to use the imbalance rate.

The computed value is 0.7963 ~ 0.8, so our dataset is imbalanced.

### *Sensible data treatment*

The first step with features treatment is to delete all sensible data, this is, any information relative to the client of the bank, in this case we would remove the *CustomerId* and its surname. We can also remove the *RowNumber* column since it does not add any value to the data.ç

### *Categorical data treatment*

To understand why we will change all categorical data to numbers, first we must think like a machine, machines can only compute the numbers, it could not manage text, everything is translated to numbers and then it operates with it. Machine Learning algorithms fit into the same case.

There are different approaches to transform categorical into numerical values, the most used techniques are One-Hot Encoding and Label Encoding.

Label Encoding creates a set of values, for example if we had 3 values for a column, we would have 0, 1, 2. This is a good approach when we have only 2 classes in a column, in our case, the gender column, though we will use label encoder for this.

One-hot-encoding makes our training data more useful and expressive and can be easily rescaled. One-hot-encoding is done by creating a column with true (1) or false (0) for each value of the column to which a hot encoding is applied. So, if we have 3 text values, we will create 3 columns with 0 if the row is not the value and 1 if it is the sample value.

We would choose one hot encoding since we have few values in each column.

pandas 'function *get_dummies* apply one hot encoding to the categorical columns.

### *Data normalization*

Normalization is a data preparation technique that is frequently used in machine learning. The process of transforming the columns in a dataset to the same scale is referred to as normalization.

We would use the *Min-Max Scaler*, which consists of subtracting the minimum value of the columns to the highest and dividing it by the range. The result column will have a minimum value of 0 and a maximum value of 1. All the values will fluctuate between those 2 values.

Normalization does not need to be applied to all datasets with numeric columns, in this case we have values in an extremely high range. We have, for example, exceedingly small values for the variable Tenure and variables such as Estimated Salary and balance that comprise up to 6-digit numbers. In these cases, it is best to apply data correction so that our algorithm can find more similarities.

Normalization can also be of major help when the techniques employed do not make assumptions about the distribution of your data.

From *Sklearn* we will use the *MinMaxScaler* function on: *CreditScore* , *Age*, *Balance*, *Tenure* and *EstimatedSalary*.

### *Merging*

We have now split into those 2 types of data, categorical and numerical, we then use *pandas'* function *concat* to join those columns together and get into the data splitting phase.

### ***Data splitting***

Now we are going to split the train data, called *X* in the data science world and the target data, the feature to predict, commonly named *y*.

We would split the data as well into train data for the algorithms to learn and test which contains the unseen data from the algorithm and would let us judge the algorithm performance.

80% of data will go into training and the 20% rest we will save for predictions.

From *Sklearn* we will use the *train_test_split* function to achieve this.

### ***Oversampling Stage***

As we have mentioned before, the dataset presents imbalance. For the oversampling techniques we are going to use SMOTE and its variations.

First, we get the list of algorithms that support multiclass oversampling, since we have more than one feature, what *smote_variants* do is oversampling each pair of features and create the same number of rows.

We will call the Smote Variants package *sv* from now on.

With the function *sv.get_all_oversamplers_multiclass()* we get a list of the oversamplers we could use on more than one feature.

We check that our selected variants are on that list.

For all the variations we will create a train data oversampled with that variation with the function *sv.MultiClassOversampling(oversampler)*. We will call this function once per oversampler and we store the result *X_train* and *y_train* sampled with that technique.

***Prediction stage***

We are going to use 3 different algorithms to produce estimations on what the label is going to be, first we are going to start with the most known Adaboost to go into more complex models like LightGBM and xGboost to see if we can achieve a better performance.

Grid Search and Random Search has been used to find the best HPs and have the best outcome possible. Some important hyperparameters are[14], [66]–[68]:

- Learning rate: This determines the impact of each tree on the outcome. Boosting algorithms work by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates. Range goes from 0 to 1.
- N estimators: number of sequential trees created on the boosting process. This HP is always an integer value, range is not defined but typical values don’t pass 100.
- Base estimator: The tree used. It is usually a Gradient Boosting Tree for XGBoost and a Decision tree for AdaBoost.
- Max depth: The maximum depth of a tree. Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
- Num leaves: maximum number of leaves, theoretically, we can set $num\ leaves = Max\ depth^2$ but its better to select a lower value.
- Subsample: It denotes the fraction of observations to be randomly samples for each tree. Subsample ratio of the training instances. Setting it to 0.5 means that they would randomly sample half of the training data prior to growing trees. - This will prevent overfitting. Subsampling will occur once in every boosting iteration. Lower values make the algorithm more conservative and prevents overfitting, but too small values might lead to under-fitting.
- Min data in leaf specifies the minimum number of observations that fit the decision criteria in a leaf. Setting it to 100,
- Gamma (γ): Minimun loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.

