Master Thesis
MSc Industrial Technology

Performance evaluation of different machine learning methods applied on churn database

Author:	L. Acorán Rodríguez Suárez
Director:	Pau Pijuan Casanova
Semester:	February 2022 - September 2022
Submission date:	September 15, 2022

ETSEIB

Barcelona School of Industrial Engineering

UPC

Polytechnic university of catalonia

Abstract

The growth of data and its storage is becoming more and more important every day. However, occasionally this information is gathered but never used, or perhaps it is improperly gathered, making the extraction of the insides difficult. As a result, while beginning any project, choosing the analysis method is just as crucial as choosing the design of the data collection strategy. Most of the time, we only focus on the analysis of the data and do not consider how it was gathered or whether the fields were actually valuable or just added noise to what we were searching for. For this reason, a trustworthy data set has been chosen for this project.

The data came from a telecom company, which, like other modern businesses, collects a lot of data. However, in this case, the data was published on the machine learning web competition Kaggle, where participants competed to build the best model to predict consumer behaviour. One of the key considerations in optimizing any organization's income is preventing customer churn. It happens when customers quit utilizing a company's goods or services, and is also referred to as customer attrition.

The main goal of this master's thesis is to analyse a Churn database and categorise the clients in order to determine whether they are likely to leave the company. To do this, two machine learning techniques will be used in the current document. Extreme Gradient Boosting and Random Forest. In order to achieve high performance, the Random Forest (RF) method creates a large number of low-performance models and combines them. In this case, the lower-performance method is called Decision Tree, so it will be explained in more detail in the following document. Similar work is done by eXtreme Gradient Boosting (XGB), although it builds new models based on earlier findings. Both are quite effective predictor models, even with unbalanced data, as will be demonstrated in the next document. This adds another level of complexity that the algorithms must overcome to execute effectively.

Different performance indicators will be provided and examined in order to determine which one is the greatest indicator to choose the best model during the process of determining the best model. Sensitivity, Specificity, Precision, F1 Score, and Geometric Mean are a few of the markers that are listed. Additionally, their trends for the various parameter values of the examined models will be shown and analysed.

The strong performance of these machine learning algorithms will once more be supported in this thesis. the affirmation of the significance and practical use of these methodologies, as in the case of this project, to comprehend processes and behaviours. All fields can benefit from the information gleaned, and a successful application will undoubtedly yield financial rewards.

The two machine learning applied algorithms' default and best models are finally shown, and their advantages and disadvantages will be evaluated while taking into account the many scenarios that exist. This thesis will demonstrate the good performance of both models, with XGB significantly outperforming RF. It will also demonstrate that while XGB performs better on precision and RF has better results on sensitivity.

Key words: machine learning, churn, random forest, extreme gradient boosting, F1 Score, Sensitivity, Specificity, Decision tree, parameter customization, grid search, Telecom

Table of content

Abstract **2**

Methodology **5**

Literature review **7**

Telecom Data Analytics Enables Better Investment Utilization 7

Big Data 8

Decision Tree **12**

Gini index 13

Random Forest **16**

Customization 18

eXtreme Gradient Boosting **19**

Parameters customization 22

Evaluation metrics **24**

Data source **25**

Context 25

Content 25

Data clean **26**

Exploratory data analysis 27

Unbalanced data **31**

Changing performance metric **31**

Random Forest performance **36**

Mtry 36

Ntree 38

Nodesize 39

XGBOOST performance **42**

Max depth 42

Gamma 43

Eta 45

Min child weight 46

Code **49**

Optimal model search **50**

Comparing xgboost against random forest **54**

Conclusion **58**

Bibliography **59**

ANNEX: CODE **61**

EDA (Exploratory Data Analysis) **63**

RandomForest_v2 **65**

rf_calculations 72

find_best_rf_model 75

XGBoosting_v4 **76**

xgb_calculations_v3 86

find_best_model 90

comparing_best_models **92**

Glossary

List of figures

Figure 1: Predictive analytics 15

Figure 2: Decision Tree 18

Figure 3: Distribution of points in case of high and low information gain. 20

Figure 4: Representation of entropy. 20

Figure 5: Branch, internal node and leaf node explanation 21

Figure 6: Difference between Bagging and Boosting 22

Figure 7: Random Forest classifier 23

Figure 8: Representation of the algorithm's operation 28

Figure 9: Summary of numeric variables 33

Figure 10: Day and Eve histograms 34

Figure 11: Night and Intl histograms 34

Figure 12: Correlation between day.minutes and day.charge 35

Figure 13: Correlation matrix 35

Figure 14: Categorical variables statistics 36

Figure 15: Histogram of State 36

Figure 16: Churn distribution 37

Figure 17: ROC Curve 40

Figure 18: ROC Curve 2 41

Figure 19: Performance metrics by mtry. purple: Sensitivity test, red: Specificity test, green: Sensitivity train, orange: Specificity train 42

Figure 20: Error by mtry. purple: Error rate, red: False class error test, green: True class error test, orange: False class error train, blue: True class error train 43

Figure 21: Performance metrics by mtry. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train 43

Figure 22: Performance metrics by ntree. purple: Sensitivity test, red: Specificity test, green: Sensitivity train, orange: Specificity train 44

Figure 23: Error by ntree. Purple: False class error test, orange: True class error test, red: False class error train, green: True class error train 44

Figure 24: Performance metrics by ntree. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train 45

Figure 25: Performance metrics by nodesize. purple: Accuracy, red: Sensitivity test, green: Specificity test, orange: Sensitivity train, blue: Specificity train 45

Figure 26: Error by nodesize. Purple: False class error test, orange: True class error test, red: False class error train, green: True class error train 46

Figure 27: Performance metrics by nodesize. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean trainç 46

Figure 28: Performance metrics by max depth. Red: Sensitivity test, orange:Specificity test, green: Sensitivity train, blue:Specificity train 48

Figure 29: Error by max depth. Red: False class error test, green: True class error test, orange: False class error train, blue: True class error train 48

Figure 30: Performance metrics by max depth. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train 49

Figure 31: Performance metrics by gamma. Red: Sensitivity test, orange:Specificity test, green: Sensitivity train, blue:Specificity train 49

Figure 32: Error by gamma. Red: False class error test, green: True class error test, orange: False class error train, blue: True class error train 50

Figure 33: Performance metrics by gamma. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train 50

Figure 34: Performance metrics by eta. Red: Sensitivity test, orange:Specificity test, green: Sensitivity train, blue:Specificity train 51

Figure 35: Error by eta. Red: False class error test, green: True class error test, orange: False class error train, blue: True class error train 51

Figure 36: Performance metrics by eta. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train 52

Figure 37: Performance metrics by min_child_weight. Red: Sensitivity test, orange:Specificity test, green: Sensitivity train, blue:Specificity train 52

Figure 38: Error by min_child_weight. Red: False class error test, green: True class error test, orange: False class error train, blue: True class error train 53

Figure 39: Performance metrics by min child weight. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train 53

Figure 40: Histograms of F1 test, F1 train and the overfitting for Extreme Gradient Boosting 57

Figure 41: Histograms of F1 test, F1 train and the overfitting for Random Forest 58

Figure 42: Representation of F1 default random test and train, F1 costum random test and train, F1 default xgboost test and train, F1 custom xgboost test and train. 60

Figure 43: Representation of Precision and Sensitivity for training and testing data on the 4 models. 61

Figure 44: Importance of Random forest default model 63

Figure 45: Importance of Random forest customised model **64**

List of Tables

Table 1: Data types **32**

Table 2: Distribution by States **36**

Table 3: Confusion Matrix **38**

Table 4: Selected values for the grid (RF) **56**

Table 5: Selected values for the grid (XGB) **57**

Table 6: Summary of F1 test, F1 train and the overfitting for Random Forest **58**

Table 7: Default and optimal values of RF and XGB **59**

Table 8: Values of F1 default random test and train, F1 costum random test and train, F1 default xgboost test and train, F1 custom xgboost test and train and Improvements. **60**

Table 9: Summary of Precision test, Precision train, Sensitivity test and Sensitivity train for RF and XGB. **61**

List of Equations

Eq. 1: Gini Index 19

Eq. 2: Weighted Gini Index 19

Eq. 3: Entropy 20

Eq. 4: Summatory of decision trees 25

Eq. 5: Prediction of the model 25

Eq. 6: Prediction at the -th step 25

Eq. 7: Rooted mean square error 25

Eq. 8: LogLoss for binary classification 25

Eq. 9: mlogloss for multi-classification 26

Eq. 10: Regularisation 26

Eq. 11: Objective function 26

Eq. 12: Gradient descent 26

Eq. 13: Boosting 26

Eq. 14: Redefinition of objective function 27

Eq. 15: first and second order gradient 27

Eq. 16: Second order Taylor approximation of Objective function 27

Eq. 17: Gradients 27

Eq. 18: Final Objective function 27

Eq. 19: Area Under the Curve 30

Eq. 20: Gini Coefficient 30

Eq. 21: Cross-entropy loss 30

Eq. 22: Type I error 38

Eq. 23: Type II error 38

Eq. 24: F1 Score 39

Eq. 25: Error rate 39

Eq. 26: Sensitivity 39

Eq. 27: Specificity 39

Eq. 28: Precision 40

Eq. 29: Overfitting **57**

Methodology

This project has as a main objective the study of two different, ensemble methods, in this case, based on Decision Trees, Random Forest and Extreme Gradient Boosting. As part of this study this essay will emphasise on the impact of customization of the different machine learning algorithms giving to the reader a global vision of the importance of choosing the right parameters when defining the model.

For doing that, a case of study has been selected to generate a model that comes from a dataset where there is a clear objective. The dataset selected has been obtained from Kraggle, an open-source web where many real problems from the different industries are presented. Everyone with an internet connection is allowed to participate in the competition and the final objective is to create the best algorithm possible. With this purpose thousands of participants invest their time while creating different discussions about the best way to face the problem so although it is a competition, the true spirit is about sharing knowledge and developing the skills.

The selected dataset is called Churn Dataset, it's a dataset updated 3 years ago. This dataset has information related to telephone companies such as money expenses, number of calls or type of contract selected. The variable objective of the study it's called Churn, it informs if the client has left the company or not. It's a very important variable that almost every company would want to be able to predict, as it allows the ones that have the information to act consequently. For example, knowing this information not only gives an idea to the company on how many users can have in the short-term future but it allows it to focus on those clients who will potentially leave the company and act to prevent it from happening.

As a source of information on this project there are given two different datasets, called TRAIN and TEST. First one to create and train the model and second one to check the results obtained on train to ensure that the algorithm isn't overfitted and it has not only the ability to predict the data that has been shown before but the data that the algorithm has never seen. That way it's proven that the algorithm has a high reliability on unknown data.

Literature review

Before beginning this study, extensive research into the telecom industry was conducted, with an emphasis on how much data this industry has and what it uses it for. We'll discuss several more methods for studying the data, including big data, which is becoming more and more popular as a method of problem-solving.

It can cost up to 25 times more to get new clients than it does to keep the ones you already have. To lower their customer churn rates, carriers rely on data analytics employing demographics, usage, connection, network performance, and more. Customer turnover research is nonetheless limited by the fact that the size of this data is too large for their current telecom analytics solutions. Users of HEAVY.AI may instantly analyze and display all of their data to find abnormalities and dive down to particular network problems that may be causing customer churn.[1]

The job of data analytics in telecom is to give firms with the simplest approach to unearth insights from all their data because a telecom company has to keep up with new technological advancements, stay ahead in one of the world's most competitive industries, and adhere to a myriad of regulations. A corporation will make more money if it uses a telecom data analytics solution that will improve its insights. The organization will be able to better anticipate customer wants and stay one step ahead of its rivals with the help of such a system.

Telecom Data Analytics Enables Better Investment Utilization

A business can base choices on data, not speculation, with the simple data analysis provided by a telecom data analytics system. Data analytics' function in the telecom industry is to provide each organization with a unified picture of its data across departmental boundaries. When data comes in from many sources across the organization, the business may use the ideas of all its teams to find the best solution for each problem.

Prioritize Data Analytics for Telecom Success

With information coming in from all of the departments and from various sources, a telecom service provider may make the most of the skills of every employee to solve problems, develop creative new workflows, and satisfy customers. The data will be presented to employees using the Necto Telecom data analytics system in beautiful infographics that are simple to interpret. To the data, they can add their own annotations to gain even more understanding of how the business is doing.

Data Analysis of Top Importance for Telecom

When it comes to data analysis, a telecom corporation cannot afford to lag behind. Telecommunications services will have spent close to two-fifths of the global total on smart data and cloud technologies, claims data analysis think leader Gartner. If the business doesn't adopt a top-notch telecom data analytics system, the data-hungry telecom sector will soon be killed off, leaving you with a dinosaur.

Connect the Data Dots with Unified Business Insights

Every division of a provider of communications and media services produces a lot of data. Other departments are unable to benefit from one another's data analysis insights without a coordinated approach. Furthermore, the data will frequently be different from that of other departments without a centralized solution, causing confusion and even resentment.

All of the departments' data is combined into one single system with a consolidated data analytics solution. Each department can add its own perspective to the data gathered, providing a fresh viewpoint that will speed up the process of finding a solution. Each department can benefit from the knowledge of the others, enhancing production and efficiency.

Imagine a Better Future with Telecom Data Analytics

Imagine a system in which all the departments in a firm may share knowledge and resources in order to better comprehend the function that data analytics plays in telecom. This is a controlled and secure approach because all users can access the data in accordance with the permissions that the IT department has already defined.

The departments will operate as one efficient, money-making machine if they pool their efforts to reduce service costs, promote expansion, and take fast action when key performance indicators (KPIs) change.[2]

Big Data

Over time, data science has proven to be highly valuable and effective. Data scientists are always thinking of new ways to apply big data solutions to daily life. Today, data is a vital component of any successful organization.

Businesses in the telecoms sector are no different. In these conditions, they cannot afford to forgo the use of data science. The telecom sector makes extensive use of data science tools to automate processes, increase revenue, create profitable marketing and business plans, visualize data, transfer data, and execute a number of other functions. Data import, export, and transfer are all crucial processes for businesses in the telecommunications sector. Data is moving across multiple communication channels at a faster rate every minute. Therefore, it is no longer necessary to use outmoded strategies and procedures.

A platform for Big Data and Cloud

The early days of telecommunications data storage were plagued by a number of problems, such as unmanageable volumes, a lack of computational power, and exorbitant pricing. New technologies, however, have changed the nature of the issues.

Technology used in the following areas:

- Data storage costs are falling every day thanks to the Cloud Platform. (AWS, Azure)
- The processing power of computers is growing at an exponential rate (Quantum Computing)
- Analytics software and tools are inexpensive, and some are even free (Knime, Python)

Because data stores were expensive in the past, data was kept in distinct, frequently incompatible silos. This was preventing a wide range of applications from using a vast amount of data. There are several opportunities for telecom data scientists as a result of the breakdown of the barriers between data storage and analysis by business intelligence (BI) providers including IBM, Oracle, SAS, Tibco, and QlikTech.

Predictive Analytics

Figure 1: Predictive analytics

Predictive analytics is a tool used by businesses in the telecommunications sector to collect insightful data that enables them to make quicker, better, and data-driven decisions. You can better understand the tastes and needs of the customer if you are aware of them. Forecasts are made using historical data and predictive analytics. Longer data collecting durations and higher data quality both increase predictability.[3]

In the literature, churn prediction has been done using a variety of methodologies, including machine learning, data mining, and hybrid techniques. These methods aid businesses in decision-making and CRM by helping them to recognize, anticipate, and keep churning customers. The decision trees are the most well-known techniques for predicting issues related to client turnover. [4].

The decision tree is constrained in that it performs better for linear data where the characteristics depend on one another than it does for complex nonlinear linkages between

attributes. However, the research demonstrates that pruning raises the decision tree's accuracy. [5].

Decision tree algorithms have a number of benefits: They can analyze categorical and numerical data, are simple to display and understand, and use a nonparametric method that does not require previous assumptions. [6].

The data used in this analysis is linear and we intend to identify rules and hidden pattern through the decision tree. A neural network based methodology for the prediction of churn customers in the telecom sector is provided in [7]. In literature, churn prediction is also performed using data certainty [8] and particle swarm optimization [9].

Another study compares ANN and decision trees for churn prediction, and the results show that the accuracy of the decision tree-based strategy is higher than that of the neural network-based approach.[10]. This work was further extended by a study which aimed at finding answers to customer loyalty results in prepaid mobile phone organizations [11]. In this work, a two-step approach is used for prediction. In the first step, RFM related features are divided into four clusters and in the second step the churn data, which is extracted in the first step, is tested on different algorithms using Decision Tree (C5.0), Decision (CART), Neural Networks,andDecision Tree (CHAID). It shows that the hybrid approach resulted in better performance as compared to a single algorithm.

The study proposed by [12] is a hybrid approach for churn prediction and results showed better performance using existing tree induction algorithm with genetic programming to derive classification rules based on customer behavior. Predictive models for churn customers regarding prepaid mobile phone companies are described in [11], [13]. In another study, authors use Support Vector Machine (SVM), Neural net, Naïve Bayes, K-nearest neighbors and Minimum-Redundancy MaximumRelevancy (MRMR) features selection technique [14]. In Statistical approaches, hybrid techniques are used for processing large amounts of customer data including regression-based techniques that produced good results in predicting and estimating churn [15]. Data mining algorithms are often used in customer history analysis and prediction. The techniques of regression trees were discussed with other commonly used data mining methods such as decision trees, rules-based learning, and neural networks [15], [4]. Naive Bayes is a guided learning module that predicts invisible data based on the position of Bayesian, is used to predict churn customer [10]. Churn problem for wireless-based customer data is discussed in [16].

There is a range of hybrid techniques proposed in the literature for churn prediction. One such technique, named KNN-LR, is a hybrid approach using Logistic Regression (LR) and K-Nearest Neighbor (KNN) algorithm is used in the study [17].

They conducted a comparison between KNN-LR, logistic regression, C 4.5 and Radial Basis Function (RBF) network and found that KNN-LR is superior in performance to all the other approaches. The novel model presented in [18] shows a hybrid approach linking the adapted k-means clustering algorithm with the classical rule inductive technique (FOIL) to predict churn customer behavior. The control of a large volume of data in today’s world provides an opportunity to improve the quality of service to the users. This data includes information about customers behavior, usage pattern and network operations.

The study [19] proposed a model for both online and offline distributed framework based on data mining techniques to predict and identify churn customers. The model is appropriate for

telecom to improve the CRM and its quality of service in different aspects. Particle Swarm Optimization (PSO) techniques are used for features selection as its preprocessing mechanism. The telecommunications service sector has undergone a major change over the past decade due to new services, state-of-the-art upgrades [20]–[21] and intensified competition due to deregulation [22]. There is a need to secure important customers, strengthen connection management of CRM and improve the profitability [23], [24]. CRM needs that a company knows and understands its business units and customers. CRM controls improvements in offers and discounts. CRM also controls which services (including media and promotions etc.) are offered to which customers.

Decision Tree

In this master thesis some ensemble learning methods based on decision trees are explained so first, we need to explain what it is and how a decision tree works.

Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning. There are two main types of decision trees, classification trees and regression trees. First one is used when the target variable takes a discrete set of values; the leaves of the tree structures represent class labels and the branches represent conjunctions of features that lead to those class labels. And regression trees are used when the target variable is continuous. In this case the leaves show the mean values of the data points that this one contains.

Both work the same way; they represent a tree where the nodes (leaves) are the condition and the edges (branches) are the possible outcomes. The process consists of splitting until no further gain can be made or a rule is met. To give a visual idea of what a decision tree is, the next Figure is presented and further explanation will be given later.

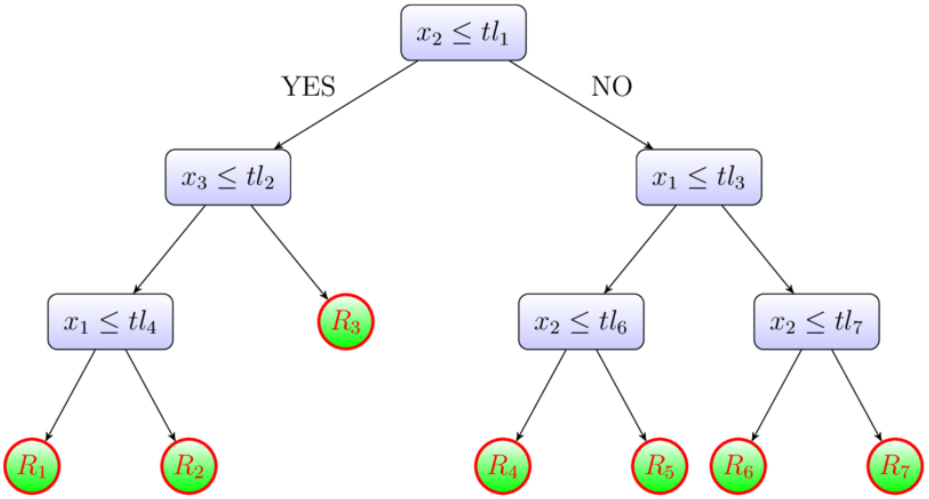

Figure 2: Decision Tree

For speaking about Decision Tree first we need to introduce some concepts that will help us to understand how the algorithm works:

- Partitioning: It refers to the process of splitting the data set into subsets. The decision of making strategic splits greatly affects the accuracy of the tree. Many algorithms are used by the tree to split a node into sub-nodes which results in an overall increase in the clarity of the node with respect to the target variable. Various Algorithms like the chi-square and Gini index are used for this purpose and the algorithm with the best efficiency is chosen.
- Pruning: This refers to the process where the branch nodes are turned into leaf nodes which results in the shortening of the branches of the tree. The essence behind this

idea is that overfitting is avoided by simpler trees as most complex classification trees may fit the training data well but do an underwhelming job in classifying new values.

- Selection of the tree: The main goal of this process is to select the smallest tree that fits the data due to the reasons discussed in the pruning section.[25]

Gini index

To select a feature to split further we need to know how impure or pure that split will be. A pure leaf is the one that only has one class in all the dataset, being only 0 or only 1. A pure sub-split means that either you should be getting only "yes" or "no", 1 or 0. Usually the algorithm splits in two subsets but it could be possible to create more than two branches if necessary. However, this option won't be discussed in this study.

We will see what output we get after splitting by taking each feature as our root node. We basically need to know the impurity of our dataset and we'll take that feature as the root node which gives the lowest impurity or say which has the lowest Gini index. Mathematically Gini index can be written as:

$$Gini\ Index\ =\ 1\ -\ \sum_{i=1}^{n}\left(P_i\right)^2 = 1\ -\ [\left(P_+\right)^2 + \left(P_-\right)^2] \quad \text{(Eq.1)}$$

Where P+ is the probability of a positive class and P_ is the probability of a negative class.