### *AdaBoost*

For AdaBoost we consider 3 steps taken into the optimization since it takes tame to create an AdaBoost model, we search 3 different HP:

1. First, we search for a n estimators value from 10 to 7000 in 3 intervals, we obtain the best value at 5000.
2. Then we find a value of 0.001 for learning rate, searching from this value to 1
3. We try to find the best base estimator changing max depth value, from 10 to 25 in 3 intervals, finding 20 as the best value in a decision tree classifier.

### *XGBoost*

In XGBoost we only do one big Grid Search with these HP involved:

- Learning rate from 0.15 to 0.5
- Max depth from 4 to 8
- Gamma from 0.2 to 0.5
- Colsample bytree from 0.45 to 0.6

We obtain a model with learning rate :0.025 so we use 1000 estimator to create the model.

### *LightGBM*

In LightGBM since it's the faster of the tree we search on a big space with Optuna [14], [69] an open source hyperparameter optimization framework that performs Random Search given a grid and an objective function, in our case the function was no other than the negative recall score so it would minimize it. The HP optimized are n estimators, learning rate, num leaves, max depth, subsample min data in leaf.

## Findings

### Own Experiment outcomes

The first thing we must reflect on the results is whether the experiment has occurred as we expected. To a certain extent, yes, we have been able to attribute those problems we have in identifying critical cases in minority classes.

The pretext for the success of the experiment is that, when we do not oversample the data, our Recall model metrics are of very low quality and therefore we are leaving many customers who are likely to leave our bank without doing anything.

Our results are reflected in table 2, that gathers all data regarding the models and its evaluation. It contains the prediction algorithm together with the different oversamplers techniques applied, each combination of those 2 groups throws results that we evaluate in the 3 metrics previously explained: Accuracy, Recall, F1-Score.To picture these values we have some figures to be discussed.

Figure 16 shows the results of the project in terms of accuracy. We do not get a very high accuracy value, but we stay in an acceptable range in most cases. It is normal to lose a little bit of accuracy as we have forced so many algorithms to look at the minority class.

Higher values could have been obtained if we would have focused on this metric, but it was not the case for us, so we are happy with the outcome. Supervised-Smote have obtained a good score as we have predicted that it could work better in a dataset with low imbalance rate.

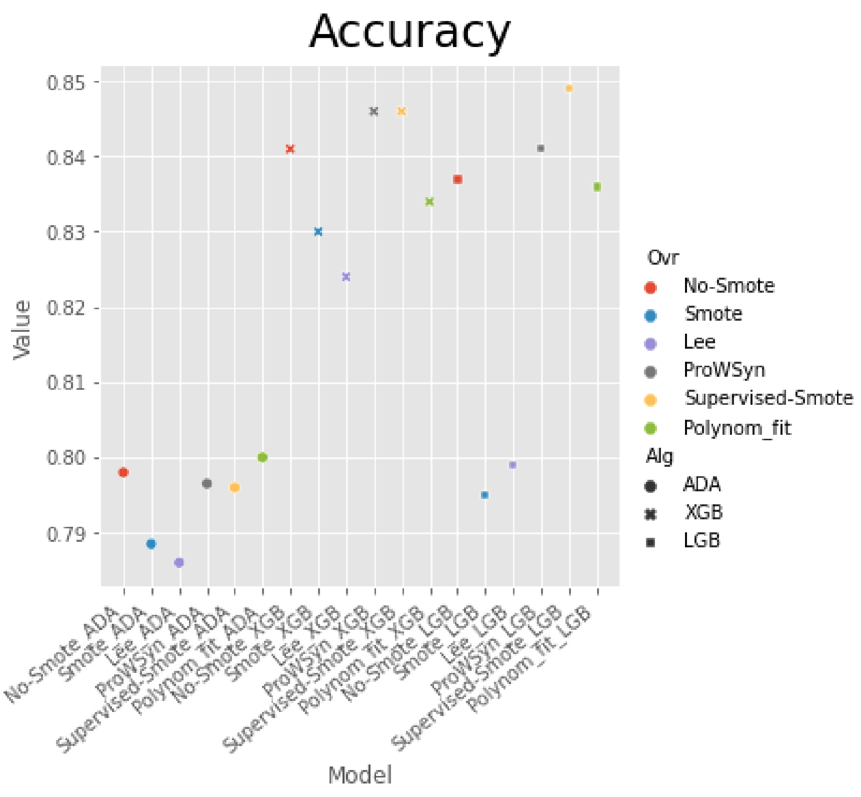


**Figure 16. Accuracy Results**

In Figure 17 we can see the F1 results. As this is a composition of both precision and recall, we obtain values that can be improved but in a correct range, the best cases are close to 0.65, an excellent F1 value with an imbalanced dataset.

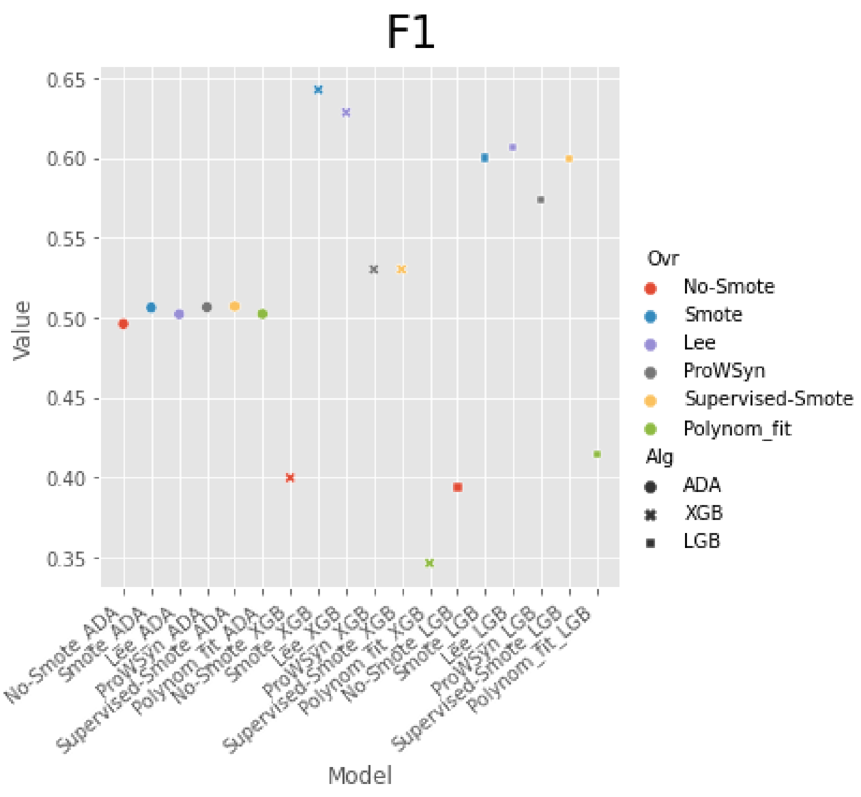
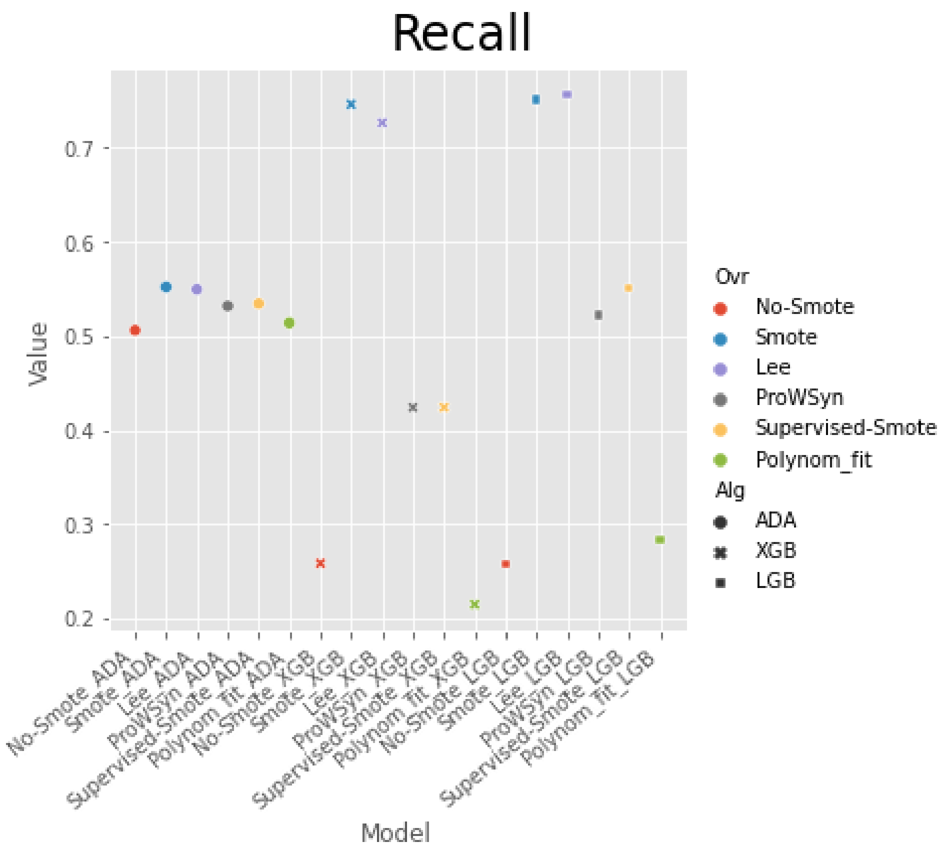