Now we need to calculate the weighted Gini index that is the total Gini index of this split. This can be calculated by:

$$Weighted\ Gini\ Index\ = \frac{n_+}{N} * G_+ + \frac{n_-}{N} * G_- \quad \text{(Eq.2)}$$

Although Gini Index is one of the most used indexes for splitting, there is another one that's especially relevant, called Entropy. This one is again a statistical property that measures how well a given attribute separates the training examples according to their target classification. In the Figure 3 an attribute with high information gain (left) splits the data into groups with a high percentage of one of the classes, giving the user real information. Opposite to the right where the split barely allows us to make a decision base on the new subsets.

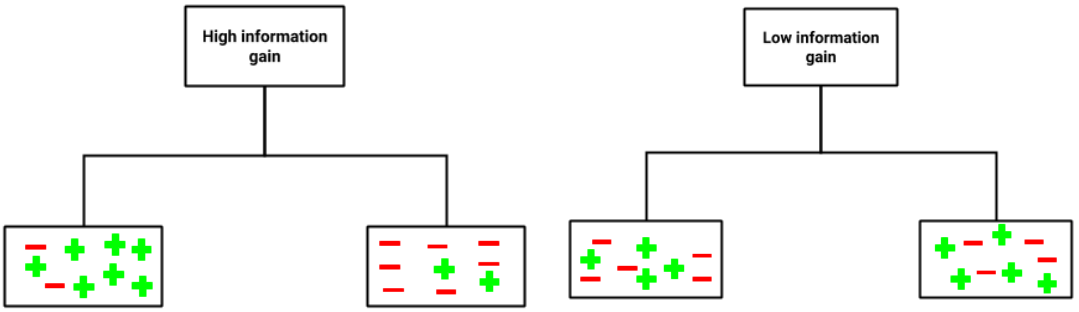

Figure 3: Distribution of points in case of high and low information gain.

Entropy measure the impurity in a group and it's defined as:

$$Entropy = \sum_{i=1}^{c} - p_i * log_2(p_i) \text{ (Eq. 3)}$$

Where p_i is the frequentist probability of a class I in our data.

Entropy is optimal when its value is 0. as it measures the disorder we will want a 0 disorder. Let's see on the next figure who it works for a better understanding.

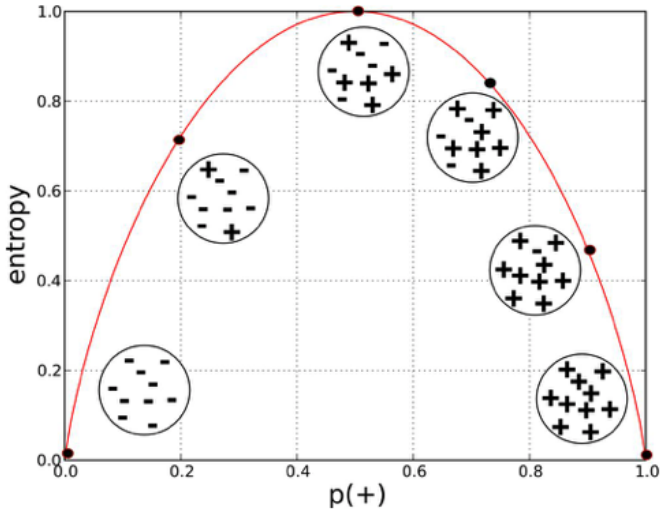

Figure 4: Representation of entropy.

Knowing now how the gain indexes are calculated, we need to explain that there are some versions of decision trees as ID3, C4.5, C5.0 or CART (Classification And Regression Trees). Although they work similar, they use different approaches to build the decision tree. In this paper ID3 will be explained because this one was one of the first approaches that appeared and it will be used in data analysis.

ID3

We are all set for the ID3 training algorithm. We start with the entire training data, and with a root. Then:

1. if the data-set is pure then
    0. construct a leaf having the name of the class
2. else
    0. choose the feature with the highest information gain
    1. for each value of that feature
        0. take the subset of the data-set having that feature value
        1. construct a child node having the name of that feature value
        2. call the algorithm recursively on the child node and the subset

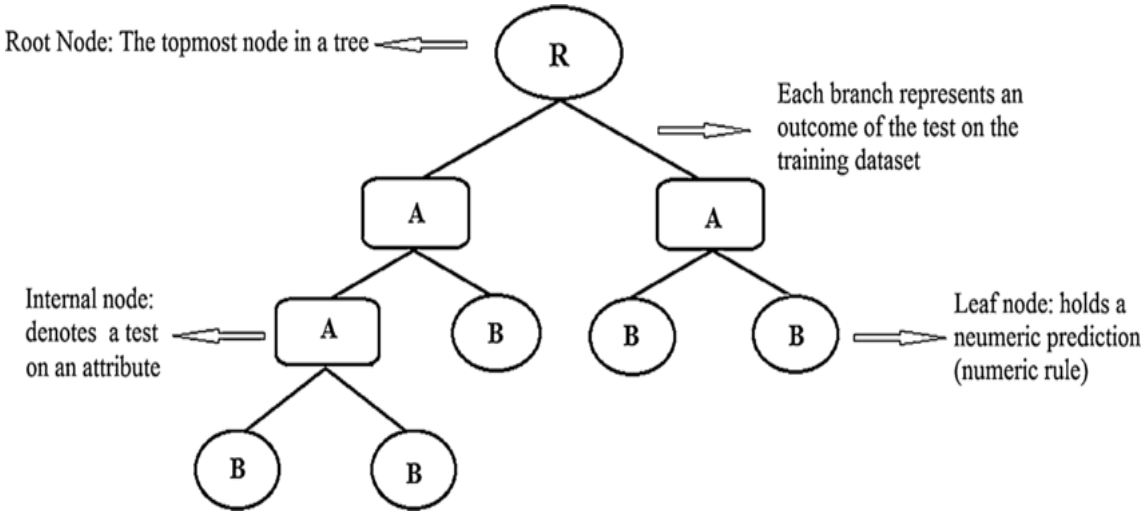

Figure 5: Branch, internal node and leaf node explanation

As we can see on Figure 5, there are three types of nodes: Root Node, that is the complete set of training data and will be split by the most representative feature. Internal node is a node that splits and Leaf node are those that don't split because they are pure, or because some other restriction has been applied, for example the depth of the tree. To simplify the tree and avoid overfitting it's possible to limit the depth of the tree, which is the number of levels that the tree has, on the Figure shown before the depth would be 4.

The depth of the tree concept leads us to the pruning concept, sometimes when a complete tree is created, understanding like a complete tree the one that has pure Leaf nodes or no more features for split, this tree tends to overfit the data, so it's really good with training data but not as good with test data. To solve that problem one of the possibilities is to prune the tree, this way we reduce the depth of the tree by transforming some Internal nodes into Leaf nodes.

Random Forest

Although Decision Tree allows the user to understand quite easily the data and its behaviour, DT has a huge dependency of the data we use to create it and it tends to overfit the training data, fact that produce a less accurate prediction when a new value is added. To solve that problem were developed ensemble algorithm based on Decision tree that outperform this one, giving excellent results to complex data sets.

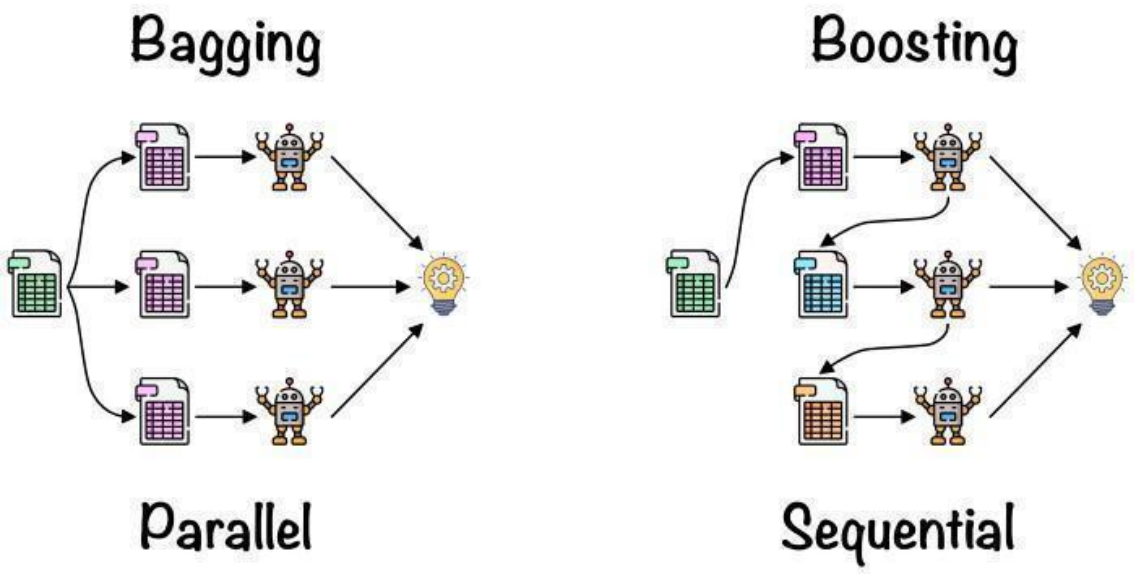

Figure 6: Difference between Bagging and Boosting

As we can see on figure 6 there are two main approaches to ensemble methods, Bagging and Boosting. While for bagging we use the majority vote, in the case of boosting the algorithm tries to fulfil what the previous tree or method hasn't been able to predict accurately, this way with every new tree the algorithm optimises the error by minimising it. Later, in this document it will be presented a method that uses boosting.

Random forest is a bagging method based on the idea that many opinions are better than just one. The method constructs many individual decision trees at training. Every tree takes a vote and then the most voted result is chosen, this process receives the name of bagging.

Random Forests can be used for either a categorical response variable, referred to as "classification", or a continuous response, referred to as "regression". Similarly, the predictor variables can be either categorical or continuous.

From a computational standpoint, Random Forests are appealing because they

- naturally handle both regression and (multiclass) classification;
- are relatively fast to train and to predict;
- depend only on one or two tuning parameters;
- have a built-in estimate of generalisation error;

- can be used directly for high-dimensional problems;
- can easily be implemented in parallel.

Statistically, Random Forests are appealing because of the additional features they provide, such as

- measures of variable importance;
- differential class weighting;
- missing value imputation;
- visualisation;

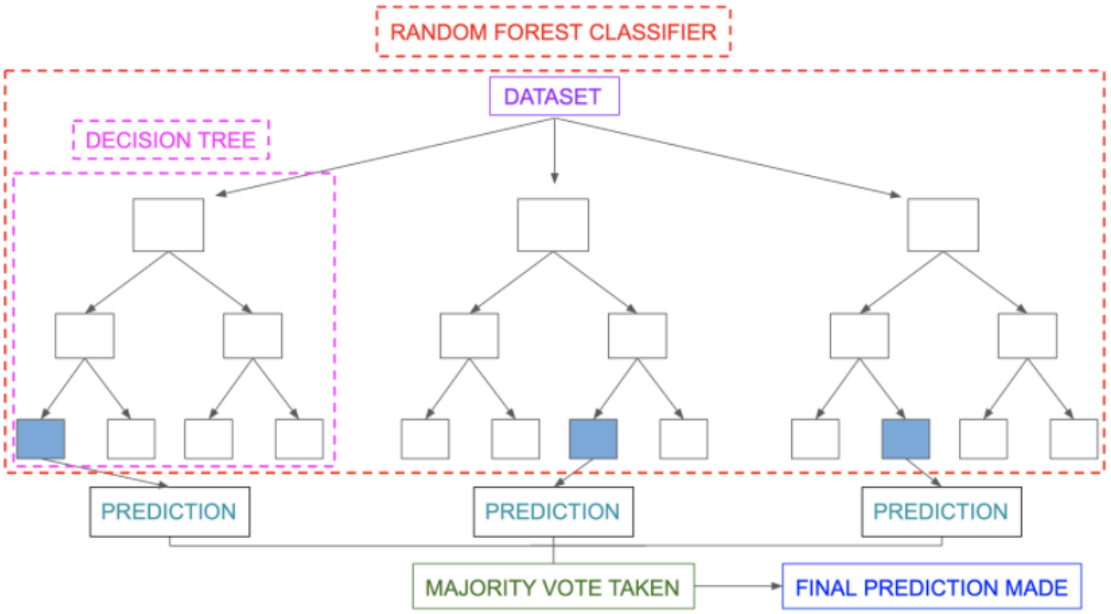

Figure 7: Random Forest classifier

One of the greatest benefits of a random forest algorithm is its flexibility. We can use this algorithm for regression as well as classification problems. It can be considered a handy algorithm because it produces better results even without hyperparameter tuning. Also, the parameters are pretty straightforward, they are easy to understand and there are also not that many of them.

One of the biggest problems in machine learning is Overfitting. We need to make a generalised model which can get good results on the test data too. Random forest helps to overcome this situation by combining many Decision Trees which will eventually give us low bias and low variance. To that we can add the main problem that presents the methods that use bagging, it is that for this method it is needed to suppose that all DT are independent but we you create all trees based on the same data this isn't really true so the user must be careful with what kind of data he is working.

Although we have presented Random Forest as a bagging method, Random Forest presents a major advantage against bagging method as the original conception. As in bagging, we build a determinate number of DT on bootstrapped training samples. But there comes the main difference, each time a split in a tree is considered, a random sample of m predictors is chosen. Typically, $m \approx \sqrt{p}$ is chosen.

This way the algorithm can't consider all the predictors when deciding which one fits the data better, this although not being really intuitive, has a really good explanation. If we had a very strong predictor in the data set, with other moderately strong predictors. Most of the trees created would have in the top split this one predictor, making those several trees really similar to each other, this lack of diversity on the forest would be against the main requisite of the random forest that is independence between trees and would make our model become less accurate. By limiting the number of predictors that are considered as an option on every split the model improves the diversity of the forest increasing this way the final performance. It ultimately leads to a reduction in both test error and OOB error over bagging.

This implementation will be significantly beneficial when having a large number of correlated predictors

The main limitation of random forest is that due to a large number of trees the algorithm takes a long time to train which makes it slow and ineffective for real-time predictions. In general, these algorithms are fast to train but quite slow to create predictions once they are trained. In most real-world applications, the random forest algorithm is fast enough but there can certainly be situations where run-time performance is important and other approaches would be preferred.

Customization

Ntree: Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.

Mtry: Number of variables randomly sampled as candidates at each split. Note that the default values are different for classification (sqrt(p) where p is number of variables in x) and regression (p/3)

Nodesize: Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time). Note that the default values are different for classification (1) and regression (5).

Max_depth: Maximum depth of a tree. The default value is 99.[26]

eXtreme Gradient Boosting

XGBoost is a scalable tree boosting model created by Tianqi Chen who implemented it on C++ and Tong He is the author of the R-package. This model has acquired interest due to its performance on Kaggle competitions.

It has shown to be easy to use, efficient, accurate and feasible. It needs a dense or sparse matrix as an input because this way it optimises the memory space and so on the processing time. The target variable has to be a numeric vector, using integers starting from 0 for classification, or real values for regression and the user can define the number of trees added to the model.

So, the objective of this model is regression or binary classification and in order to do that it used a weak classificatory model called Decision Tree that has been explained previously.

Now we are going to explain the model.

Suppose we have trees, the model is

$$\sum_{k=1}^{K} f_k \ (Eq.\ 4)$$

where each f_k is the prediction from a decision tree. The model is a collection of decision trees.

Having all the decision trees, we make prediction by

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \ (Eq.\ 5)$$

Where x_i is the feature vector for the -th data point.

Similarly, the prediction at the -th step can be defined as

$$\hat{y}_i^{(t)} = \sum_{k=1}^{K} f_k(x_i) \ (Eq.\ 6)$$

To train the model, we need to optimise a loss function, this encourages predictive models because fitting well in training data hopefully means being close to the underlying distribution.

Typically, we use

Rooted Mean Squared Error for linear regression

$$L = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \ (Eq.\ 7)$$

LogLoss for binary classification

$$L = \sum_{i=1}^{N} (y_i log(p_i) + (1 - y_i) log(1 - p_i) \ (Eq. 8)$$

mlogloss for multi-classification

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j} log(p_{i,j}) \ (Eq.\ 9)$$

Regularisation is another important part of the model. A good regularisation term controls the complexity of the model which prevents overfitting. Simple models tend to have smaller variance in future predictions, this makes prediction stable.

Define

$$\Omega = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \ (Eq.\ 10)$$

Where T is the number of leaves, and w_j^2 is the score on the -th leaf, said in another way the weight that has been assigned to the leaf.

Put loss function and regularisation together, we have the objective of the model:

$$Obj = L + \Omega \ (Eq.\ 11)$$

where loss function controls the predictive power, and regularisation controls the simplicity. In XGBoost, we use gradient descent to optimise the objective.

Given an objective $Obj\left(y, \hat{y}\right)$ to optimise, gradient descent is an iterative technique which calculate

$$\delta_{\hat{y}} Obj(y, \hat{y}) \ (Eq.\ 12)$$

at each iteration. Then we improve $\hat{y}$ along the direction of the gradient to minimise the objective.

For that we use Additive Training or Boosting, which are the same.

First, it starts from constant prediction, then a new function is added each time

$$\hat{y}_i^{(0)} = 0 \quad \hat{y}_i^{(1)} = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = \hat{y}_i^{(1)} + f_2(x_i)$$

...

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i) \ (\text{Eq. } 13)$$

Recall the definition of objective. For an iterative algorithm we can re-define the objective function as

$$Obj^{(t)} = \sum_{i=1}^{N} L(y_i, \hat{y}_i^{(t-1)}) + \sum_{i=1}^{t} \Omega(f_i) = \sum_{i=1}^{N} L(y_i, \hat{y}^{(t-1)}) + f_t(x_i)) + \sum_{i=1}^{t} \Omega(f_i) \quad (Eq.\ 14)$$

To optimise it by gradient descent, we need to calculate the gradient. The performance can also be improved by considering both the first and the second order gradient.

$$\delta_{\hat{y}_i^{(t)}} Obj^{(t)}$$

$$\delta_{\hat{y}_i^{(t)}}{}^2 Obj^{(t)} (Eq.\ 15)$$

Since we don't have derivative for every objective function, we calculate the second order Taylor

approximation of it

$$Obj^{(t)} \simeq \sum_{i=1}^{N} [L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t{}^2(x_i)] + \sum_{i=1}^{t} \Omega(f_i) \quad (Eq.\ 16)$$

where

$$g_i = \delta_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$h_i = \delta_{\hat{y}^{(t-1)}}{}^2 l(y_i, \hat{y}^{(t-1)}) \quad (Eq.\ 17)$$

Remove the constant terms, we get

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t{}^2(x_i)] + \Omega(f_t) \quad (Eq.\ 18)$$

This is the objective at the -th step. Our goal is to find a f_t to optimise it.

After the mathematical explanation on the next Figure, we can see how this algorithm works in a visual way.

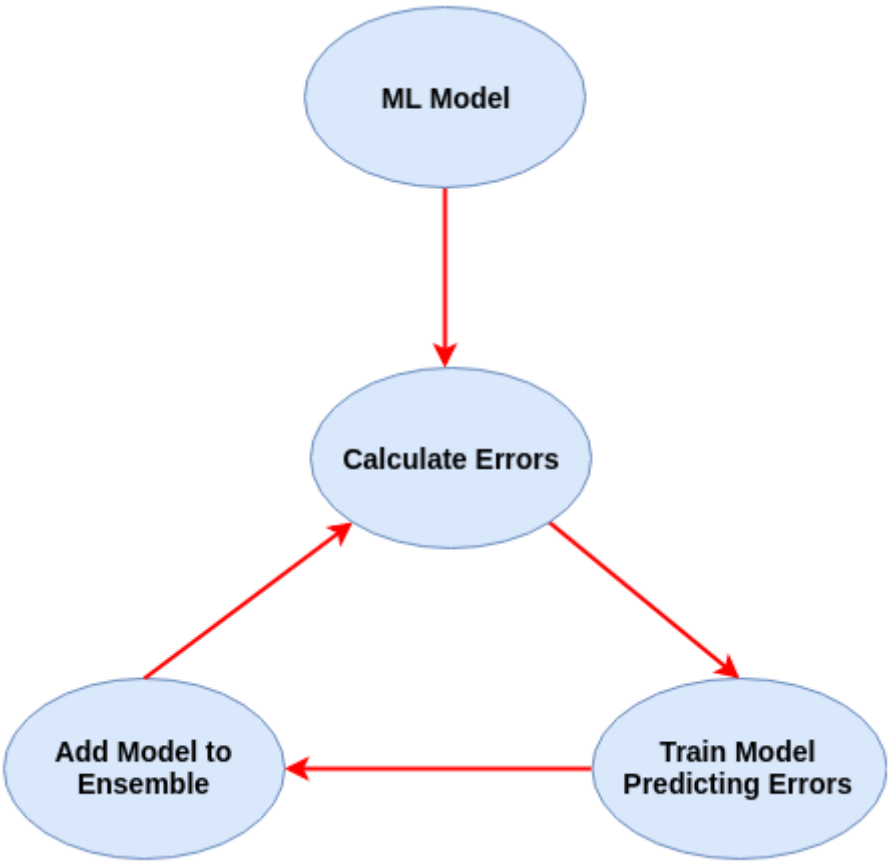

Figure 8: Representation of the algorithm's operation

As we can observe on the Figure it's an iterative algorithm that finalizes when some restriction is achieved.

Parameters customization

In order to optimise the xgboost algorithm the R package allows the user to change some parameters, adapting the algorithm to the type of data that has to be used.

Now they will be introduced and explained:

- Nthread
  - Number of parallel threads used to run XGBoost.
- Booster
  - Gbtree
  - Gblinear
- Eta:
  - Step size shrinkage used in update to prevent overfitting. We get the weights of new features on every boosting step, eta shrinks the feature weights so the process becomes more conservative. Range: [0.1] Default: 0.3
- Gamma

    - Minimum 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. Range: [0.1] Default: 0
- Max_depth
    - Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree. The exact tree method requires a non-zero value. Default: 6
- Min_child_weight
    - Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. The larger min_child_weight is, the more conservative the algorithm will be. Range: [0.inf] Default: 1
- Max_delta_step
    - Maximum delta step we allow each leaf output to be. If the value is set to 0. it means there is no constraint. If it is set to a positive value, it can help make the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced. Set it to a value of 1-10 might help control the update. Range: [0. inf] Default: 0
- Subsample
    - Maximum delta step we allow each leaf output to be. If the value is set to 0. it means there is no constraint. If it is set to a positive value, it can help make the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced. Set it to a value of 1-10 might help control the update. Range: (0.1] Default: 1
- Colsample_bytree*
- Lambda
    - L2 regularisation term on weights. Increasing this value will make the model more conservative. Default: 1
- Alpha
    - L2 regularisation term on weights. Increasing this value will make the model more conservative. Default: 0
- Lambda_bias
- scale_pos_weight [default=1]
    - Control the balance of positive and negative weights, useful for unbalanced classes. A typical value to consider: sum(negative instances) / sum(positive instances).