**Figure 17. F1 Score results**

To continue with the metrics, in Figure 18 we have the results that are perhaps of most interest to us, namely the project recall, as the f1 score is directly related to this, in the two models that have performed best. These are SMOTE and Lee-SMOTE. Obtaining Recall values close to 0.8 is great news compared to the worst conditions without SMOTE where we can see that we went from 0.25 to 0.76.

This means that we are able to detect 3 times more the exit of a customer from the bank, a critical aspect of our situation.



It is  interesting to see how despite ranking so high on the Smote variants competition [10]5], the Polynom-fit and ProSWyn variants have been no match on recall score for the parent oversampler of these: SMOTE, that may also be because throughout the experiment we have focused on optimizing the prediction models for the SMOTE data. However, not all variants perform poorly, Lee-Smote also emerges as one of the winners in the competition and achieves very good Recall values along with normal SMOTE.

This issue that theoretically stronger oversamplers have fallen in our experiment opens the door for the exploration of the optimal hyperparameters for each oversampler, focusing on the way in which it generates new samples certain hyperparameter could be selected to drastically change the model behaviour.

## Comparison with competition and final thoughts

In Figure 18 we analyse our results, we select those projects that have used the same metrics as we did with focus in involving all the recall scores on the competition. We have used weighted average precision, recall and f1 score since it was present in all the notebooks. Team referring to our project has name “us” and the projects on Table 1 are “rivals”. X-Axis, ID, referrers to the project ID, in our case to the model employed.

 First thing to note is how we are vastly outclassed in precision. It was one of the expected points, as there are so many values of a class, if a classifier only looks at that class it makes a lot of sense to get a good overall accuracy value as there are many more instances in the overall computation of that class. Still, it is worth noting that the values are better than we expected, between 0.70 and 0.75 so it is a value that we could consider good enough.

Moving on to the recall metric, we remain at the top of the list. This is a total success on our part, as this was the target we were aiming for. Not only did we get the best value of the competition, out of the top 6 in recall, we occupy 5 places, a complete victory that is only complemented when we move on to the f1-Score.

Having obtained better than expected Recall values, we were also able to stay on top in F1 Score, a metric that shows very well the balance of the classes, if the two classes had been equally important as part of the problem, this metric would be the ideal one. By also being so high, we can confirm that we have not disregarded the majority class in favour of the minority class, an achievement to be celebrated.

Finally, we move on to accuracy, being a metric that depends a lot on the distribution of the classes we have not the best values of the competition, however, our values are good enough to keep us between 0.8 -0.85, more than remarkable values when we clearly had another metric in mind.

Regarding the knowledge gained from comparing our results with the rest of the Kaggle competition, we see that perhaps we have not analysed the problem from its most problematic variant as almost all the notebooks in the competition show firmly and honourably the value obtained from accuracy as their white weapon.

However, this does not reflect the nature of the reality of the data, with such a poorly balanced dataset it should not be very difficult without oversampling to reach very high accuracy rates as the majority class appears much more frequently and with a good optimization of hyperparameters we may could have reach high accuracy peaks.

It is also a teaching of value based on this when it seems important or not to look for other metrics beyond the accuracy. If we had a dataset where the relevance of one class or another was the same it would not make sense to use an oversampler, because by obtaining a high accuracy you will already predict the correct value for all the data to come.

Therefore, the value of using one metric or another, or using oversamplers or not is beyond a problem of numbers and lies entirely in a problem of intuition. How many customers can the bank lose if we do not get all those who are likely to leave? How much potential money could this mean?

As Data Scientist it is very important not to have a tunnel vision only with the technical aspect, but to analyse, study and above all understand the nature of the data and where we can focus our efforts to get the best possible solution.

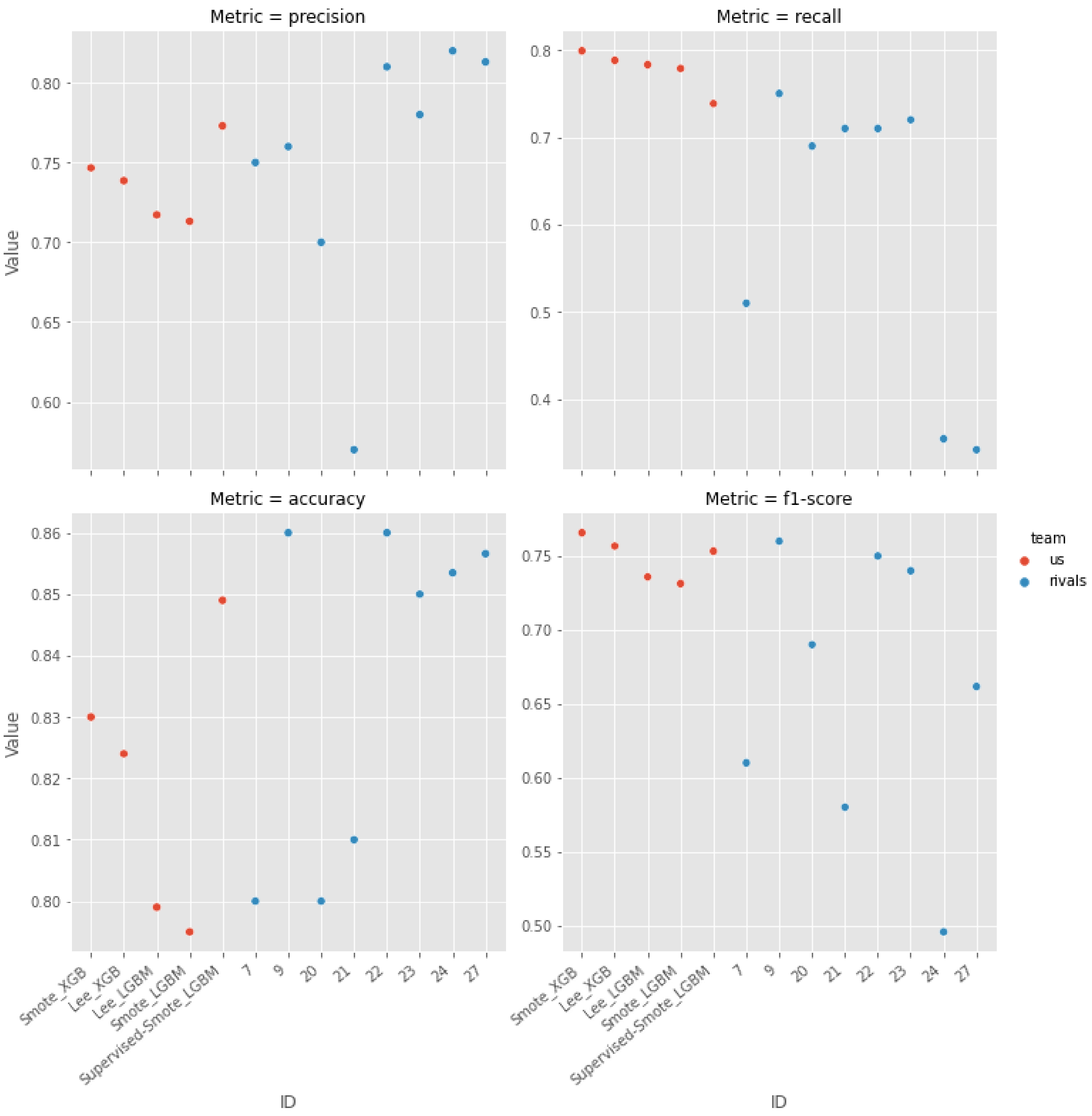


**Figure 18. Competition**

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| Results | LightGBM | Polynom-fit | 0.8360 | 0.2829 | 0.4143 |
|  |  | Supervised-Smote | 0.8490 | 0.5512 | 0.5995 |
|  |  | ProWSyn | 0.8410 | 0.5220 | 0.5737 |
|  |  | Lee | 0.7990 | 0.7561 | 0.6067 |
|  |  | Smote | 0.7950 | 0.7512 | 0.6004 |
|  |  | No - Smote | 0.8370 | 0.2585 | 0.3941 |
|  | XGBoost | Polynom-fit | 0.8340 | 0.2146 | 0.3465 |
|  |  | Supervised-Smote | 0.8460 | 0.4244 | 0.5305 |
|  |  | ProWSyn | 0.8460 | 0.4244 | 0.5305 |
|  |  | Lee | 0.8240 | 0.7268 | 0.6287 |
|  |  | Smote | 0.8300 | 0.7463 | 0.6429 |
|  |  | No - Smote | 0.8410 | 0.2585 | 0.4000 |
|  | AdaBoost | Polynom-fit | 0.8000 | 0.5140 | 0.5025 |
|  |  | Supervised-Smote | 0.7960 | 0.5344 | 0.5072 |
|  |  | ProWSyn | 0.7965 | 0.5318 | 0.5067 |
|  |  | Lee | 0.7860 | 0.5496 | 0.5023 |
|  |  | Smote | 0.7885 | 0.5522 | 0.5064 |
|  |  | No - Smote | 0.7980 | 0.5064 | 0.4963 |
|  | COMPETITION |  | Accuracy | Recall | F1_s |

**Table 2. Results**

# Conclusions

To conclude the document, we will analyse how we have been able to meet our objectives.