Evaluation metrics

As with every hypothesis, it's necessary to check the performance of the model to know if the model created it's good at predicting the variable of study. To do that there are some parameters that can be used to check the performance, every parameter is convenient in some scenarios so choosing the correct parameter to evaluate the accomplishment of the model it's critical to avoid misinterpretations of the good or bad behaviour of the model.

Among other parameters are presented below.

- Rmse
  - For regression
- Logloss
  - For classification
- Error
  - Binary classification error rate. It is calculated as #(wrong cases)/#(all cases). For the predictions, the evaluation will regard the instances with prediction value larger than 0.5 as positive instances, and the others as negative instances.
- Auc (Area under the curve)
  - For a predictor f an unbiased estimator of its AUC can be expressed by the following Wilcoxon-Mann-Whitney statistic:

$$AUC(f) = \frac{\sum_{t_0 \in \mathcal{D}^0} \sum_{t_1 \in \mathcal{D}^1} \mathbf{1}[f(t_0) < f(t_1)]}{|\mathcal{D}^0| \cdot |\mathcal{D}^1|} \quad (Eq.\ 19)$$

    where, $1\left[f\left(t_0\right) < f(t_1)\right]$denotes an indicator function which returns 1 if $f\left(t_0\right) < f\left(t_1\right)$ otherwise return 0; D^0 is the set of negative examples, and $D^1$is the set of positive examples.
    The AUC is related to the "Gini coefficient" (G_1) by the formula

$$G_1 = 2AUC - 1$$

$$G_1 = \sum_{k=1}^{n} (X_k - X_{k-1})(Y_k + Y_{k-1}) \quad (Eq.\ 20)$$

- Merror
  - Multiclass classification error rate. It is calculated as #(wrong cases)/#(all cases).
- mlogloss (Log loss, aka logistic loss or cross-entropy loss)
  - This is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of a logistic model that returns *y_pred* probabilities for its training data *y_true*. The log loss is only defined for two or more labels. For a single sample with true label $y \in \{0.1\}$ and a probability estimate $p = Pr(y = 1)$, the log loss is:

$$L_{log}(y, p) = -(y\ log(p) + (1 - y)\ log(1 - p)) (Eq.\ 21)$$

Data source

For this study a data set from a Kraggle competition placed 3 years ago has been selected. The data set is presented as follows on the web.

Context

"Predict behaviour to retain customers. You can analyse all relevant customer data and develop focused customer retention programs."

Content

The Orange Telecom's Churn Dataset, which consists of cleaned customer activity data (features), along with a churn label specifying whether a customer canceled the subscription, will be used to develop predictive models. Two datasets are made available here: The churn-80 and churn-20 datasets can be downloaded.

The two sets are from the same batch, but have been split by an 80/20 ratio. As more data is often desirable for developing ML models, let's use the larger set (that is, churn-80) for training and cross-validation purposes, and the smaller set (that is, churn-20) for final testing and model performance evaluation.

The main were two main points for selecting this data set:

- First, it’s usability, a parameter calculated by Kaggle to represent the level of documentation of the data set. This parameter is a measure of the quality of the data set.
- Second, we were looking for a data set as clean as possible as it is not the objective of the thesis to work a lot on the cleaning process as it is something that on real projects takes most of the time and wastes a lot of time. This data set was almost ready to be used and still some modifications in order to make the different models were made.
- And third, we wanted a two factor response as it was desired to make a categorical predictor. The variable “Churn” was adequate for this matter and gave us an extra point of complexity because it is unbalanced.

Data clean

First, the data set is going to be described and the main transformations will be explained although low level information will be better described on the code, so for exhaustive understanding of the cleaning process it is recommended to read the code directly which has many commentaries in order to facilitate the understanding of almost every line wrote.

Dataset TRAIN has 19 variables and 2666 rows and TEST has 19 variables and 667 rows.

The variables are presented on table 1.

On the following table it's shown the original type of the different variables and after transformation. This will be important because later it will be explained that XGBoost only accepts a matrix type as input so every variable that isn't an integer or a float will need a transformation and codification to be incorporated to the study.

Table 1: Data types

Variable	**Type on the original dataset**	**Type after transformation**
State	Character	Factor with 51 levels
Account.length	Integer	Integer
Area.code	Integer	Factor with 3 levels
International.plan	Character	Factor with 2 levels
Voice.mail.plan	Character	Factor with 2 levels
Number.vmail.messages	Integer	Integer
Total.day.minutes	Numeric	Numeric
Total.day.calls	Integer	Integer
Total.day.charge	Numeric	Numeric
Total.eve.minutes	Numeric	Numeric
Total.eve.calls	Integer	Integer
Total.eve.charge	Numeric	Numeric
Total.night.minutes	Numeric	Numeric
Total.night.calls	Integer	Integer
Total.night.charge	Numeric	Numeric
Total.intl.minutes	Numeric	Numeric
Total.intl.calls	Integer	Integer
Total.intl.charge	Numeric	Numeric

Customer.service.calls	Integer	Integer
Churn	Character	Factor with 2 levels

As mentioned before, cleaning the data will consist of codification of State, International.plan, Voice.mail.plan, Area.code and Churn as those are Characters on the given dataset.

Exploratory data analysis

In order to have a better understanding of the data an exploratory data analysis is performed. Firstly, with general information obtained with the summary command on the total set. The total set is the union of the two given datasets TEST and TRAIN. The objective is to obtain a general vision of the data and to detect any anomaly that could affect the machine learning algorithm applied after.

```
Total.day.charge Total.eve.minutes Total.eve.calls Total.eve.charge
Min.   : 0.00    Min.   :  0.0     Min.   :  0.0   Min.   : 0.00
1st Qu.:24.43    1st Qu.:166.6     1st Qu.: 87.0   1st Qu.:14.16
Median :30.50    Median :201.4     Median :100.0   Median :17.12
Mean   :30.56    Mean   :201.0     Mean   :100.1   Mean   :17.08
3rd Qu.:36.79    3rd Qu.:235.3     3rd Qu.:114.0   3rd Qu.:20.00
Max.   :59.64    Max.   :363.7     Max.   :170.0   Max.   :30.91

Total.night.minutes Total.night.calls Total.night.charge
Min.   : 23.2       Min.   : 33.0     Min.   : 1.040
1st Qu.:167.0       1st Qu.: 87.0     1st Qu.: 7.520
Median :201.2       Median :100.0     Median : 9.050
Mean   :200.9       Mean   :100.1     Mean   : 9.039
3rd Qu.:235.3       3rd Qu.:113.0     3rd Qu.:10.590
Max.   :395.0       Max.   :175.0     Max.   :17.770

Total.intl.minutes Total.intl.calls Total.intl.charge
Min.   : 0.00      Min.   : 0.000   Min.   :0.000
1st Qu.: 8.50      1st Qu.: 3.000   1st Qu.:2.300
Median :10.30      Median : 4.000   Median :2.780
Mean   :10.24      Mean   : 4.479   Mean   :2.765
3rd Qu.:12.10      3rd Qu.: 6.000   3rd Qu.:3.270
Max.   :20.00      Max.   :20.000   Max.   :5.400
```

Figure 9: Summary of numeric variables

Figure 9 shows the statistics of numeric values as it can be seen Median and Mean are quite similar. That means that data are centred and follow a normal distribution in most cases. To check the normal distribution of the data, histograms are plotted next for continuous variables.

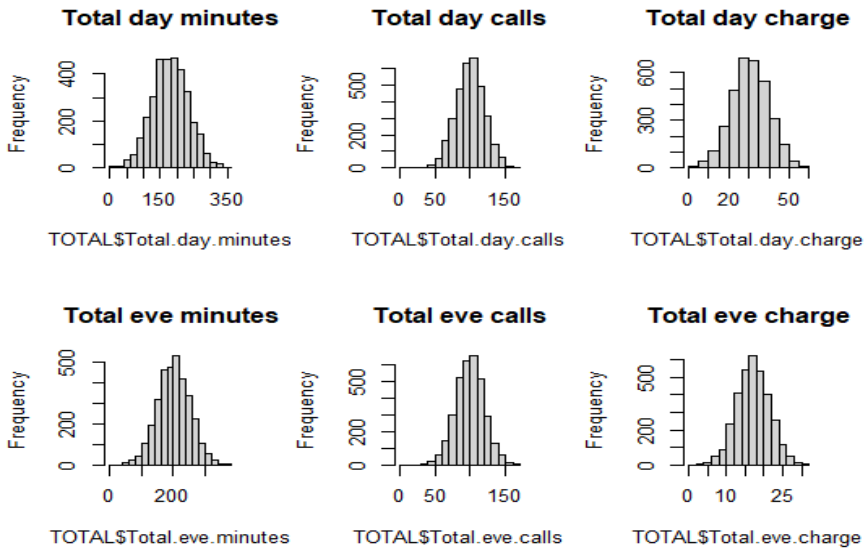

Figure 10: Day and Eve histograms

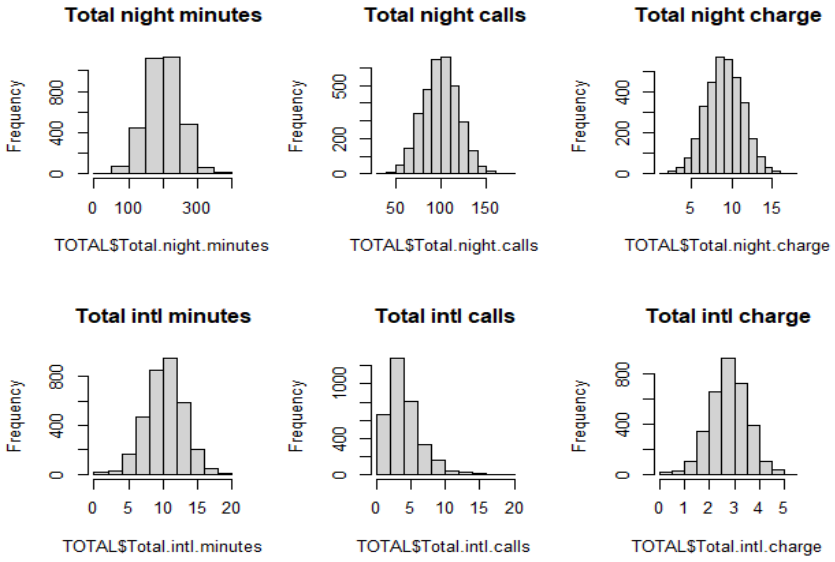

Figure 11: Night and Intl histograms

Another important information that it's crucial to know is if there is any correlation between the different variables. Next it can be shown the detected correlation between minutes and charge for the four categories Day, Eve, Night and Intl. The "Minutes" contain the time spent on calls and "Charge" the daily amount of money paid. It's normal that those variables are related so this will have to be taken into account when doing the different models as the Decision Tree algorithm works on the basis that the variables aren't correlated or the correlation is weak.

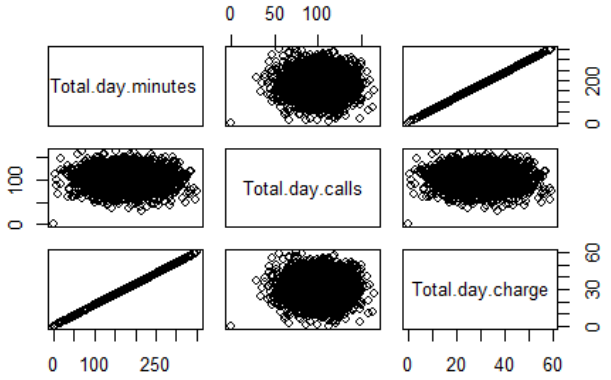

Figure 12: Correlation between day.minutes and day.charge

Above it (Figure 13) can be seen the correlation between the variables mentioned before and that there is no other correlation between the rest of the variables. All this is reflected on the Correlation Matrix shown above. Notice that correlation matrix only shows the numeric variables.

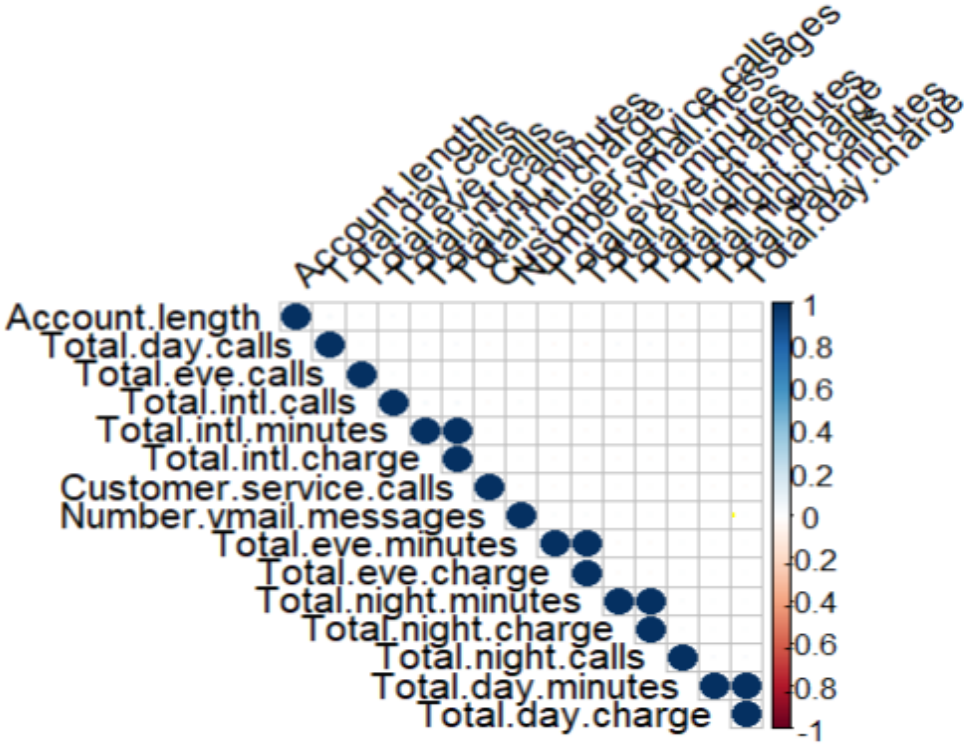

Figure 13: Correlation matrix

After studying the numeric variables the categorical ones are presented now on the Figure above(Figure 14).

Area.code	International.plan	Voice.mail.plan	Churn
408: 838	No :3010	No :2411	False:2850
415:1655	Yes: 323	Yes: 922	True : 483
510: 840			

Figure 14: Categorical variables statistics

State variable is defined as the State where the client lives so there are 51 levels corresponding to the 51 States of America. On the next table it's shown the distribution and after that it's plotted.

Table 2: Distribution by States

AK	AL	AR	AZ	CA	CO	CT	DC	DE	FL	GA	HI	IA	ID	IL	IN	KS	KY
52	80	55	64	34	66	74	54	61	63	54	53	44	73	58	71	70	59
LA	MA	MD	ME	MI	MN	MO	MS	MT	NC	ND	NE	NH	NJ	NM	NV	NY	OH
51	65	70	62	73	84	63	65	68	68	62	61	56	68	62	66	83	78
OK	OR	PA	RI	SC	SD	TN	TX	UT	VA	VT	WA	WI	WV	WY			
61	78	45	65	60	60	53	72	72	77	73	66	78	106	77			

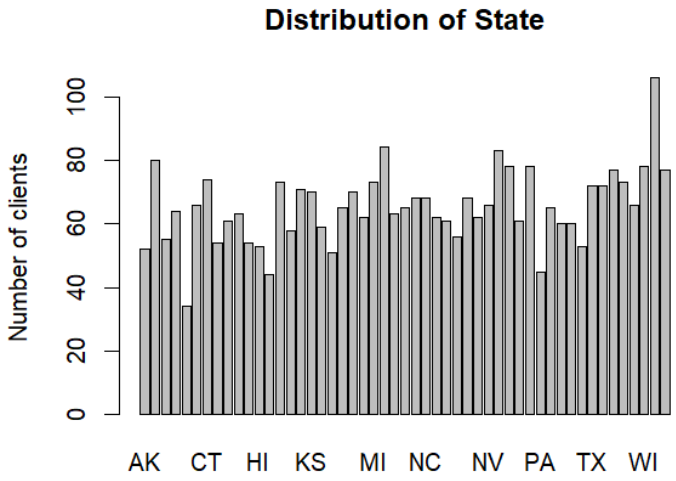

Figure 15: Histogram of State

In this study the variable response is Churn, defined as true when a client leaves the company and false when it stays. As it is the variable response it's significantly important to check it closely. In this case with Figure 16 it can be detected that the variable isn't balanced, meaning that the dataset has many values as "False" and only a few as "True". The best scenario is working with balanced data but in this case some procedures will be applied in order to work with the unbalanced data without having a bad performance because of it.

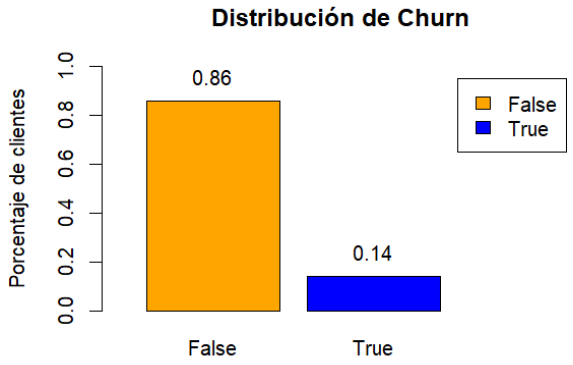

Figure 16: Churn distribution

Unbalanced data

Sometimes when facing machine learning problems, we will find different types of data set. Unbalanced data is one of them. First, we will start by explaining what it is. When you collect data most of the time it would be ideal to have homogeneous data. If we collect data from gender a 50/50 range or if we collect information of what political party prefers some people it would be ideal to have, if there were 5 options, 20% for each party. But as we know the world isn't balanced so this ideal won't be possible many times. On the opposite, we find for example information about cancer, as we know, most people don't get cancer, only a small percentage of the population, like this we could find many other examples. The point is that the last example it's a clear, an extreme example of what unbalanced data is. However, we don't need to get to that extreme to consider a set unbalanced; it would be enough to have a 1:4 proportion to consider it unbalanced.

What is the problem then? Sometimes when trying to make a model that fits the data set it will be found that this one matches 80% of the predictions. At first look, we could consider that as a good model as it has a good percentage of success but what if I told you that 80% of the data belong to one category and 20% to the other? That's what happens when you have unbalanced data, the model gets lazy, it decides that all the predictions will be 1 or 0 and as we have 80% of the data of that category it will succeed most of the time. However, we can clearly see that it isn't a good model, to fight against this event there are some strategies that can be used.

Collecting data

In some cases, unbalanced data, it's just an expression of bad luck when collecting data, so if it's easy, collecting more data could solve the problem without complicated strategies.

Changing performance metric

If we have unbalanced data, Accuracy won't be the best way to check the real performance of the model. Instead, we could use other metrics as:

- Confusion Matrix: A breakdown of predictions into a table showing correct predictions (the diagonal) and the types of incorrect predictions made (what classes incorrect predictions were assigned).
- Precision: A measure of a classifier exactness.
- Recall: A measure of a classifier completeness
- F1 Score (or F-score): A weighted average of precision and recall.

Although they are all interesting, we will focus on Confusion Matrix as this performance metric has been used on the analysis object of this study.

For a binary classification problem, the table has 2 rows and 2 columns. Across the top are the observed class labels and down the side are the predicted class labels. Each cell contains the number of predictions made by the classifier that fall into that cell.

Table 3: Confusion Matrix

		Actual Class	
		Positive	Negative
Predicted	Positive	True Positive (1)	False Positive (3)
	Negative	False Negative (2)	True Negative (4)

From this table we can obtain some metrics that will allow us to identify the correct performance of the model. For that, most of the time we will need to focus on some of the metrics as it isn't impossible to optimise them all at the same time. Later it will be explained which metric it's better in which scenarios.

- Type I error
- Type II error
- F1
- Error rate
- Accuracy
- Sensitivity or Recall
- Specificity
- Precision

Type I error

It's calculated as shown in the next formula.

$$Type\ I\ error\ = \frac{False\ Positive}{True\ Negative+False\ Positive} = \frac{(3)}{(4)+(3)} \text{ (Eq. 22)}$$

Type II error

It's calculated as shown in the next formula.

$$Type\ II\ error\ = \frac{False\ Negative}{True\ Positive+False\ Negative} = \frac{(2)}{(1)+(2)} \text{ (Eq. 23)}$$

F1 Score

The traditional F measure is calculated as follows:

$$F1\ Score = \frac{2*Precision*Sensitivity}{Precision+Sensitivity} \text{ (Eq. 24)}$$

This is the harmonic mean of the two fractions. This is sometimes called the F score and might be the most common metric used on imbalanced classification problems.

Error rate

It's calculated as shown in the next formula.

$$Error\ rate = \frac{False\ Positive+False\ Negative}{True\ Negative+False\ Positive+False\ Negative+True\ Negative} = \frac{(3)+(2)}{(4)+(3)+(2)+(1)} \text{ (Eq.25)}$$

Accuracy

With the Error Rate we can calculate accuracy as 1-Error Rate. This metric is useful when working with balanced data, with symmetric datasets, the false negatives and false positives have similar costs.

Sensitivity or Recall

It's calculated as shown in the next formula.

$$Sensitivity = \frac{True\ Positive}{False\ Negative+True\ Positive} = \frac{(1)}{(2)+(1)} \text{ (Eq. 26)}$$

Sensitivity is a good metric when identifying positives is essential, we can't afford the occurrence of False Negatives. This is the example of detection of diseases, it's critical to detect it as soon as possible even if that means to predict more False Positives.

Specificity

It's calculated as shown in the next formula.

$$Specificity = \frac{True\ Negative}{True\ Negative+False\ Positive} = \frac{(4)}{(4)+(3)} \text{ (Eq. 27)}$$

Specificity is the best indicator when you want to cover all true negatives. It's used when raising a false alarm (False Positive) is worse than a False Negative. This would be the case of an alcoholaemia test after a deadly accident.

Precision

It's calculated as shown in the next formula.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{(1)}{(1)+(3)} \quad \text{(Eq. 28)}$$

Precision allows you to be confident of the predicted positives, it's used when the occurrence of false positives is unacceptable. An example would be spam mail filtering, as it isn't important if some spam mails arrive to the user, most of them would be filtered.

When analysing the metric shown before most times, we would like to optimise Sensitivity and Specificity at the same time, being 1 the optimal value. Said in another way it allows us to measure the True Positives Rate and the False Positives Rate. The goal is to analyse the predictive power in ensuring the detection of as many true positives as possible while minimizing false positives. This way it's possible to analyse the classifier behaviour over the full range of thresholds.

To draw the curve, we use sensitivity as y-axis and 1-specificity as x-axis so the area under the curve will give us the evaluation of the performance, being 1 the perfect match.