The main aim of the project was to build a model capable of identifying which customers could potentially exit the service. We have achieved this as our model identifies almost 8/10 customers who have exited the service in the given dataset. Furthermore, we have done this without discounting the customers who are still within the service, which we also count at about 7-8 customers out of 10.

This allows us to conclude that our main objective has been met.

In our own model search we found that the metric we were going to use as a priority, accuracy, did not meet the needs of our scenario, so we turned to other metrics such as Recall and f1-score to make up for its weaknesses.

This also represents one of our secondary objectives, the search for the most suitable metrics for the project.

Regarding data processing, we have applied normalisation techniques, we have eliminated redundant variables or variables that contained very little useful information and we have replaced categorical variables with binary information.

## Limitations

One limitation has arisen in the project: processing power. Working with models that generate a lot of information per second, a good source of calculations was needed. Not having a Pro version of Google Collaboratory meant that we were sometimes unable to run the entire notebook due to Ram memory limitations.

However, by running the code in parts over several hours we were able to reduce this limitation to some extent.

## Future scope

This project has made us realise that there is still a lot of ground to be covered. If we focus on the results, we see that when we tune the parameters of the boosting algorithms, we get very good results for some oversamplers but others drop even more than the model without data augmentation.

This opens the door to a study of the optimal hyperparameters for each oversampling technique, and we can obtain from this research an understanding of how the hyperparameters fit the generated data.

## Bibliography

[1] Z.-H. Zhou, “Machine learning challenges and impact: an interview with Thomas Dietterich,” *Natl. Sci. Rev.*, vol. 5, no. 1, pp. 54–58, Jan. 2018, doi: 10.1093/nsr/nwx045.

[2] R. Mohammed, J. Rawashdeh, and M. Abdullah, “Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results,” in *2020 11th International Conference on Information and Communication Systems (ICICS)*, Apr. 2020, pp. 243–248. doi: 10.1109/ICICS49469.2020.239556.

[3] M. Hossin and S. M.N, “A Review on Evaluation Metrics for Data Classification Evaluations,” *Int. J. Data Min. Knowl. Manag. Process*, vol. 5, pp. 01–11, Mar. 2015, doi: 10.5121/ijdkp.2015.5201.

[4] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, “Handling imbalanced datasets: A review,” *GESTS Int. Trans. Comput. Sci. Eng.*, vol. 30, pp. 25–36, Nov. 2005.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.

[6] E. Fix and J. L. Hodges, “Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties,” *Int. Stat. Rev. Rev. Int. Stat.*, vol. 57, no. 3, pp. 238–247, 1989, doi: 10.2307/1403797.

[7] G. Batista and M. C. Monard, “A Study of K-Nearest Neighbour as an Imputation Method,” 2003.

[8] G. Kovács, “Smote-variants: A python implementation of 85 minority oversampling techniques,” *Neurocomputing*, vol. 366, pp. 352–354, Nov. 2019, doi: 10.1016/j.neucom.2019.06.100.

[9] “Ranking — smote_variants 0.1.0 documentation.” https://smote-variants.readthedocs.io/en/latest/ranking.html (accessed Jun. 01, 2022).

[10] G. Kovács, “An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets,” *Appl. Soft Comput.*, Jul. 2019, doi: 10.1016/j.asoc.2019.105662.

[11] S. Gazzah and N. ESSOUKRI BEN AMARA, “New Oversampling Approaches Based on Polynomial Fitting for Imbalanced Data Sets,” Sep. 2008, pp. 677–684. doi: 10.1109/DAS.2008.74.

[12] S. Barua, Md. M. Islam, and K. Murase, “ProWSyn: Proximity Weighted Synthetic Oversampling Technique for Imbalanced Data Set Learning,” in *Advances in Knowledge Discovery and Data Mining*, Berlin, Heidelberg, 2013, pp. 317–328. doi: 10.1007/978-3-642-37456-2_27.

[13] Y. Freund and R. E. Schapire, “Experiments with a New Boosting Algorithm.” 1996.

[14] B. Bischl *et al.*, “Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges.” arXiv, Nov. 24, 2021. Accessed: Jul. 28, 2022. [Online]. Available: http://arxiv.org/abs/2107.05847

[15] “Introduction to Boosted Trees — xgboost 1.6.1 documentation.” https://xgboost.readthedocs.io/en/stable/tutorials/model.html (accessed Jul. 28, 2022).

[16] L. Breiman, “Random Forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[17] W. I. Ai and P. Langley, “Induction of One-Level Decision Trees,” in *Proceedings of the Ninth International Conference on Machine Learning*, 1992, pp. 233–240.

[18] “A Guide To Understanding AdaBoost,” *Paperspace Blog*, Feb. 23, 2020. https://blog.paperspace.com/adaboost-optimizer/ (accessed Jun. 02, 2022).

[19] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[20] J. H. Friedman, “Greedy function approximation: A gradient boosting machine.,” *Ann. Stat.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001, doi: 10.1214/aos/1013203451.

[21] G. Ke *et al.*, “LightGBM: A Highly Efficient Gradient Boosting Decision Tree,” in *Advances in Neural Information Processing Systems*, 2017, vol. 30. Accessed: Jul. 28, 2022. [Online]. Available: https://papers.nips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html

[22] J. Brownlee, “How to Develop a Light Gradient Boosted Machine (LightGBM) Ensemble,” *Machine Learning Mastery*, Nov. 24, 2020. https://machinelearningmastery.com/light-gradient-boosted-machine-lightgbm-ensemble/ (accessed Jul. 28, 2022).

[23] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943, doi: 10.1007/BF02478259.

[24] B. Widrow, “An adaptive ‘ADALINE’ Neuron using chemical ‘memistors,’” *Tech. Rep.*, vol. 2, no. 1553, Oct. 1960, [Online]. Available: https://www-isl.stanford.edu/~widrow/papers/t1960anadaptive.pdf

[25] D. Srivastava and L. Bhambhu, “Data classification using support vector machine,” *J. Theor. Appl. Inf. Technol.*, vol. 12, pp. 1–7, Feb. 2010.

[26] “What is SVM | Build an Image Classifier With SVM,” *Analytics Vidhya*, Jun. 18, 2021. https://www.analyticsvidhya.com/blog/2021/06/build-an-image-classifier-with-svm/ (accessed Aug. 10, 2022).

[27] “Quantal response curves for experimentally uncontrolled variables – ScienceOpen.” https://www.scienceopen.com/document?vid=25b1e892-14d3-4154-bdfa-73821c931349 (accessed Aug. 10, 2022).

[28] M. D. Fiuza Pérez and J. C. Rodríguez Pérez, “La regresión logística: una herramienta versátil,” *Nefrología*, vol. 20, no. 6, pp. 495–500, Dec. 2000.

[29] "Churn Model (Accuracy: 88.2%; Recall: 88.7%)." https://kaggle.com/code/catherinesohk/churn-model-accuracy-88-2-recall-88-7 (accessed Aug. 23, 2022).

[30] "Customer Churn Modelling | XGBoost | 87%." https://kaggle.com/code/suyashlakhani/customer-churn-modelling-xgboost-87 (accessed Aug. 23, 2022).

[31] "Churn(ML_all_in_one)." https://kaggle.com/code/aliderakhshesh/churn-ml-all-in-one (accessed Aug. 23, 2022).

[32] "Java's Dilemma." https://kaggle.com/code/duttasd28/java-s-dilemma (accessed Aug. 23, 2022).

[33] "Churn Modelling." https://www.kaggle.com/shubh0799/churn-modelling (accessed Jun. 09, 2022).

[34] "Customer Churn Modeling: EDA + FE + Model 💼." https://kaggle.com/code/galaxygeorge/customer-churn-modeling-eda-fe-model (accessed Aug. 23, 2022).

[35] "🤖Neural Networks OPT w/ Keras Tuner and Optuna🛠." https://kaggle.com/code/ludovicocuoghi/neural-networks-opt-w-keras-tuner-and-optuna (accessed Aug. 23, 2022).

[36] "Customer Churn Prediction Using ANN." https://kaggle.com/code/niteshyadav3103/customer-churn-prediction-using-ann (accessed Aug. 23, 2022).

[37] "Churn-Classification." https://kaggle.com/code/souhardyaganguly/churn-classification (accessed Aug. 23, 2022).

[38] "Churn: ANN + Data Viz. + Tuning (86.3% Accuracy)." https://kaggle.com/code/siddheshera/churn-ann-data-viz-tuning-86-3-accuracy (accessed Aug. 23, 2022).

[39] "modelling for freshers with 6 classifiers(acc 87%)." https://kaggle.com/code/sachinsharma1123/modelling-for-freshers-with-6-classifiers-acc-87 (accessed Aug. 23, 2022).

[40] "Churn Modeling w/TF." https://kaggle.com/code/tahaakr/churn-modeling-w-tf (accessed Aug. 23, 2022).