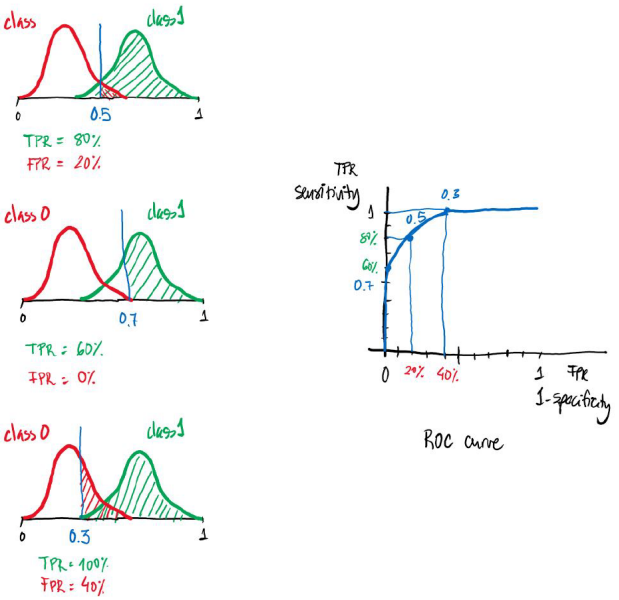

Figure 17: ROC Curve

As it can be observed on the next Figure 18 the real distribution of Positives and Negatives don't affect the performance of this metric, as we can see on the Figure a ROC Curve close to the diagonal indicates a poor performance of the predictor.

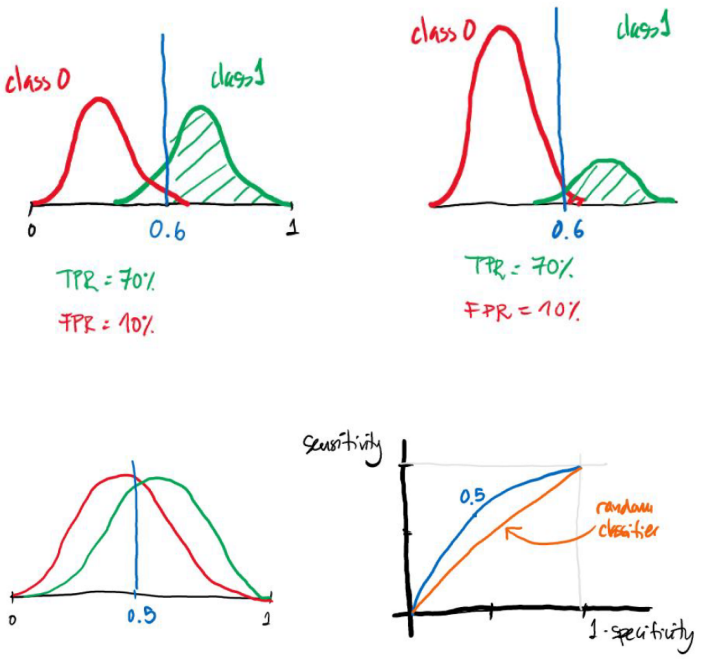

Figure 18: ROC Curve 2

Resampling the dataset

Another option is changing the dataset to have more balanced data, this is called sampling the dataset and there are main methods that can be used to even-up the classes.

- Adding copies of instances from the under-represented class called over-sampling or sampling with replacement.
- You can delete instances from the over- represented class, called under-sampling.

As those are easy to implement it would be advisable to do both options and compare them.

Some rules of Thumb

- If you have more than tens or hundreds of thousands of instances, then consider under-sampling.
- If you have less than tens or hundreds of thousands of instances, then consider under-sampling.
- Consider testing different resampled ratios, it's not necessarily a 1:1 ratio always.

Random Forest performance

After explaining the different algorithms and their parameters, a discussion about the performance of the model depending on each parameter is presented in order to choose the best value of each parameter and make the best model. Later it will be compared with a default model and with the other algorithm. For every parameter, three gràfics will be presented and discussed. First, one will show Specificity and Sensitivity. Second one, class errors and last one Geometrical mean and F1

Mtry

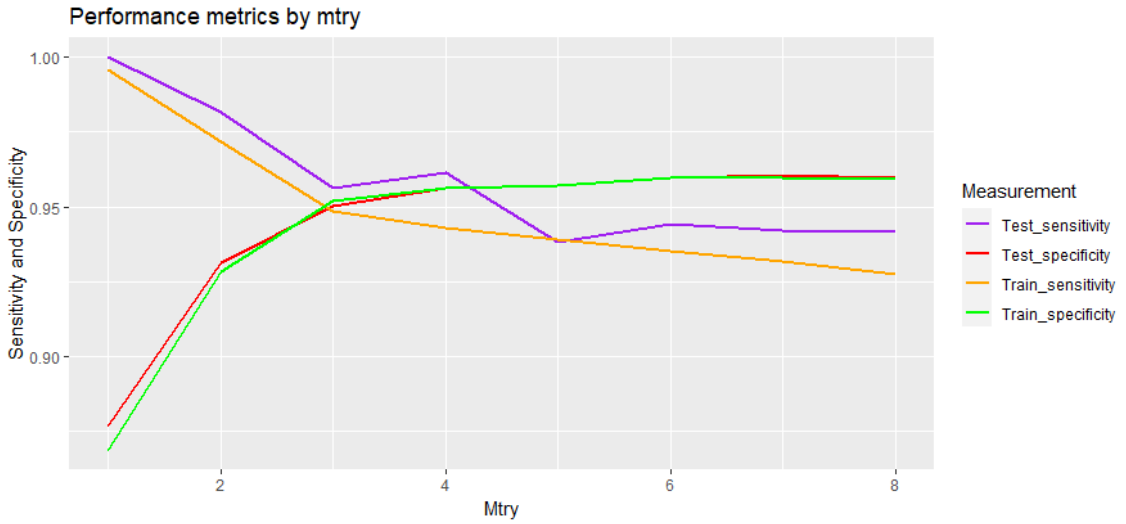

Figure 19: Performance metrics by mtry. purple: Sensitivity test, red: Specificity test, green: Sensitivity train, orange: Specificity train

On the Figure 19 we can observe how specificity increases his performance until mtry=4 where it stabilises. On the other hand, sensitivity decreases, this is because the model it's giving up sensitivity to increase specificity. These two performance metrics are complementary so it's hard to increase one without making the other decrease. In the case of study, as it has only few positive values against many negatives, specificity should be the parameter that gives us a better idea of the real performance of the model. Observing only these indicators the best option is defining mtr=4 because increasing its value more makes sensitivity decreases more than the increasing of specificity.

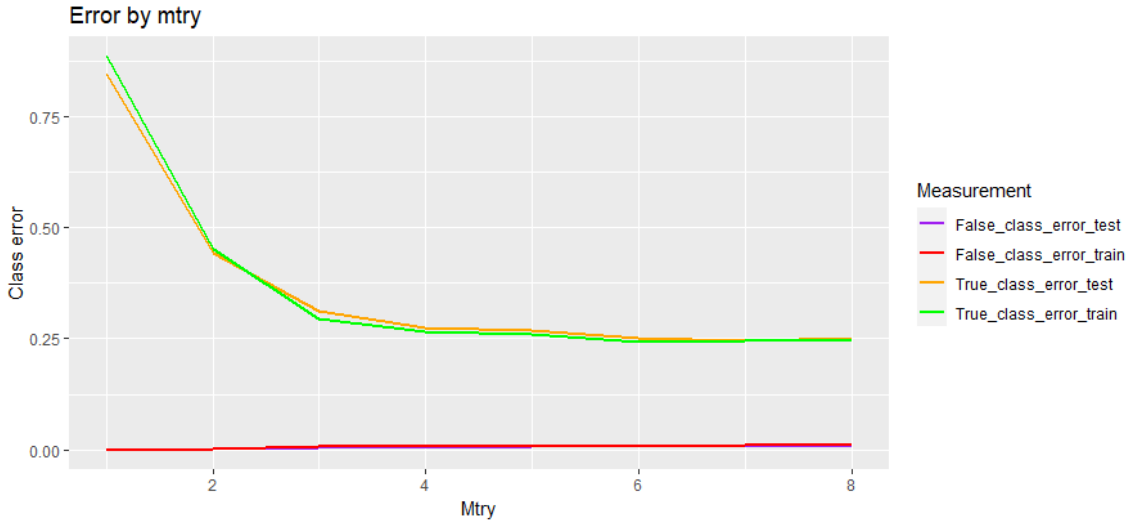

Figure 20: Error by mtry. purple: Error rate, red: False class error test, green: True class error test, orange: False class error train, blue: True class error train

Error class error is calculated as those true or false predicted that aren't true or false divided by total amount of trues and falses predicted. This shows the main problem with unbalanced data, as the study has many values that are negative, random forest tends to give a negative result most of the time as the percentage of false/true is 86/14. so only by predicting always false we get an accuracy of 86%. However this algorithm doesn't work because it is unable to predict true values, in our case the main reason for developing this algorithm. Mtry =4 seems to give the best balance between these indicators.

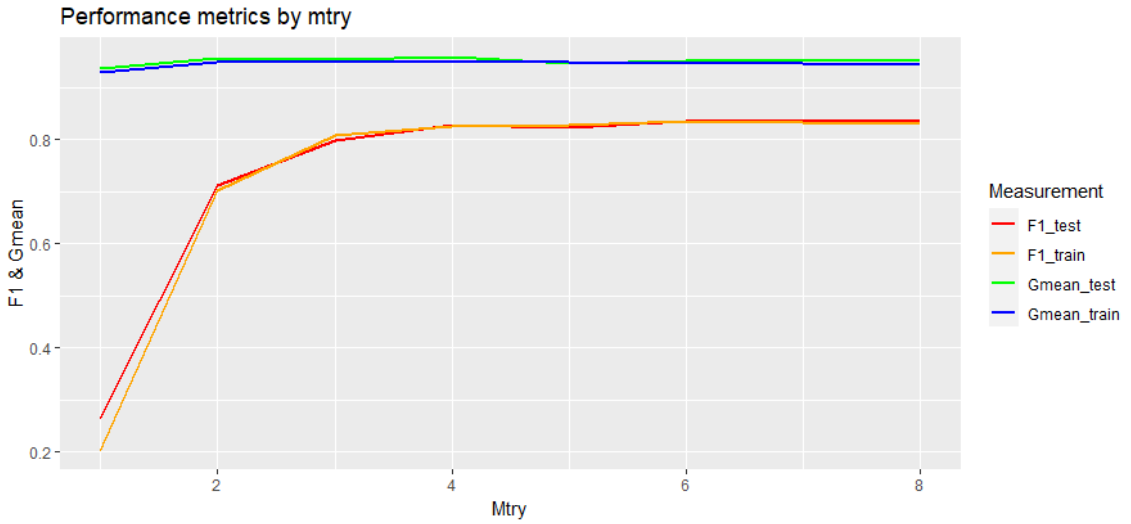

Figure 21: Performance metrics by mtry. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train

Although the parameters discussed before give us information the best parameters to decide are F1 and Geometric mean. In this project both will be plotted but ultimately, the decision will be based on the F1 measure of the test because training data could be overfitted. Looking at Figure 21 we find that we have a maximum at mtry equal to 4 and 6 for F1_test and

Gmean_test. The main tendency is quite similar to previous plots as all indicators are related, with mtry bigger than 4 F1 has a good performance over 82%.

Ntree

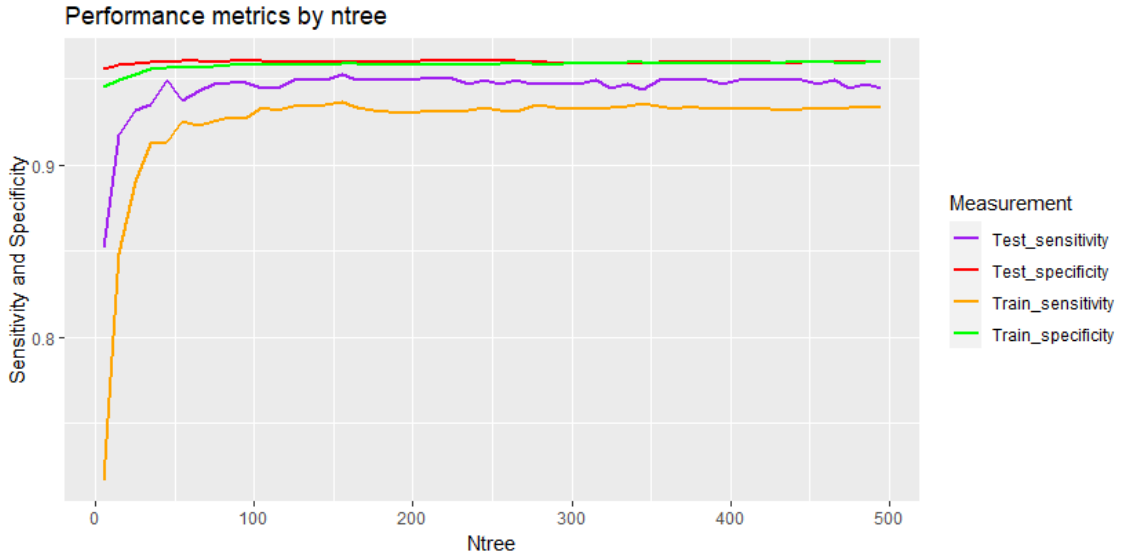

Figure 22: Performance metrics by ntree. purple: Sensitivity test, red: Specificity test, green: Sensitivity train, orange: Specificity train

In the figure above it can be observed that increasing the number of trees increases both sensitivity and specificity, being specificity the indicator that has an exponential improvement on low values until it stabilises. Ntree is a parameter that makes random forest time of calculation increase as the number of trees so in order to keep the time performance low it's better to keep this parameter as low as possible. Observing the plot a good value for this parameter may be between 80 and 150.

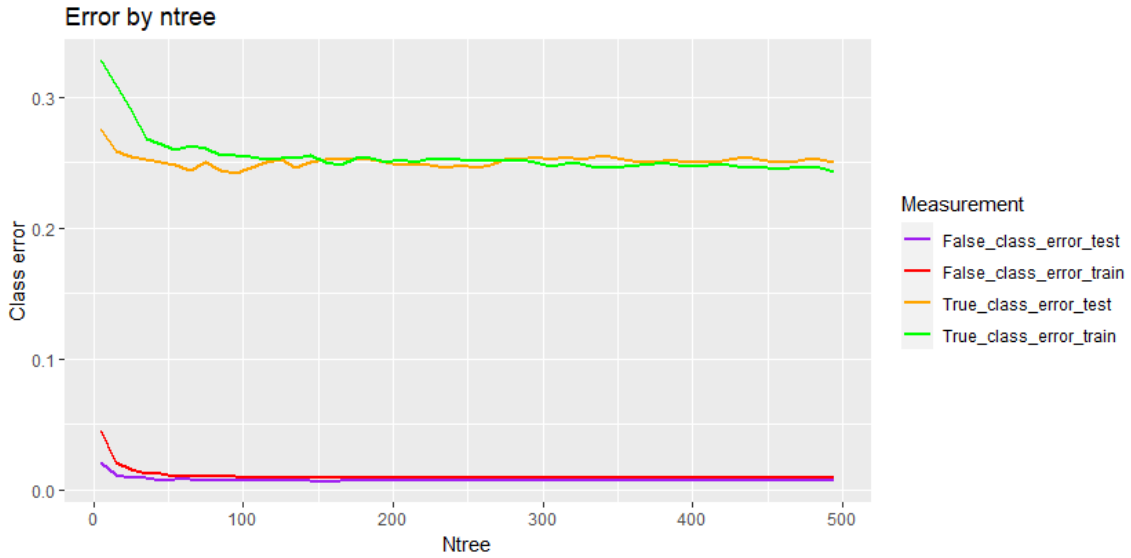

Figure 23: Error by ntree. Purple: False class error test, orange: True class error test, red: False class error train, green: True class error train

On Figure 23 we observe that the performance with low values of ntree is quite worse however it goes from a 35% to a 25% of error on true class just be designating to ntree a value higher than 100. Training and test evaluation are quite similar and as we mentioned before unbalanced data makes the true performance always worse. This indicator does something that's usually difficult, it improves both indicators at the same time. So choosing the correct value is the key to a good performance, always taking care to not compromise time's performance.

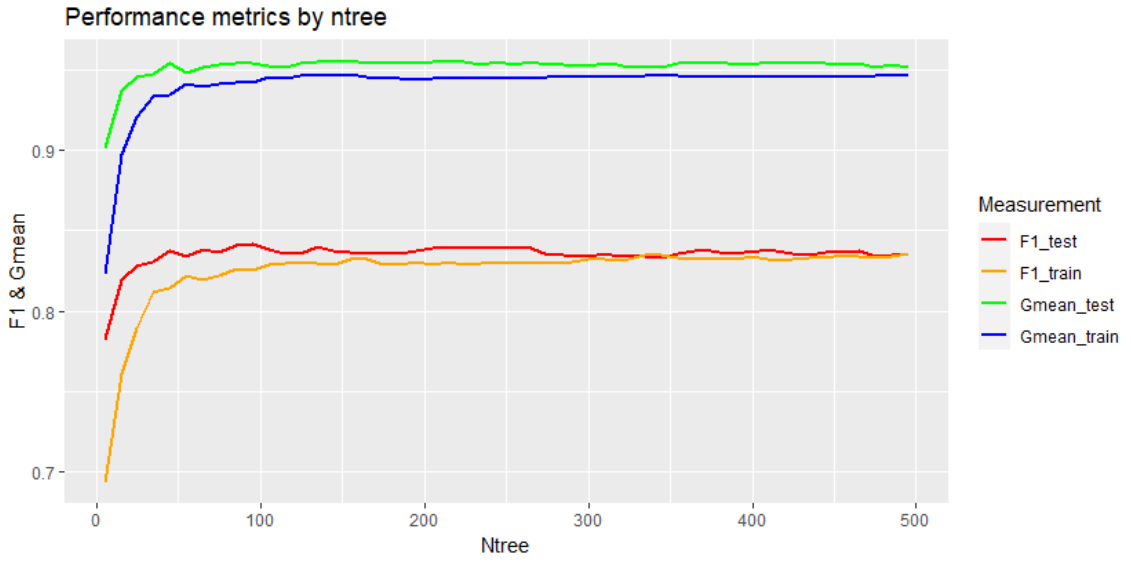

Figure 24: Performance metrics by ntree. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train

As in previous plots, results get better just increasing until 50. we notice that Geometrical mean has similar improvement with F1. However there is a gap between those indicators, with F1 score lower by more than 10 %.

Nodesize

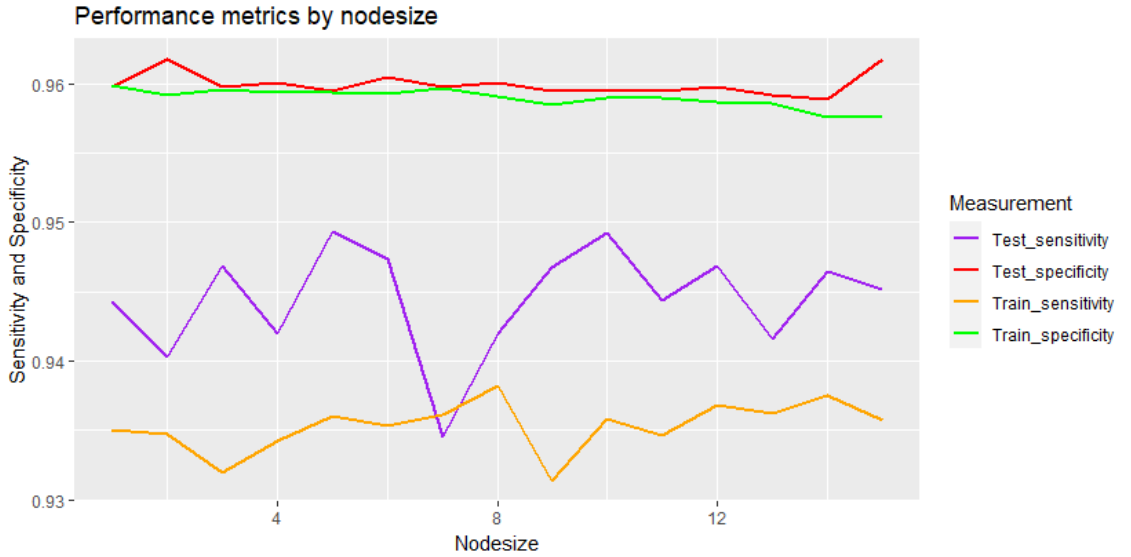

Figure 25: Performance metrics by nodesize. purple: Accuracy, red: Sensitivity test, green: Specificity test, orange: Sensitivity train, blue: Specificity train

In the case of nodesize, we can appreciate that increasing or decreasing the value doesn't affect the model performance as the performance metrics values stay stable for all the range. It can be noticed that nodesize=7 has a strange value where test and train specificity lines cross but this can be explained by the variance of the model calculations. The other factor is the range of every line which as it can be observed, it's quite small, a fact that agrees with the idea that in this case of study the parameter nodesize can be omitted as it has no effect on the model.

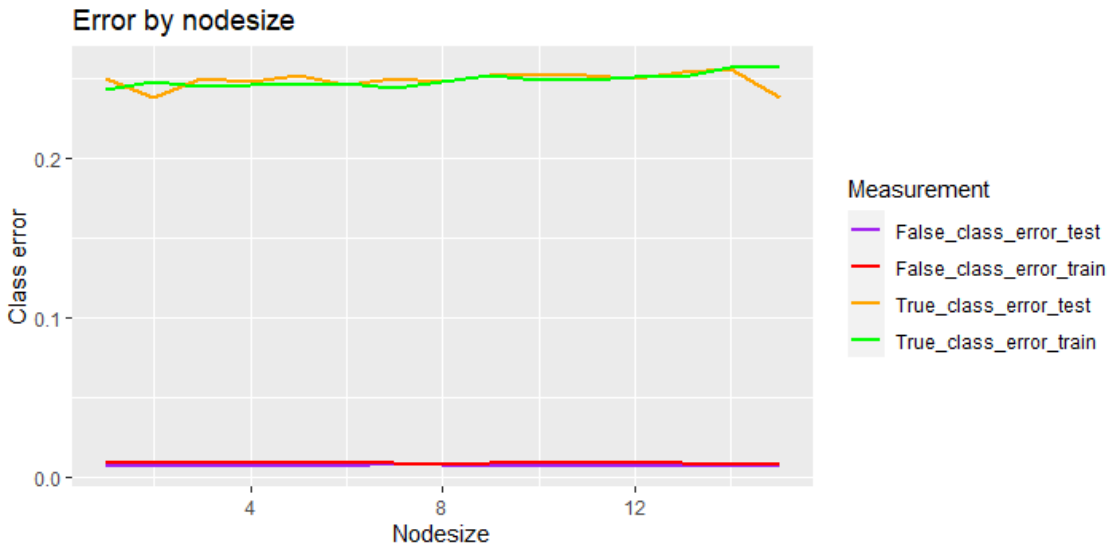

Figure 26: Error by nodesize. Purple: False class error test, orange: True class error test, red: False class error train, green: True class error train

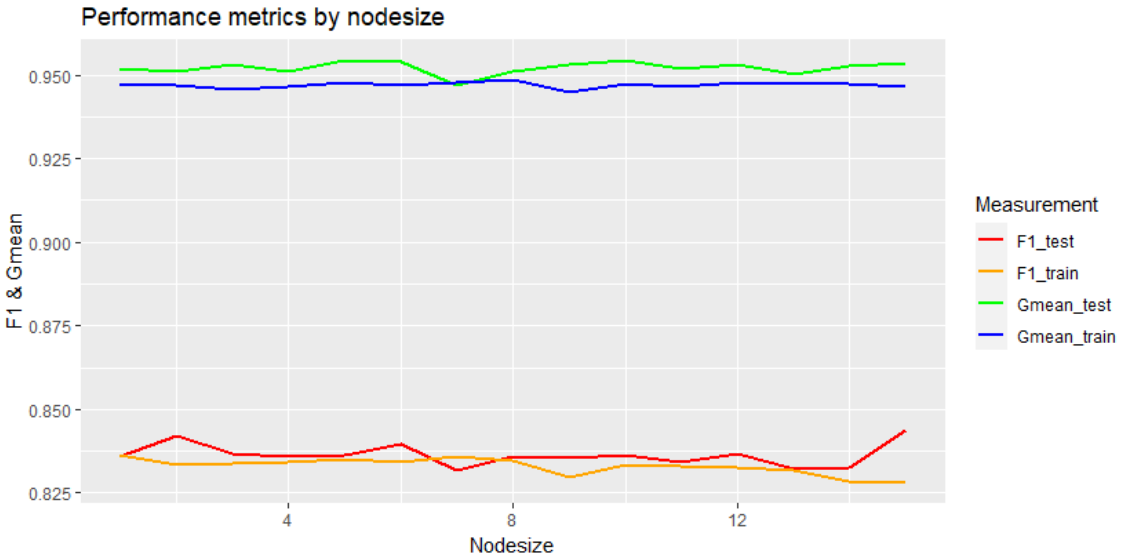

Figure 27: Performance metrics by nodesize. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train

As mentioned before, all parameters are stable for different values of the nodesize parameter so there is no point in choosing a determined parameter. This parameter helps us understand

that not every parameter has always a big impact on the performance, so part of understanding the customization process is to choose the parameters that have a real impact on the algorithm and discard those that haven't.

XGBOOST performance

Max depth

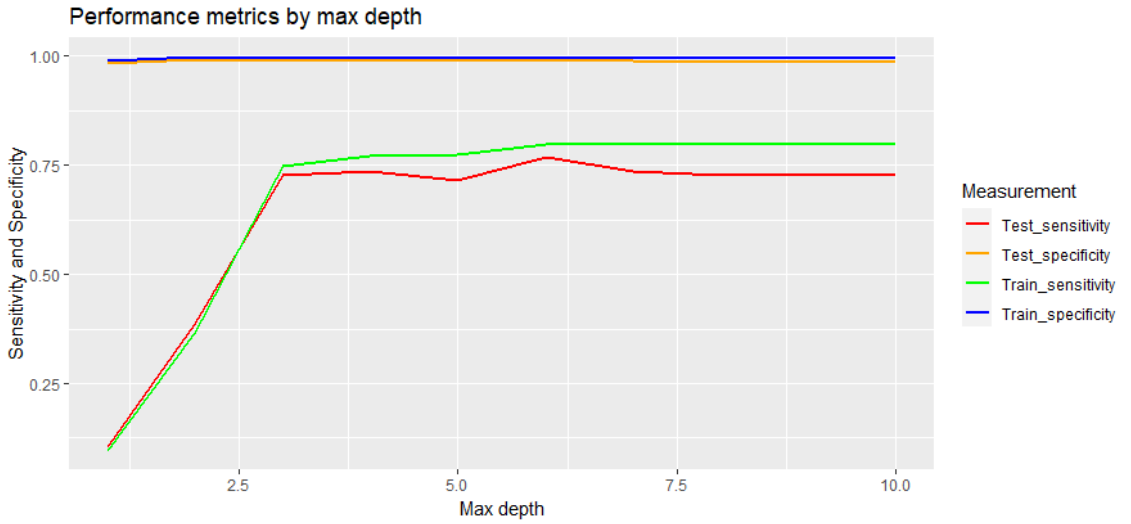

Figure 28: Performance metrics by max depth. Red: Sensitivity test, orange:Specificity test, green: Sensitivity train, blue:Specificity train

On Figure 28 we see that for low values of max depth the performance of specificity it's compromised, with max depth upper than 3 his value is quite better. Here we can observe that random forest is overfitting training data because test data has a lower performance. This is always a factor to take into account. We always need to check that our model isn't very overfitted, otherwise we won't get the guessed results.

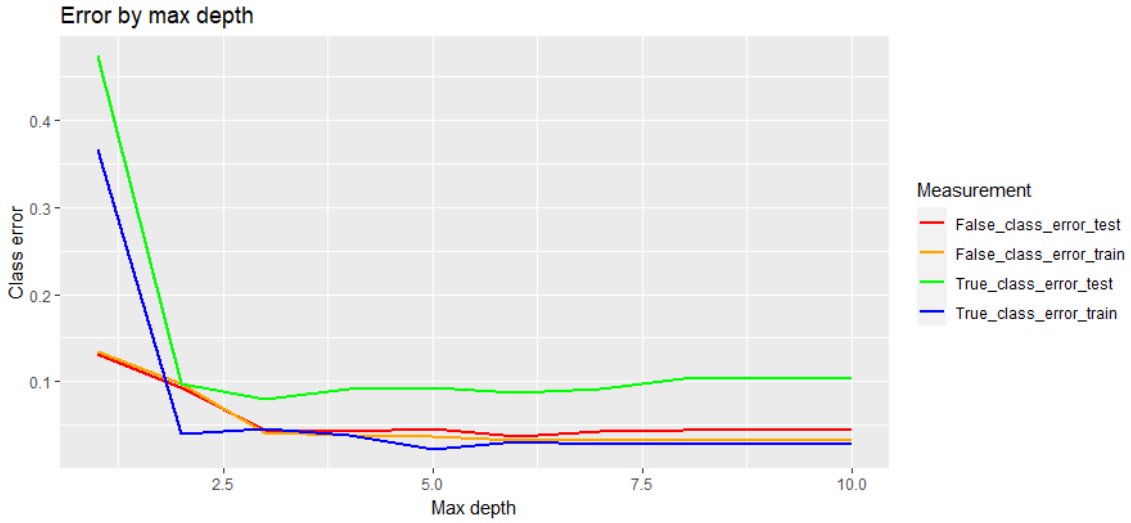

Figure 29: Error by max depth. Red: False class error test, green: True class error test, orange: False class error train, blue: True class error train

The tendency repeats on this plot so we can confirm that a max depth between 3 and 5 it's the best option to make the model. The overfitting can be appreciated here too.

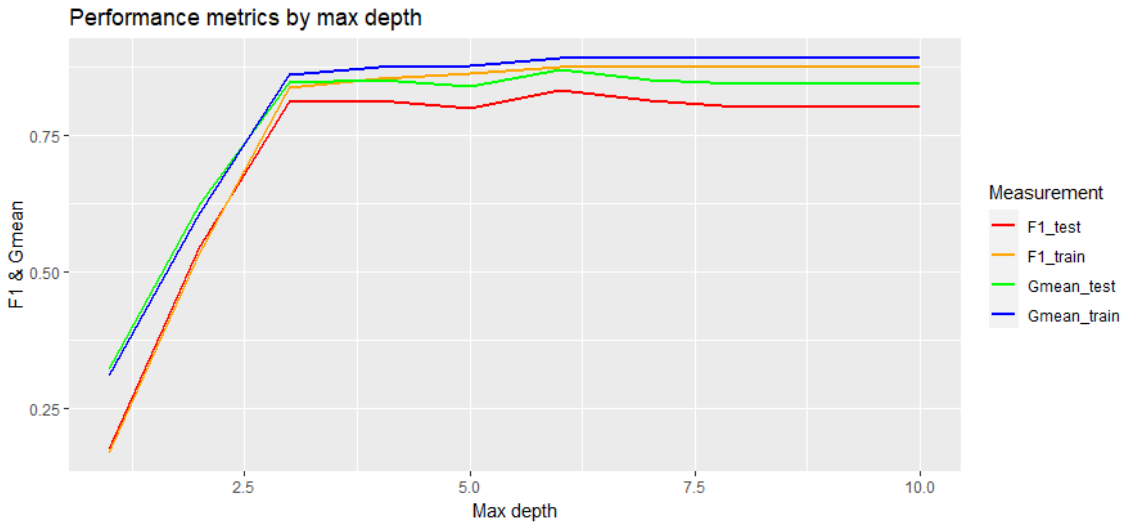

Figure 30: Performance metrics by max depth. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train

Finally, we confirm what previous parameters were showing, the most important to guarantee the good performance is to assign a value to max depth upper than 3.

Gamma

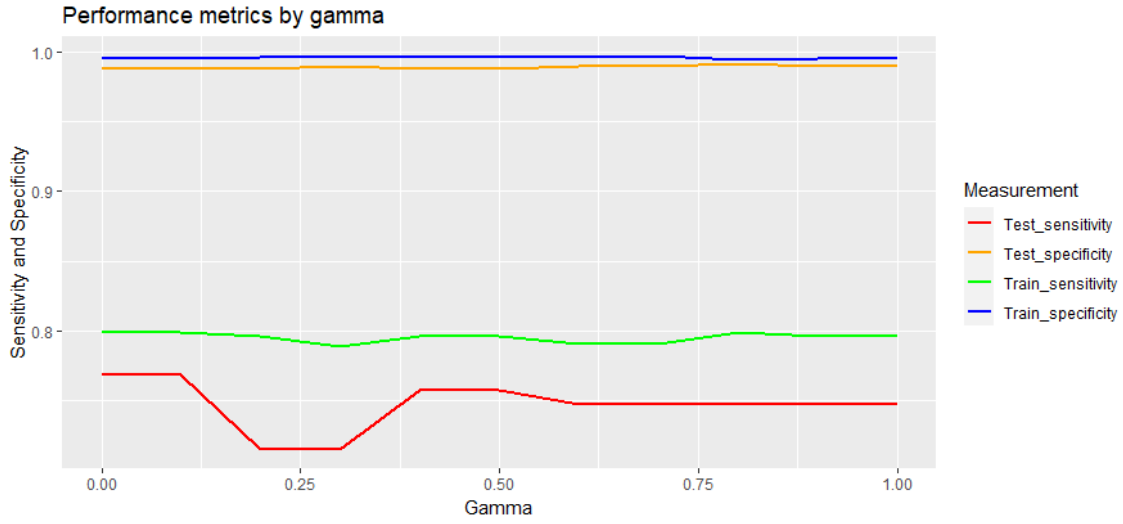

Figure 31: Performance metrics by gamma. Red: Sensitivity test, orange:Specificity test, green: Sensitivity train, blue:Specificity train

As explained before, on the Extreme gradient boosting parameters section gamma is the minimum loss reduction required to make a further partition on a leaf node, meaning that if the loss reduction isn't bigger than the value of gamma the leaf node becomes a final node that doesn't split. This at the same time means that with a higher gamma the model will have shorter branches as those will stop splitting sooner. However, on the plot we observe that on the studied data, changing gamma doesn't have a real impact on the performance of the model. If we had to choose a value the highest performance is with gamma equals to 0.

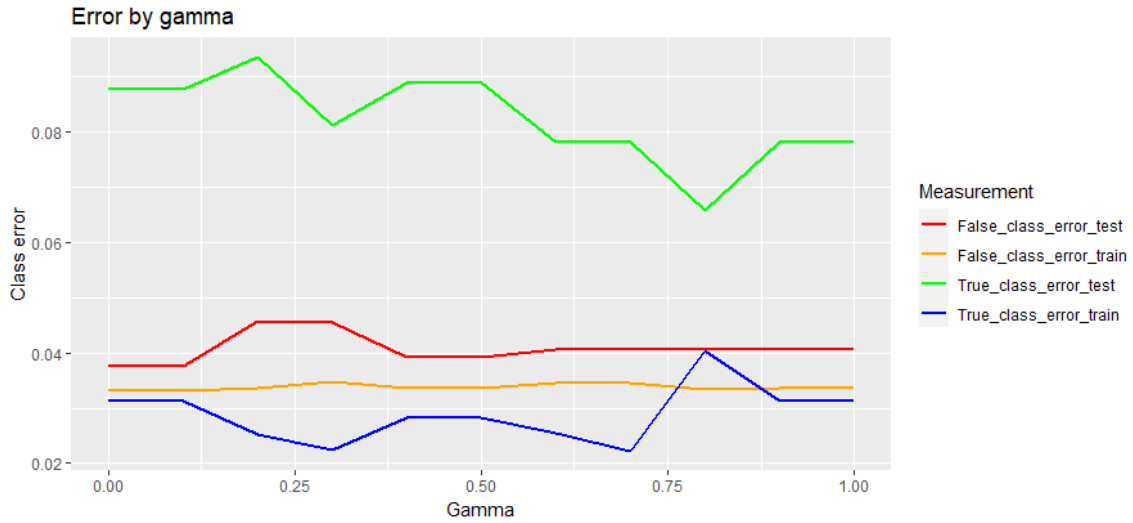

Figure 32: Error by gamma. Red: False class error test, green: True class error test, orange: False class error train, blue: True class error train

In the Figure above we can see how increasing gamma shows a tendency to decrease the true class error of test data, one of the main objectives of the model as its default behaviour tends to predict more false responses than true one's. So based on these parameters the best option is to get a gamma equal to 0.8. This is a conclusion that doesn't fit with the previous decision on Figuren 32. This shows that looking at the wrong parameter when studying a model can be misleading even if you are able to understand what the plot is showing.

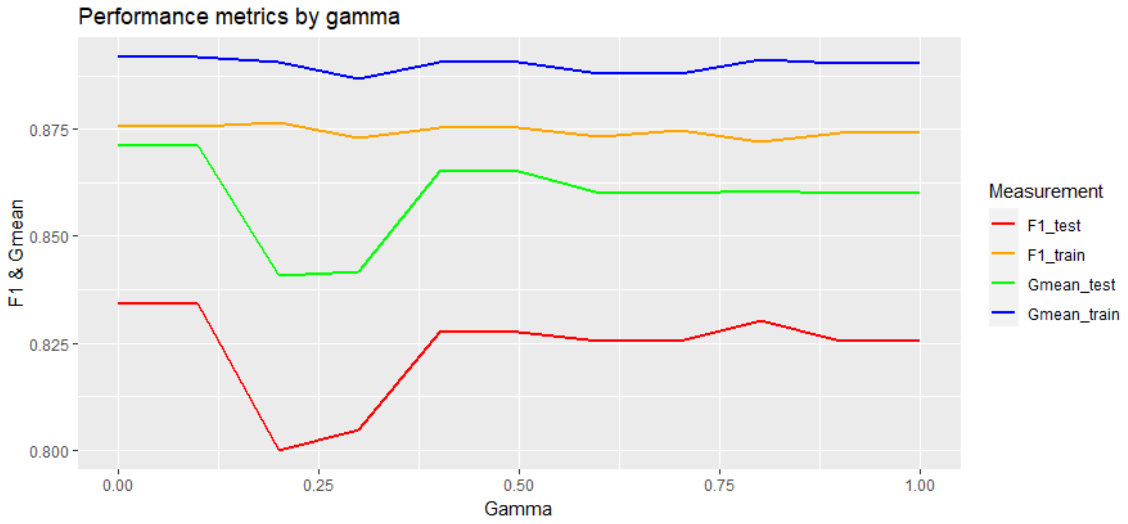

Figure 33: Performance metrics by gamma. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train

On the final plot, again the better performance of true class is hidden again, however the maximum value is on 0.8, with a really small difference between this and other values but thanks to having analysed the other parameters we can confidently choose the 0.8 as the best value in this case.

Eta

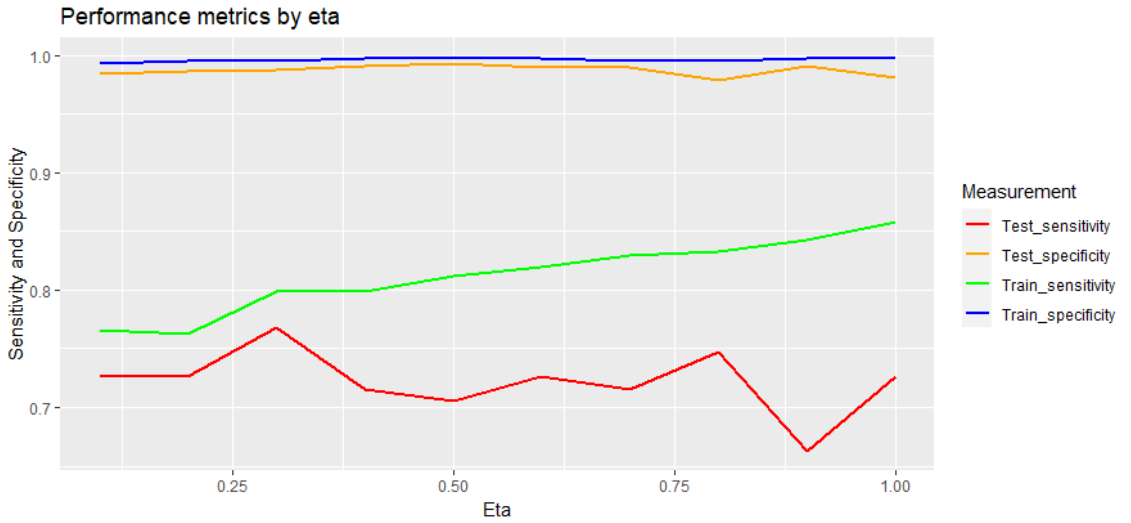

Figure 34: Performance metrics by eta. Red: Sensitivity test, orange:Specificity test, green: Sensitivity train, blue:Specificity train

Eta is a parameter that prevents overfitting. Its default value is 0.3, a point where we can observe the best performance. One thing we notice is that the total range is smaller than most parameters, so the plot has taken values from 0 to 1, this allows us to watch the gap between train and test dataset. In most plots as the range for the parameter is wider, the range of y increases a lot, not letting appreciate this gap on key values. In this case, the variance of the response variables is really low so thi plot is the best way to show its relative performance.

Training data always outperforms test data but the gap between their results gets bigger by increasing the value of eta. The conclusion is that the gap between training and test data draws a parable that has its maximum at 0.3 where the gap is lower than a 4%.

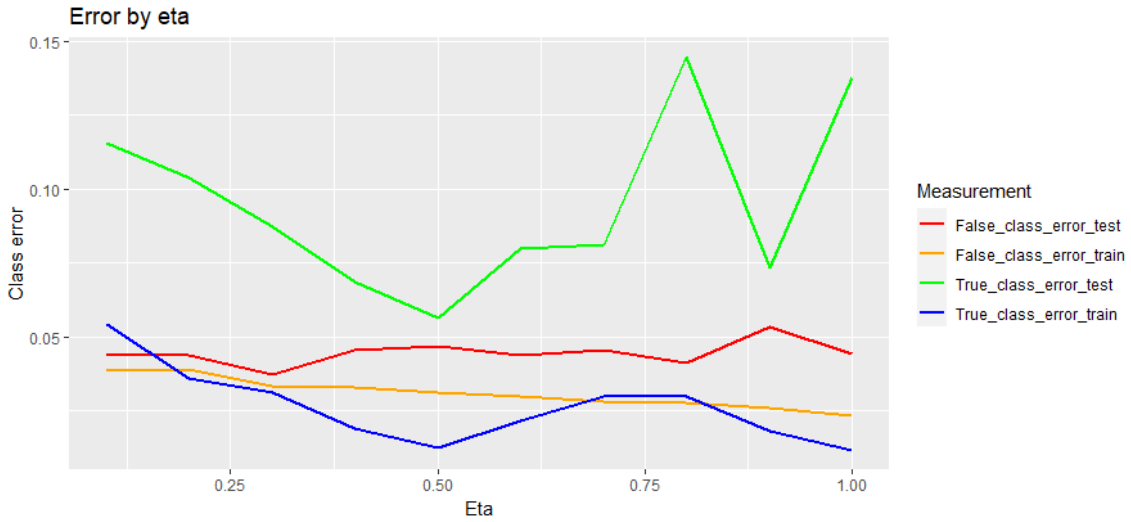

Figure 35: Error by eta. Red: False class error test, green: True class error test, orange: False class error train, blue: True class error train

The parameters on Figure 35 don't show any clear tendency. However we can notice that the gap between test and train data it's quite stable until 0.7 but increases after, this shows an overfitting on training data, something that we need to avoid to make a good model.

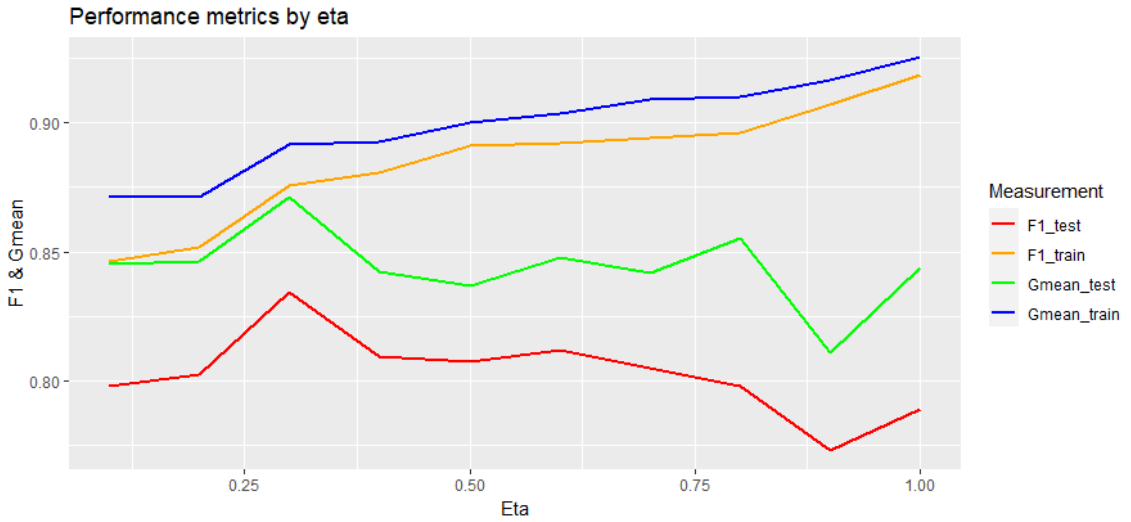

Figure 36: Performance metrics by eta. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train

As on previous plots we observe a maximum on eta equal to 0.3 however the performance is quite similar for all the range. With geometric mean we watch a tendency almost identical to specificity on plot 34, that's due to the high performance of sensitivity.