[41] "churnModel." https://kaggle.com/code/ahmetburabua/churnmodel (accessed Aug. 23, 2022).

[42] "Churn Modelling | EDA | ANN." https://kaggle.com/code/d4rklucif3r/churn-modelling-eda-ann (accessed Aug. 23, 2022).

[43] "Customer Churn Prediction with Keras Tuner." https://kaggle.com/code/anuragupadhyay6212/customer-churn-prediction-with-keras-tuner (accessed Aug. 23, 2022).

[44] "XGBoost & Cross_Validation." https://kaggle.com/code/avishakemaji/xgboost-cross-validation (accessed Aug. 23, 2022).

[45] “ANN Simplified (Churn Dataset).” https://kaggle.com/code/rohitamalnerkar/ann-simplified-churn-dataset (accessed Aug. 23, 2022).

[46] “ANN (Churn Modelling).” https://kaggle.com/code/harshavarshney/ann-churn-modelling (accessed Aug. 23, 2022).

[47] “91% Accuracy / NeuralNetwork/ SupportVectorMachine.” https://kaggle.com/code/vyombhatia/91-accuracy-neuralnetwork-supportvectormachine (accessed Aug. 23, 2022).

[48] “Customer_Churn_ANN.” https://kaggle.com/code/dhruvkalia/customer-churn-ann (accessed Aug. 23, 2022).

[49] “Churn | XGBOOST | EDA.” https://kaggle.com/code/mokar2001/churn-xgboost-eda (accessed Aug. 23, 2022).

[50] “Churn Modeling : EDA + ANN + ML.” https://kaggle.com/code/berkinkaplanolu/churn-modeling-eda-ann-ml (accessed Aug. 23, 2022).

[51] “Bank-Prediction(ANN&RandomForest).” https://kaggle.com/code/sumitsingh20/bank-prediction-ann-randomforest (accessed Aug. 23, 2022).

[52] “EDA + 86% ANN explained.” https://kaggle.com/code/agustinpugliese/eda-86-ann-explained (accessed Aug. 23, 2022).

[53] “Just use Pipeline!” https://kaggle.com/code/mahmoudafify/just-use-pipeline (accessed Aug. 23, 2022).

[54] “Comparing Classification for Churn Prediction.” https://kaggle.com/code/mabolhal/comparing-classification-for-churn-prediction (accessed Aug. 23, 2022).

[55] “Churn prediction with LR , Trees , Random Forest.” https://kaggle.com/code/wadihaleid/churn-prediction-with-lr-trees-random-forest (accessed Aug. 23, 2022).

[56] “Churn Modelling.” https://kaggle.com/code/djrmarques/churn-modelling (accessed Aug. 23, 2022).

[57] “Churn: Visualization, RandomForest(86%), ANN(86%).” https://kaggle.com/code/hayrettinbalc/churn-visualization-randomforest-86-ann-86 (accessed Aug. 23, 2022).

[58] “Neural Network to Model Employee Churn.” https://kaggle.com/code/madhavmalhotra/neural-network-to-model-employee-churn (accessed Aug. 23, 2022).

[59] “sklearn.metrics.classification_report,” *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.metrics.classification_report.html (accessed Aug. 18, 2022).

[60] “Project Jupyter.” https://jupyter.org (accessed Aug. 18, 2022).

[61] “Welcome to Python.org,” *Python.org*. https://www.python.org/ (accessed Aug. 18, 2022).

[62] “pandas - Python Data Analysis Library.” https://pandas.pydata.org/ (accessed Aug. 18, 2022).

[63] M. Waskom, “seaborn: statistical data visualization,” *J. Open Source Softw.*, vol. 6, no. 60, p. 3021, Apr. 2021, doi: 10.21105/joss.03021.

[64] “Kaggle: Your Machine Learning and Data Science Community.” https://www.kaggle.com/ (accessed Jun. 09, 2022).

[65] “Kaggle: todo lo que hay que saber sobre esta plataforma,” *Formación en ciencia de datos | DataScientest.com*, Dec. 14, 2021. https://datascientest.com/es/kaggle-todo-lo-que-hay-que-saber-sobre-esta-plataforma (accessed Jun. 09, 2022).

[66] “Gradient Boosting | Hyperparameter Tuning Python,” *Analytics Vidhya*, Feb. 21, 2016. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/ (accessed Jul. 28, 2022).

[67] “A Guide on XGBoost hyperparameters tuning.” https://kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning (accessed Jul. 28, 2022).

[68] “Parameters — LightGBM 3.3.2.99 documentation.” https://lightgbm.readthedocs.io/en/latest/Parameters.html (accessed Jul. 28, 2022).

[69] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework.” arXiv, Jul. 25, 2019. doi: 10.48550/arXiv.1907.10902.