Min child weight

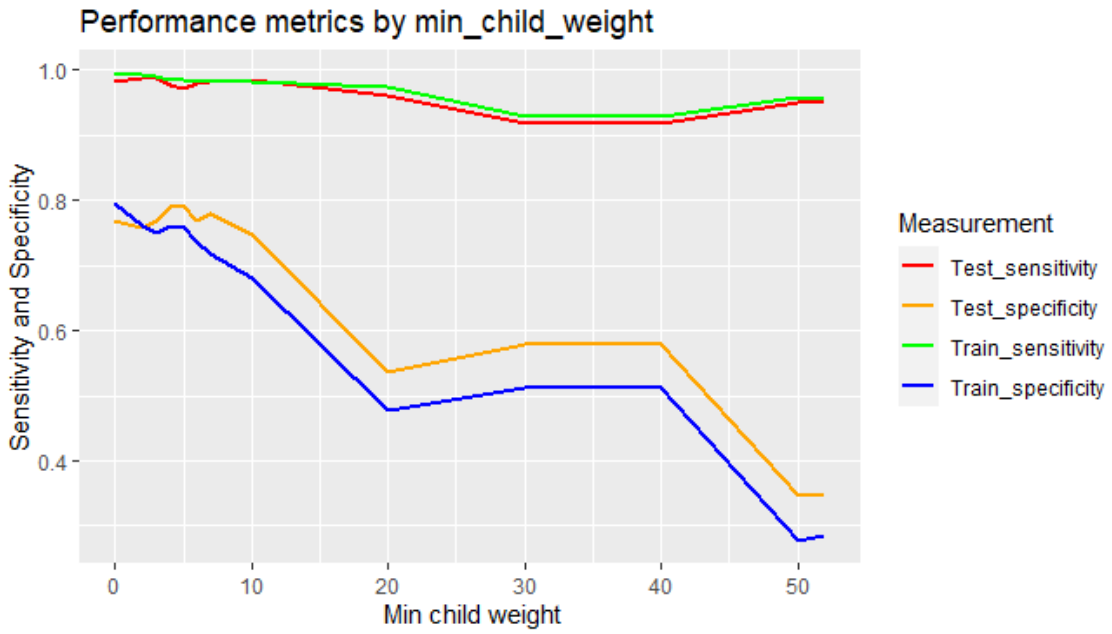

Figure 37: Performance metrics by min_child_weight. Red: Sensitivity test, orange:Specificity test, green: Sensitivity train, blue:Specificity train

Min child weight is one of the few parameters that have an infinite range meaning that is up to the data scientist which range to choose. After some proof it was decided to take the range from 1 to 52.

This range allows you to watch how the increasing of this variable decreases the results of specificity more than most variables for the selected range. Another fact that we can observe is that testing data specificity overperforms training data for values bigger than 3.

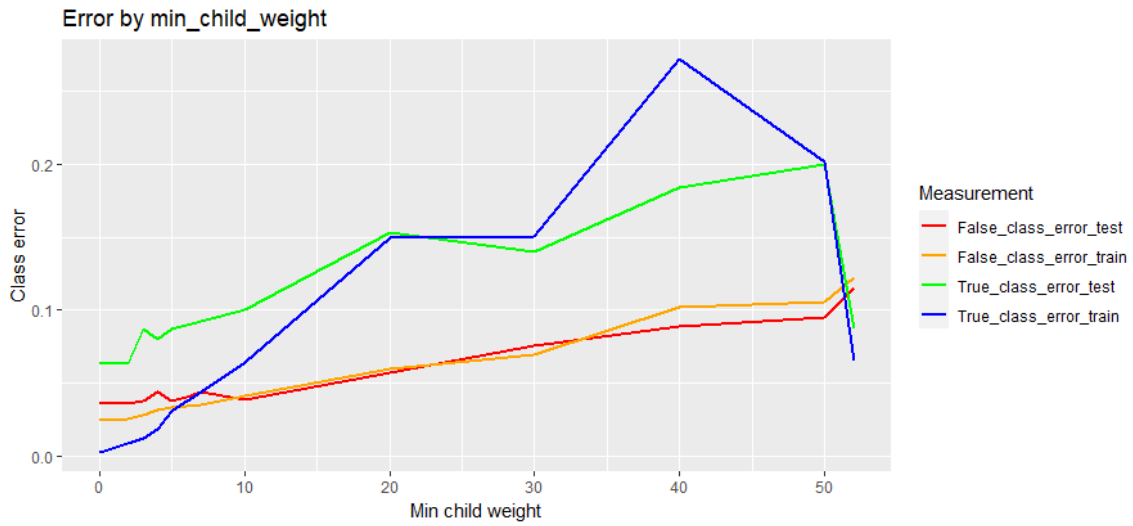

Figure 38: Error by min_child_weight. Red: False class error test, green: True class error test, orange: False class error train, blue: True class error train

Figure 38 in opposition to Figure 37 has an increasing tendency for all variables however we can observe a minimum around on min child weight equals to 2 for the true class error on the testing data indicator and another relative minimum on 10 which allows to see that the performance for the range 0 to 10 isn't linear as it is for upper values.

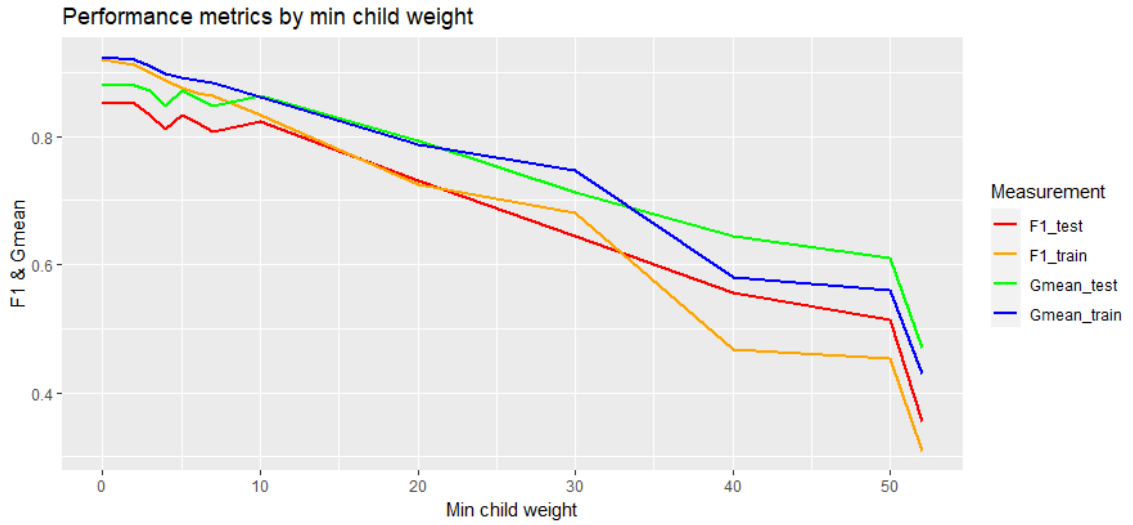

Figure 39: Performance metrics by min child weight. Red: F1 measure test, green: Geometric mean test, orange: F1 measure train, blue: Geometric mean train

Although the geometric mean has a similar tendency as True class error on testing data from Figure 38 for F1 is it clear that the indicator decreases as the variable of the plot is increased so the best value would be min child weight equal to 0.

Code

For this project, 8 R scripts were developed:

- **EDA (Exploratory Data Analysis)**
- **RandomForest_v2**
  - rf_calculations
  - find_best_rf_model
- **XGBoosting_v4**
  - xgb_calculations_v3
  - find_best_model
- **comparing_best_models**

In EDA is placed all the exploratory analysis, it's important to say that the two datasets used in this study are attached as .csv files. They are called churn-bigml-20.csv and churn-bigml-80.csv.

Next it goes RandomForest_v2, here the calculations and the plotting is done thank to rf_calculations and find_best_rf_model where there has been created two function, the first one in rf_calculations give all the indicators shown before and the second one only returns F1 Score test and train with the parameters used to create the model. So first algorithms are used to make the figures and second one to find the best model among many models.

XGBoosting_v4 has the same structure, this document uses the functions placed in xgb_calculations_v3 and find_best_model to do all the necessary calculations.

Finally, the results of find_best_rf_model is been placed on a .csv file due to the time needed to do the calculation, this allows the reader to look for all the data if wanted, the file is called rf_dades.csv and it's attached to this project. On the other hand, the results of find_best_model, the algorithm to find the optimal Extreme Gradient Boosting model, are placed on xgb_dades.csv. Comparing_best_models use the intel from those calculations and then use the best and default models to show the improvement made thanks to the customization.

All the code can be found on the annex for further consultation.It has many comments so any user can easily follow what the code is doing on every step.

The following libraries were used on this project:

```
library(ggplot2)
library(lattice)
library(xgboost)
library(caTools)
library(dplyr)
library(caret)
library(Matrix)
library(xgboost)
library(randomForest)
library(ggplot2)
library(tidyverse)
```

Optimal model search

Although there were some already made algorithms to find the best parameters. Those were using different indicators to decide the best performance so finally it was decided to create a function that given the data and a grid of all the combinations, we decided to try, it would give us the best combination.

For random forest the values of each parameter were used to create the models

Table 4: Selected values for the grid (RF)

Parameter	**Values**
mtry	(1, 2, 3, 4, 5, 6 ,7 , 8, 9, 10)
ntree	(25, 50. 75, 100. 125, 150. 175, 200. 225, 250. 275, 300. 325, 350. 375)
nodesize	(3, 4, 5, 6, 7, 8, 9, 10)

1200 models were created and F1 test and F1 train were calculated for each model.

For extreme gradient boosting the values of each parameter were used to create the models.

Table 5: Selected values for the grid (XGB)

Parameter	**Values**
max depth	(1, 2, 3, 4, 5, 6 ,7 , 8, 9, 10)
eta	(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)
gamma	(0.0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)
min_child_weight	(0.1,3,4,5,6,7,10.20.30)
nrounds	(1, 2, 3, 4, 5, 6, 10)

77000 models were created and F1 test and F1 train were calculated for each model. However 280 NaN values were returned on models with nrounds=3. So those models were ruled out so they weren't useful, these values can appear when calculating F1 Score because it is calculated as following

$$F1\ Score = \frac{2*Precision*Sensitivity}{Precision+Sensitivity}$$ (Eq. 24)

So for cases where Precision and Sensitivity are equal to 0 and indetermination appears.

After obtaining the results the best model was chosen based on F1 score calculated with the test data set. However some insides can be extracted from the grid of models calculated as presented above.

Plotting the results of all the models shows some interesting facts. Starting with XGB, we can point out that most models have a really good performance on both F1 test and train as 75% of models have an F1 test upper than 0.629 and 25% are upper than 0.818. The mean of overfitting is 1,954% while the median is -1,597%. This parameter is calculated as:

$Overfitting = \frac{F1\ test - F1\ train}{F1\ train}$ (Eq.29) So a positive value indicates overfitting, while the mean is positive but the median is negative, this means that more than 50% of models don't have overfitting but those that have overfitting have a higher percentage of overfitting.

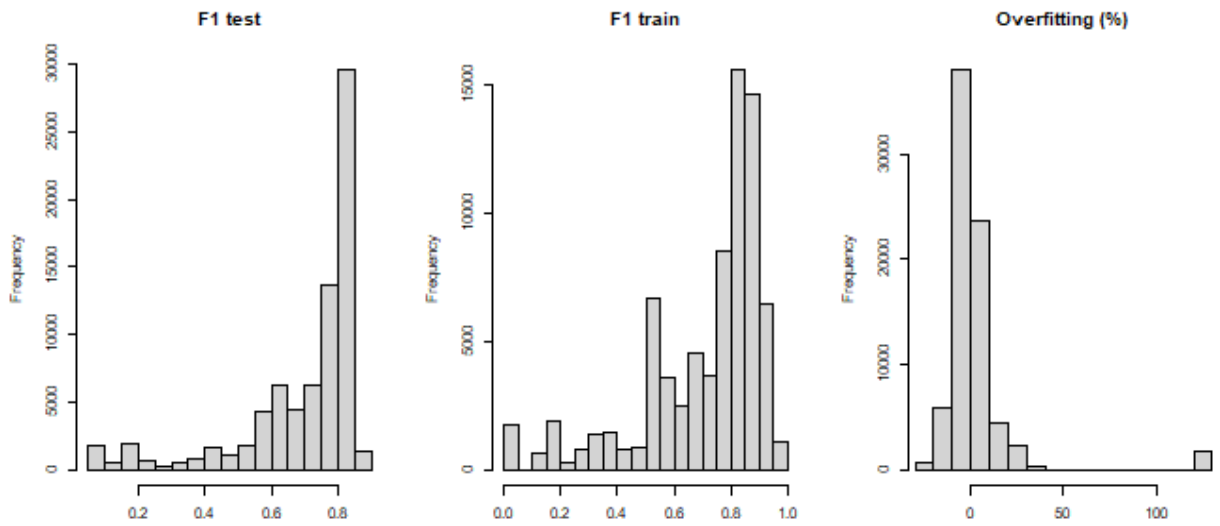

Figure 40: Histograms of F1 test, F1 train and the overfitting for Extreme Gradient Boosting

Table 5: F1 test, F1 train and the overfitting for Extreme Gradient Boosting

xgb	F1 test	F1 train	Overfitting	Overfitting percentage
Min.	0.07692	0.03526	-0.23239	-26,399%
1st Qu	0.62921	0.59724	-0.05399	-6,44%
Median	0.78495	0.79830	-0.01181	-1,597%
Mean	0.69563	0.71019	-0.01455	1,954%
3rd Qu	0.81818	0.85915	0.02180	3,651%
Max.	0.88506	1	0.11544	122,41%

For random forest less parameters were selected for looking for the optimal model so the total number of models is limited. This is in part because of the parameters and in part because of the time needed to do the calculations as the total timing was about 4 hours, compared with XGB, it took over 4 hours too to calculate 77.000 models.

The first thing we notice is that most models are concentrated, 75% of models show an F1 upper of 80% and an overfit between 0% and 5%. Random forest tends to overfit the data a 3% more than XGB. The other interesting fact is that minimum performance of F1 goes dowm to 21% think that, first show the importance of a good selection of the parameters but also show that the default parameters are good because the difference of performance is small (3% of difference) but still we can observed that 50% of the models evaluated have a better performance.

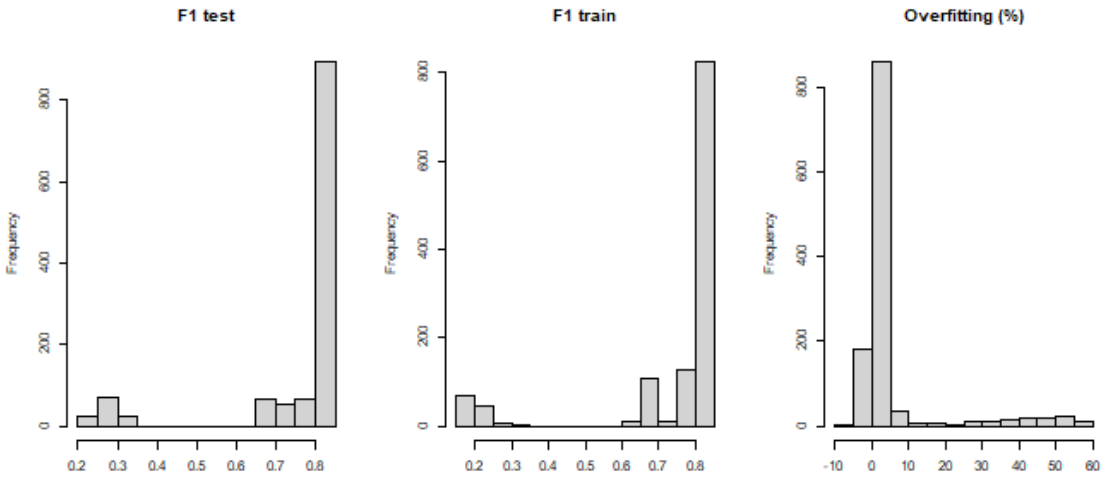

Figure 41: Histograms of F1 test, F1 train and the overfitting for Random Forest

Table 6: Summary of F1 test, F1 train and the overfitting for Random Forest

random forest	F1 test	F1 train	Overfitting	Overfitting percentage
Min.	0.2133	0.1578	-0.0244437	-9.9561%
1st Qu	0.7994	0.7917	0.002186	0.2657%
Median	0.8299	0.8226	0.007205	0.8849%
Mean	0.7592	0.7434	0.015854	5.0029%
3rd Qu	0.8345	0.8291	0.017691	2.2962%
Max.	0.8463	0.8386	0.126776	59.6531%

Comparing xgboost against random forest

In order to understand the performance of those algorithms 4 models have been created, 2 with default parameters of xgboost from the library (xgboost) and random forest from the library (randomForest) and the other 2 with the parameters that have given the best performance of F1 score, an indicator that has been selected because it gives a better interpretation of the result for unbalanced data according to the documentation [27].

Although there were some already made algorithms to find the best parameters. Those were utilizing various metrics to determine the best performance, so it was ultimately decided to develop a function that, given the data and a grid of all the possible combinations, would recommend the best one.

This approach has some limitations as it is impossible to try all possible combinations in a limited amount of time but for the purpose of this thesis it has shown really good results that will be exposed now.

The next table shows the custom parameters of every algorithm with its default value and the customised value after applying the grid search.

Table 7: Default and optimal values of RF and XGB

Random forest			*Extreme gradient boosting*		
	default	*optimal*		*default*	*optimal*
Mtry	*4*	*7*	*Max depth*	*6*	*8*
Ntree	*500*	*175*	*Gamma*	*0*	*0.7*
Nodesize	*1*	*3*	*Eta*	*0.3*	*0.6*
			Min child weight	*1*	*0*
			Nrounds	*2*	*10*

After making the 4 selected models, now their performances will be evaluated in Figure 42 where it can be shown F1 for test and training data of all 4 models.

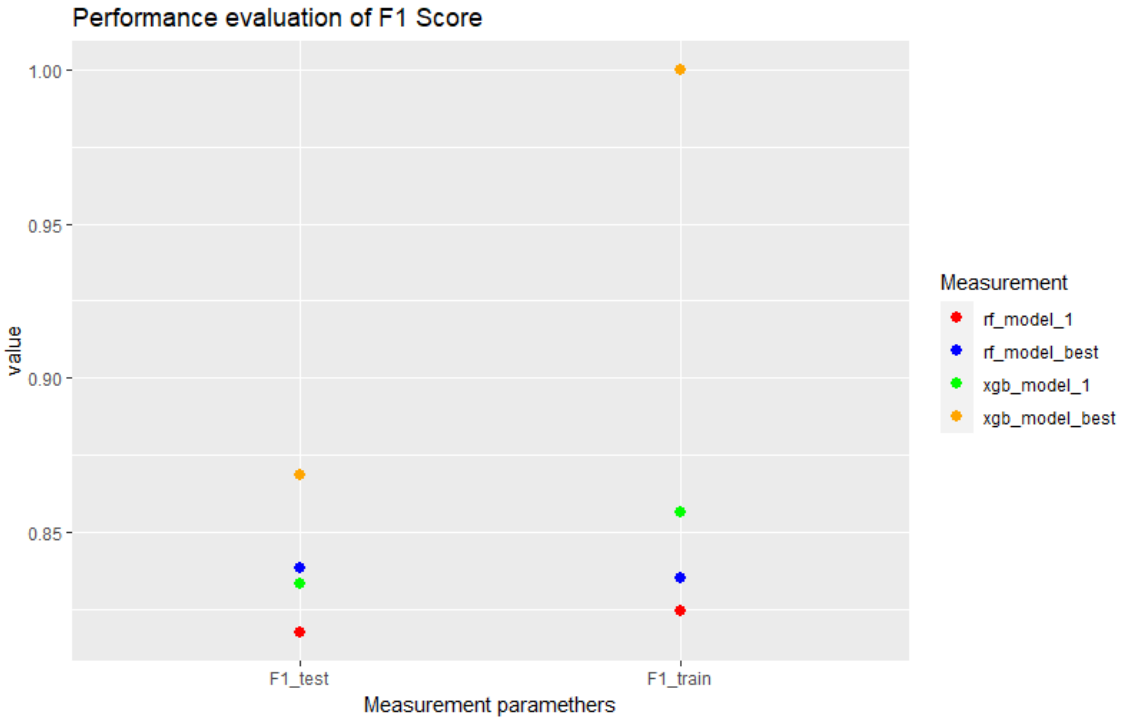

Figure 42: Representation of F1 default random test and train, F1 costum random test and train, F1 default xgboost test and train, F1 custom xgboost test and train.

Table 8: Values of F1 default random test and train, F1 costum random test and train, F1 default xgboost test and train, F1 custom xgboost test and train and Improvements.

	F1 test	*F1 train*
Random Forest Default	*0.8176191*	*0.8248909*
Random Forest Optimal	*0.8463146*	*0.8272176*
Improvement	*+3,509%*	*0.282%*
XGB Default	*0.8333333*	*0.8567416*
XGB Optimal	*0.8850575*	*0.9802891*
Improvement	*+6,207%*	*+14.421%*

As we can see, models perform better after being customised. The custom xgboost model performs somewhat better than random forest, but both models exhibit extremely high performance as measured by the F1 indicator. Extreme gradient boosting overfits the data, especially in training data, is one characteristic we find. In the end, we were able to achieve an F1 Score of 0.8851 for the custom XGB and 0.846 for the custom Random forest. The customization resulted in a performance improvement of 3,51% for Random forest and 6,21% for XGB. Since initial performance was so strong, the gain is rather small. Other important indications are shown and discussed in the following Figure 43.

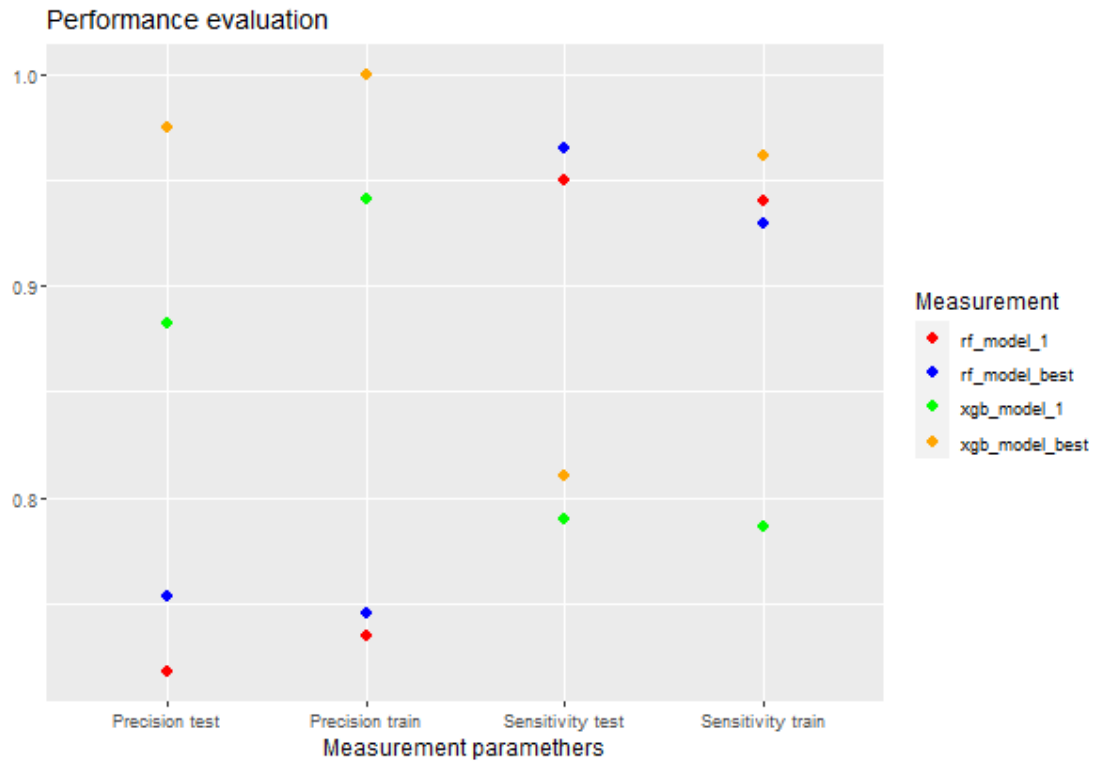

Figure 43: Representation of Precision and Sensitivity for training and testing data on the 4 models.

Looking at Figure 43, the first thing we notice is that almost all parameters perform better with the optimal model. However there is an exception on the Sensitivity train of Random Forest, this may seem an anomaly but actually the difference between the values is almost negligible. Sensitivity performs better on Random Forest while Precision does it on Extreme Gradient Boosting this means that Random forest predicts less False Negatives while XGB predicts less False Positives, in the case of study it is preferable to have more False Positives than False Negatives. False Negatives indicate that the corporation won't take action on the clients who the algorithms predict will leave the company (True Positives and False Positives), even though they might still do so. In every instance, the increase in parameters from the standard model to the customized model is greater for XGB, nearly doubling Random forest. While Random forest has an improvement for Precision test of 4,99% and for Sensitivity test of 1,59%, XGB increases Precision test by 10,46% and Sensitivity test by 2,67%.

Table 9: Summary of Precision test, Precision train, Sensitivity test and Sensitivity train for RF and XGB.

	Precision test	***Precision train***	***Sensitivity test***	***Sensitivity train***
Random Forest Default	*0.7178947*	*0.7345361*	*0.9499623*	*0.9406408*
Random Forest Optimal	*0.7536842*	*0.7453608*	*0.9650383*	*0.9293109*
Improvement	*4.99%*	*1.47%*	*1.59%*	*-1.20%*

XGB Default	*0.8823529*	*0.9413580*	*0.7894737*	*0.7860825*
XGB Optimal	*0.9746835*	*1*	*0.8105263*	*0.9613402*
Improvement	*10.46%*	*6.23%*	*2.67%*	*22.30%*

After comparing the indicators, one of the advantages from random forest is its explicability with the parameter "importance", this one can be plotted. So above the importance of the two random forest models are presented. Using the variable importance plot in R Studio, the best spectral variables that were important in the decision tree model are shown in Figures 44 and 45. A large value of MeanDecreaseAccuracy or MeanDecreaseGini indicates that the explanatory variables are an important predictor for Churn. Depending on which criteria we choose, some variables are more important or less. We will first discuss MeanDecreaseAccuracy before moving on to MeanDecreaseGini. Customer.service.calls performed the best for MeanDecreaseAccuracy in both the default and optimal models' variable importance tables, while Area.code, Total.night.calls, State Account.length, Total.day.calls, and Total.eve.calls carried no weight in either model, though their relative importance did not change. In the variable importance table for the default model and the ideal for MeanDecreaseGini, Total.day.minutes performed best, whilst Area.code is the less important variable. In order to speed up processing, it was decided to create models for Random Forest without this variable for the functions that provided the calculations.

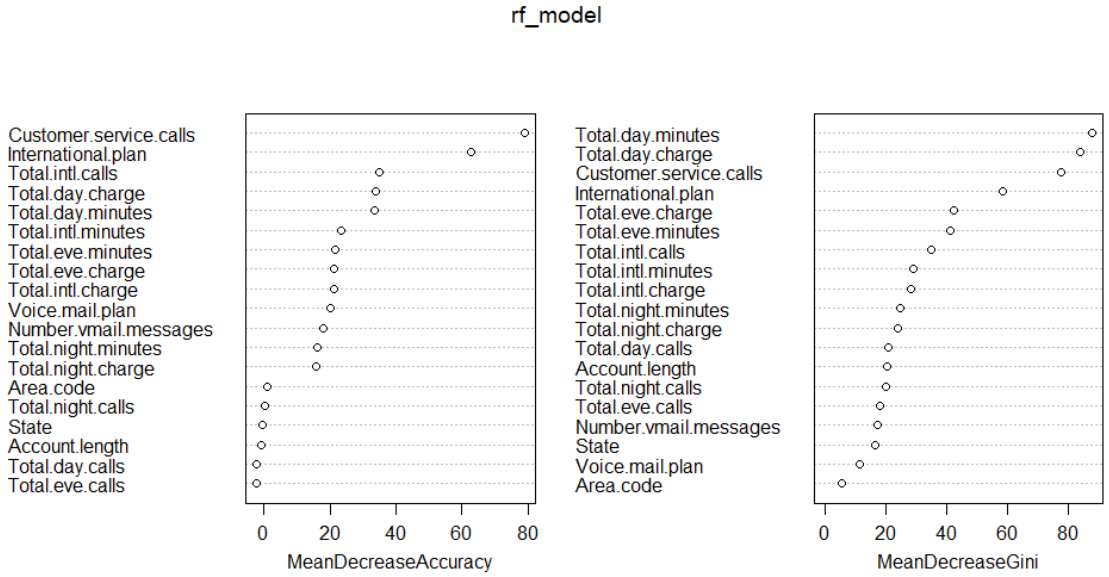

Figure 44: Importance of Random forest default model

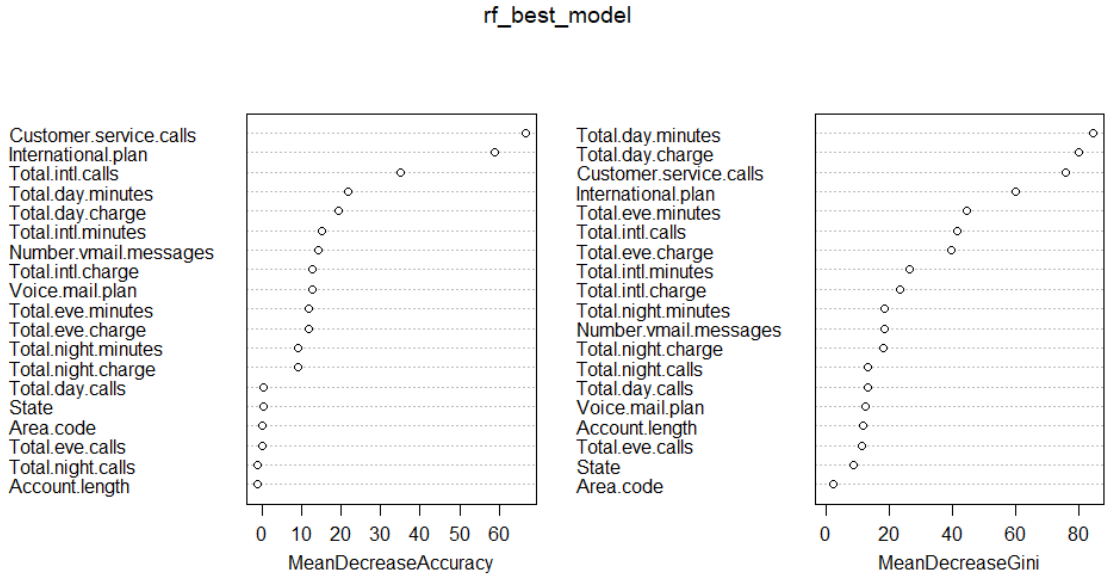

Figure 45: Importance of Random forest customised model

This figure is crucial when the data analyst and the end user are trying to interpret the data; in most circumstances, knowing which variables influence the replies can aid in reaching a decision. Since the Today.day.minutes and Customer.service.calls variables, for both the default and custom models, have the greatest influence on the response, looking at these fields will reveal crucial information. It is actually not surprising that these calls have an impact on the client's decision to switch because the majority of customers contact customer service when they are unsatisfied. Given that the data for both models is identical, it is not a surprise that the majority of them are ranked in the same order of importance.

The user of xgboost, on the other hand, is only interested in the outcomes, which are typically predictions, because it is more like a dark box where it is difficult to trace the variables. This is useful in a variety of circumstances and enables you to take action before the prediction is realized. It is important to note that one benefit of xgboost in this study, where we are comparing random forest performance against that of xgboost, is its processing speed. Since the study doesn't involve a real-time process, the speed difference between xbgoost and random forest isn't significant. However, in other situations, this might be a solid reason to choose xbgoost over random forest.

Conclusion

Two machine learning techniques were introduced and examined in this research with the goal of understanding the impact that personalization has on the effectiveness of the models. Since having unbalanced data has been noted as a regular problem with this kind of project, choosing the appropriate performance indicators is essential to choosing the optimum model.

The creation of four machine learning models was engaged in this project: random forest, extreme gradient boosting, random forest with customized parameters, and extreme gradient boosting with customized parameters. The main objective, which will be illustrated in the lines that follow, is to look at the performance boost brought on by customization. They were employed to predict the variable Churn, which denotes whether a customer has left the business or not.

The training data set, which comprises 80% of the total data set, is used to train the models. Then, they are tested using the test data set, which only contains 20% of the data because no records were removed during the data cleaning procedure. To allow its application on the employed machine learning approaches, some codification has been done to a select few regions.

We discover that most models perform better with training data, which is consistent with the assumption that models frequently overfit the data. Random forest overestimates on average by 5%, whereas XGB overestimates by only 2%, according to research. We have always made decisions based on the test set results because we were aware of this fact and the algorithm did not use test data to generate the model.

A proprietary algorithm has been built to generate and assess on the F1 Score a grid of the user-selectable range of values for the various parameters provided by RF and XGB. The best model out of a total of 1200 models for RF and 77,000 models for XGB was chosen based on F1 measure performance.

Increasing the parameters mtry and ntree results in greater performance, according to some RF insights. However, for ntree, performance is best at 175; so, increasing the value doesn't improve results and increase calculation time. For XGB, we discovered that maximum depth performed well for values higher than 3, while minimum child weight performed best at 0.

For RF and XGB, the improvement as a result of customization is 3,51% and 6,21%, respectively. Perhaps it does appear to be a significant improvement, but in a multibillion dollar industry, this distinction can have benefits worth millions of dollars. XGB has an F1 test score of 0.8850575, whereas the top RF model has a score of 0.8463146. XGB outperforms XGB by 4,5 %. But when it comes to False Negatives, RF performs better than XGB, which does better with False Positives. If the objective is to retain as many clients as possible is preferable to predict False Positives in this study than False Negatives (clients who churn but the algorithm predicts the opposite). Because False Negatives would be clients that churn without any action to prevent it.

The next step is to sectorize the clients using the chosen model and put a plan of action in place to keep them from defecting. Different ways of client retention may be used if some other ML algorithms, like clustering, were used to categorize such clients according to their attributes.

Bibliography

[1] HEAVY.IA, “Customer Churn Analysis in Telco” [online]. Available: https://www.heavy.ai/use-case/customer-churn-analysis

[2] Panorama, “ The Role of Data Analytics in the Telecom Industry” [online]. Available: https://panorama.com/blog/the-role-of-data-analytics-in-the-telecom-industry/

[3] Tutort Academy, “Role of Data Science in Telecom Industry” , Jan. 2013 [online]. Available: https://medium.com/@-TutortAcademy/role-of-data-science-in-telecom-industry-7b5dd74adf1

[4] V. Lazarov and M. Capota, ‘‘Churn prediction,’’ Bus. Anal. Course, TUM Comput. Sci, Technische Univ. München, Tech. Rep., 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download? doi=10.1.1.462.7201&rep=rep1&type=pdf

[5] R. Vadakattu, B. Panda, S. Narayan, and H. Godhia, ‘‘Enterprise subscription churn prediction,’’ in Proc. IEEE Int. Conf. Big Data, Nov. 2015, pp. 1317–1321.

[6] M. Hassouna, A. Tarhini, T. Elyas, and M. S. AbouTrab. (Jan. 2016). ‘‘Customer churn in mobile markets a comparison of techniques.’’ [Online]. Available: https://arxiv.org/abs/1607.07792

[7] A. Sharma and P. K. Kumar. (Sep. 2013). ‘‘A neural network based approach for predicting customer churn in cellular network services.’’ [Online]. Available: https://arxiv.org/abs/1309.3945

[8] A. Amin, F. Al-Obeidat, B. Shah, A. Adnan, J. Loo, and S. Anwar, ‘‘Customer churn prediction in telecommunication industry using data certainty,’’ J. Bus. Res., vol. 94, pp. 290–301, Jan. 2019.

[9] J. Vijaya and E. Sivasankar, ‘‘An efficient system for customer churn prediction through particle swarm optimization based feature selection model with simulated annealing,’’ Cluster Comput., pp. 1–12, Sep. 2017. doi: 10.1007/s10586-017-1172-1.

[10] V. Umayaparvathi and K. Iyakutti, ‘‘Applications of data mining techniques in telecom churn prediction,’’ Int. J. Comput. Appl., vol. 42, no. 20, pp. 5–9, Mar. 2012.

[11] A. T. Jahromi, M. Moeini, I. Akbari, and A. Akbarzadeh, ‘‘A dual-step multi-algorithm approach for churn prediction in pre-paid telecommunications service providers,’’ J. Innov. Sustainab., vol. 1, no. 2, pp. 2179–3565, 2010.

[12] V. Yeshwanth, V. V. Raj, and M. Saravanan, ‘‘Evolutionary churn prediction in mobile networks using hybrid learning,’’ in Proc. 25th Int. FLAIRS Conf., Mar. 2011, pp. 471–476.

[13] G. Nie, W. Rowe, L. Zhang, Y. Tian, and Y. Shi, ‘‘Credit card churn forecasting by logistic regression and decision tree,’’ Expert Syst. Appl., vol. 38, no. 12, pp. 15273–15285, Nov./Dec. 2011.

[14] P. T. Kotler, Marketing Management: Analysis, Planning, Implementation and Control. London, U.K.: Prentice-Hall, 1994.

[15] S. A. Qureshi, A. S. Rehman, A. M. Qamar, A. Kamal, and A. Rehman, ‘‘Telecommunication subscribers’ churn prediction model using machine learning,’’ in Proc. 8th Int. Conf. Digit. Inf. Manage., Sep. 2013, pp. 131–136.

[16] S. V. Nath and R. S. Behara, ‘‘Customer churn analysis in the wireless industry: A data mining approach,’’ in Proc. Annu. Meeting Decis. Sci. Inst., vol. 561, Nov. 2003, pp. 505–510.

[17] Y. Zhang, J. Qi, H. Shu, and J. Cao, ‘‘A hybrid KNN-LR classifier and its application in customer churn prediction,’’ in Proc. IEEE Int. Conf. Syst., Man Cybern., Oct. 2007, pp. 3265–3269.

[18] Y. Huang and T. Kechadi, ‘‘An effective hybrid learning system for telecommunication churn prediction,’’ Expert Syst. Appl., vol. 40, no. 14, pp. 5635–5647, Oct. 2013.

[19] F. F. Reichheld and W. E. Sasser, Jr., ‘‘Zero defections: Quality comes to services,’’ Harvard Bus. Rev., vol. 68, no. 5, pp. 105–111, 1990.

[20] A. Amin et al., ‘‘Cross-company customer churn prediction in telecommunication: A comparison of data transformation methods,’’ Int. J. Inf. Manage., vol. 46, pp. 304–319, Jun. 2019.

[21] R. Rajamohamed and J. Manokaran, ‘‘Improved credit card churn prediction based on rough clustering and supervised learning techniques,’’ Cluster Comput., vol. 21, no. 1, pp. 65–77, Mar. 2018.

[22] Y. Huang, B. Huang, and M.-T. Kechadi, ‘‘A rule-based method for customer churn prediction in telecommunication services,’’ in Proc. Pacific– Asia Conf. Knowl. Discovery Data Mining. Berlin, Germany: Springer, 2011, pp. 411–422.

[23] A. Idris and A. Khan, ‘‘Customer churn prediction for telecommunication: Employing various various features selection techniques and tree based ensemble classifiers,’’ in Proc. 15th Int. Multitopic Conf., Dec. 2012, pp. 23–27.

[24] J. Hadden, A. Tiwari, R. Roy, and D. Ruta, ‘‘Computer assisted customer churn management: State-of-the-art and future trends,’’ Comput. Oper. Res., vol. 34, no. 10, pp. 2902–2917, Oct. 2007.

[25] Kaapalx, “ Decision Tree in R Programming” , Dec. 2021. [online]. Available: https://www.geeksforgeeks.org/decision-tree-in-r-programming/

[26] Theo Saarinen, “Package ‘Rforestry’” Jul. 2022. [online]. Available: https://cran.r-project.org/web/packages/Rforestry/Rforestry.pdf

[27] Joos Korstanje, “The F1 score“ Aug. 2021. [online] Available: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

ANNEX: CODE

EDA (Exploratory Data Analysis) **64**

RandomForest_v2 **66**

rf_calculations 73

find_best_rf_model 76

XGBoosting_v4 **77**

xgb_calculations_v3 87

find_best_model 91

comparing_best_models **93**

EDA (Exploratory Data Analysis)

```
#EXPLORATORY DATA ANALYSYS



#LOADING DATA
setwd("C:/Users/pc/Desktop/tfm")
library(tidyverse)
TRAIN<-read.csv("churn-bigml-80.csv")
TEST<-read.csv("churn-bigml-20.csv")
TOTAL<-rbind(TRAIN,TEST)

#DATA CLEANING FOR RANDOM FOREST
TEST$State<- factor(TEST$State)
TEST$International.plan<- factor(TEST$International.plan)
TEST$Voice.mail.plan<- factor(TEST$Voice.mail.plan)
TEST$Churn<- factor(TEST$Churn)
TEST$Area.code<- factor(TEST$Area.code)
TRAIN$State<- factor(TRAIN$State)
TRAIN$International.plan<- factor(TRAIN$International.plan)
TRAIN$Voice.mail.plan<- factor(TRAIN$Voice.mail.plan)
TRAIN$Churn<- factor(TRAIN$Churn)
TRAIN$Area.code<- factor(TRAIN$Area.code)
TOTAL$State<- factor(TOTAL$State)
TOTAL$International.plan<- factor(TOTAL$International.plan)
TOTAL$Voice.mail.plan<- factor(TOTAL$Voice.mail.plan)
TOTAL$Churn<- factor(TOTAL$Churn)
TOTAL$Area.code<- factor(TOTAL$Area.code)

str(TOTAL)
summary(TOTAL)


#Numeric variables
numerical<- TOTAL %>% select(-c(Churn,State,
                                International.plan, Voice.mail.plan ,
Area.code))
summary(numerical)

#Categorical variables
categor<- TOTAL[, c(3,4,5,20)]
summary(categor)
table(TOTAL$State)
names(TOTAL$State)


#Histograms
layout(matrix(c(1:6), nrow=2, byrow=TRUE))
```

```
hist(TOTAL$Total.day.minutes,main="Total day minutes")
hist(TOTAL$Total.day.calls, main="Total day calls")
hist(TOTAL$Total.day.charge, main="Total day charge")

hist(TOTAL$Total.eve.minutes,main="Total eve minutes")
hist(TOTAL$Total.eve.calls, main="Total eve calls")
hist(TOTAL$Total.eve.charge, main="Total eve charge")

layout(matrix(c(1:6), nrow=2, byrow=TRUE))

hist(TOTAL$Total.night.minutes,main="Total night minutes")
hist(TOTAL$Total.night.calls, main="Total night calls")
hist(TOTAL$Total.night.charge, main="Total night charge")

hist(TOTAL$Total.intl.minutes,main="Total intl minutes")
hist(TOTAL$Total.intl.calls, main="Total intl calls")
hist(TOTAL$Total.intl.charge, main="Total intl charge")

layout(matrix(c(1:1)))
hist(TOTAL$Customer.service.calls, xlim=c(-0.5, 7.5))

#distribution of State
layout(matrix(c(1:1)))
counts <- table(TOTAL$State)
barplot(counts, main="Distribution of State", ylab="Number of clients")

#distribution of churn
counts <- round(prop.table(table(TOTAL$Churn)),2)
barp<-barplot(counts, col=c("orange", "blue"),
              main="Distribución de Churn", ylab="Porcentaje de
clientes",
              legend.text=c("False", "True"),
              ylim=c(0,1), xlim=c(0,3.3),font.axis=1)
text(barp, counts + 0.1, labels = counts)


ggplot(TOTAL, aes(x=International.plan, y=Total.eve.calls)) +
  geom_bar(stat = "identity")
plot(TOTAL[,1:3])
plot(TOTAL[,c("Total.day.minutes","Total.day.calls","Total.day.charge")]
)

#Correlation matrix
EXTRACION<-c(1, 3, 4, 5, 20)
Total2<- TOTAL[,-EXTRACION]
Total2<- data.matrix(Total2, rownames.force = NA)
res <- cor(Total2)
round(res, 2)
library(corrplot)
corrplot(res, type = "upper", order = "hclust",
         tl.col = "black", tl.srt = 45)
```

RandomForest_v2

```
####RANDOM FOREST####


####LOADING DATA####
setwd("C:/Users/pc/Desktop/tfm")
library(tidyverse)
library(randomForest)
library(ggplot2)
TRAIN<-read.csv("churn-bigml-80.csv")
TEST<-read.csv("churn-bigml-20.csv")
TOTAL<-rbind(TRAIN,TEST)

####DATA CLEANING FOR RANDOM FOREST####
TEST$State<- factor(TEST$State)
TEST$International.plan<- factor(TEST$International.plan)
TEST$Voice.mail.plan<- factor(TEST$Voice.mail.plan)
TEST$Churn<- factor(TEST$Churn)
TEST$Area.code<- factor(TEST$Area.code)
TRAIN$State<- factor(TRAIN$State)
TRAIN$International.plan<- factor(TRAIN$International.plan)
TRAIN$Voice.mail.plan<- factor(TRAIN$Voice.mail.plan)
TRAIN$Churn<- factor(TRAIN$Churn)
TRAIN$Area.code<- factor(TRAIN$Area.code)
TOTAL$State<- factor(TOTAL$State)
TOTAL$International.plan<- factor(TOTAL$International.plan)
TOTAL$Voice.mail.plan<- factor(TOTAL$Voice.mail.plan)
TOTAL$Churn<- factor(TOTAL$Churn)
TOTAL$Area.code<- factor(TOTAL$Area.code)


####DEFAULT MODEL####
baggedTrees = randomForest(as.factor(Churn) ~ .,
                           data = TRAIN,
                           ntree = 500,
                           mtry = as.integer(sqrt((dim(TRAIN)[2]-1))),
                           keep.forest = TRUE,
                           xtest = TEST[,-20],
                           ytest = as.factor(TEST$Churn),
                           importance=TRUE)

baggedTrees
####Showing importance####
baggedTrees[10]
####Sensitivity TEST####
Sensitest
<-baggedTrees[[17]][[3]][[4]]/(baggedTrees[[17]][[3]][[3]]+baggedTrees[
[17]][[3]][[4]])
```

```
####Specifivity TEST####
Spectest<-
baggedTrees[[17]][[3]][[1]]/(baggedTrees[[17]][[3]][[2]]+baggedTrees[[1
7]][[3]][[1]])
####error false class TEST####
baggedTrees[[17]][[3]][[5]]

####error true class TEST####
baggedTrees[[17]][[3]][[6]]

####Confusion matrix TEST####
baggedTrees[[17]][[3]]

####Sensitivity TRAIN####
Sensitrain <-
baggedTrees[[5]][[4]]/(baggedTrees[[5]][[3]]+baggedTrees[[5]][[4]])
####Specifivity TRAIN####
Spectestrain<-
baggedTrees[[5]][[1]]/(baggedTrees[[5]][[2]]+baggedTrees[[5]][[1]])

####error false class TRAIN####
baggedTrees[[5]][[5]]]

####error true class TRAIN####
baggedTrees[[5]][[6]]

####Confusion matrix TRAIN####
baggedTrees[[5]]

####Importance####
importance(baggedTrees)
varImpPlot(baggedTrees)



#BEST MODEL####
rfGrid <-  expand.grid(mtry = (1:10),
                       ntree = (1:15)*25,
                       nodesize = seq(3,10, by=1))

t_ini<-Sys.time()
source('find_best_rf_model.R')
results<- find_best_rf_model(TRAIN,TEST,rfGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t

"""
mtry ntree nodesize   F1_test  F1_train
  7   175        3 0.8463146 0.8272176
"""
#Time difference of 3.808516 hours
```

```
#Grid analysis####
#to save the results of the grid search in csv
#write.csv2(results[[1]], 'rf_dades_10092022.csv')
#to load these results on R Studio
results_grid_search<-read.csv2('rf_dades_10092022.csv')
#for removing Na results in cases they exist
results_grid_search<-results_grid_search[!is.na(results_grid_search$F1_t
est),]
results_grid_search$mtry<-factor(results_grid_search$mtry)
results_grid_search$ntree<-factor(results_grid_search$ntree)
results_grid_search$nodesize<-factor(results_grid_search$nodesize)
rf_overfit<-results_grid_search$F1_test-results_grid_search$F1_train
rf_overfit_percent<-(results_grid_search$F1_test-results_grid_search$F1_
train)*100/results_grid_search$F1_train
summary(results_grid_search)
summary(rf_overfit)
summary(rf_overfit_percent)

#Plot
layout(matrix(c(1:3), nrow=1, byrow=TRUE))
hist(results_grid_search$F1_test,main="F1 test", xlab='')
hist(results_grid_search$F1_train, main="F1 train",xlab='')
hist(rf_overfit_percent, main="Overfitting (%)",xlab='')


#PLOTS####

####MTRY ####
rfGrid <-  expand.grid(mtry = (1:8),
                       ntree = 500,
                       nodesize = 1,
                       max_depth  = 99)

t_ini<-Sys.time()
source('rf_calculations.R')
results<- rf_calculation(TRAIN,TEST,rfGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t

#3.6 mins


x<-seq(1,8, by=1)
paste("Optimal value for mtry:", x[which.max(results$F1_test)]) ####7
paste("Optimal value for mtry:", x[which.max(results$Gmean_test)]) ####4

#PLOT 1####
Test_sensitivity<-results$rfsenstest
Test_specificity<-results$rfspetest
Train_sensitivity<-results$rfsenstrain
Train_specificity<-results$rfspetrain
x<-seq(1,8, by=1)
```

```
df <- data.frame(x,Test_sensitivity,
                 Train_sensitivity ,Test_specificity,
                 Train_specificity) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("purple", "red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by mtry', x='Mtry',
       y='Sensitivity and Specificity')


#PLOT 2####
False_class_error_test<-results$rfeftest
False_class_error_train<-results$rfeftrain
True_class_error_test<-results$rfettest
True_class_error_train<-results$rfettrain
x<-seq(1,8, by=1)

df <- data.frame(x,False_class_error_test,
                 False_class_error_train ,True_class_error_test,
                 True_class_error_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("purple","red","orange", "green","blue"))+
  labs(title='Error by mtry', x='Mtry', y='Class error')

#PLOT 3####
F1_test<- results$F1_test
F1_train<- results$F1_train
Gmean_test<- results$Gmean_test
Gmean_train<- results$Gmean_train
x<-seq(1,8, by=1)

df1 <- data.frame(x,F1_test, F1_train, Gmean_test, Gmean_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df1, aes(x=x, y=value, group=Measurement,
                colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by mtry', x='Mtry',
       y='F1 & Gmean')

####NTREE####
rfGrid <-  expand.grid(mtry = 6,
```

```
                        ntree = seq(5,500, by=10),
                        nodesize = 1,
                        max_depth  = 99)

t_ini<-Sys.time()
source('rf_calculations.R')
results<- rf_calculation(TRAIN,TEST,rfGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t
#11,55 mins


x<-seq(5,500, by=10)
paste("Optimal value for number of trees:", x[which.max(F1_test)])
####65
paste("Optimal value for number of trees:", x[which.max(Gmean_test)])
####35

#PLOT 1####
Test_sensitivity<-results$rfsenstest
Test_specificity<-results$rfspetest
Train_sensitivity<-results$rfsenstrain
Train_specificity<-results$rfspetrain
x<-seq(5,500, by=10)

df <- data.frame(x,Test_sensitivity,
                 Train_sensitivity ,Test_specificity,
                 Train_specificity) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("purple", "red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by ntree', x='Ntree',
       y='Sensitivity and Specificity')



#PLOT 2####
False_class_error_test<-results$rfeftest
False_class_error_train<-results$rfeftrain
True_class_error_test<-results$rfettest
True_class_error_train<-results$rfettrain
x<-seq(5,500, by=10)

df <- data.frame(x,False_class_error_test,
                 False_class_error_train ,True_class_error_test,
                 True_class_error_train) %>%
  gather(Measurement, value ='value', 2:5)
```

```
ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("purple","red","orange", "green","blue"))+
  labs(title='Error by ntree', x='Ntree', y='Class error')

#PLOT 3####
F1_test<- results$F1_test
F1_train<- results$F1_train
Gmean_test<- results$Gmean_test
Gmean_train<- results$Gmean_train
x<-seq(5,500, by=10)
df1 <- data.frame(x,F1_test, F1_train, Gmean_test, Gmean_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df1, aes(x=x, y=value, group=Measurement,
                colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by ntree', x='Ntree',
       y='F1 & Gmean')


####NODESIZE Using For loop to identify the right nodesize for model####
rfGrid <-  expand.grid(mtry = 6,
                       ntree = 500,
                       nodesize = seq(1,15, by=1),
                       max_depth  = 99)

t_ini<-Sys.time()
source('rf_calculations.R')
results<- rf_calculation(TRAIN,TEST,rfGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t#7.5 mins


x<-seq(1,15, by=1)
paste("Optimal value for number of nodesize:",
x[which.max(results$F1_test)]) ####15
paste("Optimal value for number of nodesize:",
x[which.max(results$Gmean_test)]) ####5

#PLOT 1####
Test_sensitivity<-results$rfsenstest
Test_specificity<-results$rfspetest
Train_sensitivity<-results$rfsenstrain
Train_specificity<-results$rfspetrain
x<-seq(1,15, by=1)

df <- data.frame(x,Test_sensitivity,
                 Train_sensitivity ,Test_specificity,
```

```
                  Train_specificity) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("purple", "red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by nodesize', x='Nodesize',
       y='Sensitivity and Specificity')



#PLOT 2####
False_class_error_test<-results$rfeftest
False_class_error_train<-results$rfeftrain
True_class_error_test<-results$rfettest
True_class_error_train<-results$rfettrain
x<-seq(1,15, by=1)
df <- data.frame(x,False_class_error_test,
                 False_class_error_train ,True_class_error_test,
                 True_class_error_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("purple","red","orange", "green","blue"))+
  labs(title='Error by nodesize', x='Nodesize', y='Class error')

#PLOT 3####
F1_test<- results$F1_test
F1_train<- results$F1_train
Gmean_test<- results$Gmean_test
Gmean_train<- results$Gmean_train
x<-seq(1,15, by=1)

df1 <- data.frame(x,F1_test, F1_train, Gmean_test, Gmean_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df1, aes(x=x, y=value, group=Measurement,
                colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by nodesize', x='Nodesize',
       y='F1 & Gmean')
```

rf_calculations

```
rf_calculation<-function(TRAIN,TEST,rfGrid){
  library(tidyverse)
  library(randomForest)
  library(ggplot2)
  TEST<- TEST[,-1]
  TRAIN<- TRAIN[,-1]
  set.seed(1)

  df<-data.frame(rfGrid,
                 F1_test=rep(-1, nrow(rfGrid)),
                 F1_train=rep(-1, nrow(rfGrid)),
                 rfsenstest=rep(-1, nrow(rfGrid)),
                 rfsenstrain=rep(-1, nrow(rfGrid)),
                 rfspetest=rep(-1, nrow(rfGrid)),
                 rfspetrain=rep(-1, nrow(rfGrid)),
                 rfprecisiontest=rep(-1, nrow(rfGrid)),
                 rfprecisiontrain=rep(-1, nrow(rfGrid)),
                 Gmean_test=rep(-1, nrow(rfGrid)),
                 Gmean_train=rep(-1, nrow(rfGrid)),
                 rfeftest=rep(-1, nrow(rfGrid)),
                 rfeftrain=rep(-1, nrow(rfGrid)),
                 rfettest=rep(-1, nrow(rfGrid)),
                 rfettrain=rep(-1, nrow(rfGrid)))
  for (i in seq(1,nrow(df), by=1)) {
    Accuracy<-c()
    rfsenstest<-c()
    rfsenstrain<-c()
    rfspetest<-c()
    rfspetrain<-c()
    rfeftest<-c()
    rfeftrain<-c()
    rfettest<-c()
    rfettrain<-c()
    rfprecisiontest<-c()
    rfprecisiontrain<-c()
    F1_test<-c()
    F1_train<-c()
    Gmean_test<-c()
    Gmean_train<- c()
    for (k in c(1,2,3,4,5)){
      set.seed(k*76+43)
      model <- randomForest(as.factor(Churn) ~ .,
                            data = TRAIN,
                            ntree = rfGrid$ntree[i] ,
                            mtry = rfGrid$mtry[i],
                            nodesize= rfGrid$nodesize[i],
                            max_depth= rfGrid$max_depth[i],
                            keep.forest = TRUE,
                            xtest = TEST[,-19],
```

```
                              ytest = as.factor(TEST$Churn),
                              importance=TRUE)
      predValid <- predict(model, TEST, type = "class")
      Accuracy<-c(Accuracy, mean(predValid == TEST$Churn))
      ####Error de la class False TRAIN
      rfeftrain<-c(rfeftrain, model[[5]][[5]])
      ####Error de la class True TRAIN
      rfettrain<-c(rfettrain, model[[5]][[6]])
      ####Error de la class False TEST
      rfeftest<-c(rfeftest, model[[17]][[3]][[5]])
      ####Error de la class True TEST
      rfettest<-c(rfettest,model[[17]][[3]][[6]])
      ####Sensitivity TRAIN
      rfsenstrain1<- model[[5]][[4]]/(model[[5]][[3]]+model[[5]][[4]])
      rfsenstrain<-c(rfsenstrain, rfsenstrain1)
      ####Specitivity TRAIN
      rfspetrain1<- model[[5]][[1]]/(model[[5]][[2]]+model[[5]][[1]])
      rfspetrain<-c(rfspetrain, rfspetrain1)
      ####Sensitivity TEST
      rfsenstest1<-
model[[17]][[3]][[4]]/(model[[17]][[3]][[3]]+model[[17]][[3]][[4]])
      rfsenstest<-c(rfsenstest, rfsenstest1)
      ####Specitivity TEST

rfspetest1<-model[[17]][[3]][[1]]/(model[[17]][[3]][[2]]+model[[17]][[3
]][[1]])

rfspetest<-c(rfspetest,model[[17]][[3]][[1]]/(model[[17]][[3]][[2]]+mod
el[[17]][[3]][[1]]))
      rfprecisiontest1<- 1-model[[17]][[3]][[6]]
      rfprecisiontest<-c(rfprecisiontest, rfprecisiontest1)
      rfprecisiontrain1<- 1-model[[5]][[6]]
      rfprecisiontrain<-c(rfprecisiontrain, rfprecisiontrain1)
      F1_test<-c(F1_test,((2*(rfprecisiontest1)*rfsenstest1/
                           ((rfprecisiontest1)+rfsenstest1))))
      F1_train<-c(F1_train, (2*rfprecisiontrain1*rfsenstrain1/
                              (rfprecisiontrain1+rfsenstrain1)))
      Gmean_test<-c(Gmean_test, sqrt(((rfsenstest1))*
                                       (rfspetest1)))
      Gmean_train<- c(Gmean_train, sqrt((rfsenstrain1)*
                                          (rfspetrain1)))
    }

    df$rfeftest[i]<-mean(rfeftest)
    df$rfeftrain[i]<-mean(rfeftrain)
    df$rfettest[i]<-mean(rfettest)
    df$rfettrain[i]<-mean(rfettrain)
    df$rfsenstest[i]<-mean(rfsenstest)
    df$rfsenstrain[i]<-mean(rfsenstrain)
    df$rfspetest[i]<-mean(rfspetest)
    df$rfspetrain[i]<-mean(rfspetrain)
    df$rfprecisiontest[i]<-mean(rfprecisiontest)
    df$rfprecisiontrain[i]<-mean(rfprecisiontrain)
```

```
    df$F1_test[i]<-mean(F1_test)
    df$F1_train[i]<-mean(F1_train)
    df$Gmean_test[i]<-mean(Gmean_test)
    df$Gmean_train[i]<-mean(Gmean_train)
  }
  return (df)
}
```

find_best_rf_model

```
find_best_rf_model<-function(TRAIN,TEST,rfGrid){
  library(tidyverse)
  library(randomForest)
  library(ggplot2)
  TEST<- TEST[,-1]
  TRAIN<- TRAIN[,-1]
  set.seed(1)

  df<-data.frame(rfGrid,
  F1_test=rep(-1, nrow(rfGrid)),
  F1_train=rep(-1, nrow(rfGrid)))


for (i in seq(1,nrow(df), by=1)) {
  F1_test<-c()
  F1_train<-c()
  for (k in c(1,2,3,4,5)){
    set.seed(k*76+43)
    model <- randomForest(as.factor(Churn) ~ .,
                          data = TRAIN,
                          ntree = rfGrid$ntree[i] ,
                          mtry = rfGrid$mtry[i],
                          nodesize= rfGrid$nodesize[i],
                          max_depth= rfGrid$max_depth[i],
                          keep.forest = TRUE,
                          xtest = TEST[,-19],
                          ytest = as.factor(TEST$Churn),
                          importance=TRUE)

    ####Sensitivity TRAIN
    rfsenstrain1<- model[[5]][[4]]/(model[[5]][[3]]+model[[5]][[4]])

    ####Sensitivity TEST
    rfsenstest1<-
model[[17]][[3]][[4]]/(model[[17]][[3]][[3]]+model[[17]][[3]][[4]])
    rfprecisiontest1<- 1-model[[17]][[3]][[6]]
    rfprecisiontrain1<- 1-model[[5]][[6]]

    F1_test<-c(F1_test,((2*(rfprecisiontest1)*rfsenstest1/
                           ((rfprecisiontest1)+rfsenstest1))))
    F1_train<-c(F1_train, (2*rfprecisiontrain1*rfsenstrain1/
                      (rfprecisiontrain1+rfsenstrain1)))

  }

  df$F1_test[i]<-mean(F1_test)
  df$F1_train[i]<-mean(F1_train)
}
best_model<-df[which(df$F1_test== max(df$F1_test, na.rm = t)),]
return (list(df, best_model))
}
```

XGBoosting_v4

```
#EXTREME GRADIENT BOOSTING####


#LOADING DATA####
setwd("C:/Users/pc/Desktop/tfm")
library(tidyverse)
TRAIN<-read.csv("churn-bigml-80.csv")
TEST<-read.csv("churn-bigml-20.csv")
TOTAL<-rbind(TRAIN,TEST)

#EXTREME GRADIENT BOOSTING####
install.packages("xgboost")
install.packages("caTools")
install.packages("dplyr")
install.packages("Matrix")
library(ggplot2)
library(lattice)
library(xgboost)
library(caTools)
library(dplyr)
library(caret)
library(Matrix)
library(xgboost)



#DATA CLEANING FOR XGBOOSTING####

TESTX<-TEST %>% mutate(Churn=recode(Churn, 'False'=0, 'True'=1))
TESTX<-TESTX %>% mutate(Voice.mail.plan=recode(Voice.mail.plan,
                                               'No'=0,
                                               'Yes'=1))
TESTX<-TESTX %>% mutate(International.plan=recode(International.plan,
                                                  'No'=0,
                                                  'Yes'=1))
TESTX<- TESTX %>% mutate(Area.code=recode(Area.code, '408'=0, '415'=1,
                                          '510'=2))
TESTX<-TESTX %>% mutate(State=recode(State, 'AK'=0, 'AL'=1,  'AR'=2,
'AZ'=3,  'CA'=4,
                                     'CO'=5, 'CT'=6,  'DC'=7,  'DE'=8,
'FL'=9,
                                     'GA'=10, 'HI'=11, 'IA'=12, 'ID'=13,
'IL'=14,
                                     'IN'=15, 'KS'=16, 'KY'=17, 'LA'=18,
'MA'=19,
                                     'MD'=20, 'ME'=21, 'MI'=22, 'MN'=23,
'MO'=24,
                                     'MS'=25, 'MT'=26, 'NC'=27, 'ND'=28,
'NE'=29,
```

```
                                        'NH'=30, 'NJ'=31, 'NM'=32, 'NV'=33,
'NY'=34,
                                        'OH'=35, 'OK'=36, 'OR'=37, 'PA'=38,
'RI'=39,
                                        'SC'=40, 'SD'=41, 'TN'=42, 'TX'=43,
'UT'=44,
                                        'VA'=45, 'VT'=46, 'WA'=47, 'WI'=48,
'WV'=49,
                                        'WY'=50))
TRAINX<-TRAIN %>% mutate(Churn=recode(Churn, 'False'=0, 'True'=1))
TRAINX<-TRAINX %>% mutate(Voice.mail.plan=recode(Voice.mail.plan,
                                                 'No'=0,
                                                 'Yes'=1))
TRAINX<-TRAINX %>% mutate(International.plan=recode(International.plan,
                                                    'No'=0,
                                                    'Yes'=1))
TRAINX<- TRAINX %>% mutate(Area.code=recode(Area.code, '408'=0, '415'=1,
                                            '510'=2))
TRAINX<-TRAINX %>% mutate(State=recode(State, 'AK'=0, 'AL'=1,  'AR'=2,
'AZ'=3,  'CA'=4,
                                       'CO'=5, 'CT'=6,  'DC'=7,  'DE'=8,
'FL'=9,
                                       'GA'=10, 'HI'=11, 'IA'=12,
'ID'=13, 'IL'=14,
                                       'IN'=15, 'KS'=16, 'KY'=17,
'LA'=18, 'MA'=19,
                                       'MD'=20, 'ME'=21, 'MI'=22,
'MN'=23, 'MO'=24,
                                       'MS'=25, 'MT'=26, 'NC'=27,
'ND'=28, 'NE'=29,
                                       'NH'=30, 'NJ'=31, 'NM'=32,
'NV'=33, 'NY'=34,
                                       'OH'=35, 'OK'=36, 'OR'=37,
'PA'=38, 'RI'=39,
                                       'SC'=40, 'SD'=41, 'TN'=42,
'TX'=43, 'UT'=44,
                                       'VA'=45, 'VT'=46, 'WA'=47,
'WI'=48, 'WV'=49,
                                       'WY'=50))
train_set <- TRAINX
test_set <- TESTX
X_train<-data.matrix(train_set[-20])
X_test<-data.matrix(test_set[-20])
y_train<-data.matrix(train_set$Churn)
y_test<-data.matrix(test_set$Churn)
xgb_train <- xgb.DMatrix(data = X_train, label = y_train)
xgb_test <- xgb.DMatrix(data = X_test, label = y_test)


#INITIAL XGBOOST MODEL####
bst <- xgboost(data = xgb_train, max.depth = 6, eta = 0.3, nthread = 2,
               nrounds = 2, objective = "binary:logistic", verbose = 1,
               eval_metric='error')
```

```
#TRAIN PREDICTION####
y_pred <- predict(bst,xgb_train)
lvl = c('False','.True')
y_pred_label <- lvl[as.numeric(y_pred>0.5)+1]
y_actual_label <- lvl[as.numeric(TRAINX$Churn)+1]
#Confusion matrix TRAIN####
conf<-table(y_pred_label,y_actual_label)

#Sensitivity TRAIN####
sensitivitytrain<- conf[[1]]/(conf[[1]]+conf[[2]])

#Specificity TRAIN####
specificitytrain<- conf[[4]]/(conf[[3]]+conf[[4]])

#Confusion matrix with error TRAIN####
cbind(conf,"Error"=c(sensitivitytrain,specificitytrain))

#TEST PREDICTION####
y_pred <- predict(bst,xgb_test)
lvl = c('False','.True')
y_pred_label <- lvl[as.numeric(y_pred>0.5)+1]
y_actual_label <- lvl[as.numeric(TESTX$Churn)+1]
#Confusion matrix TEST####
conf<-table(y_pred_label,y_actual_label)

#Sensitivity TEST####
sensitivitytest<- conf[[1]]/(conf[[1]]+conf[[2]])

#Specificity TEST####
specificitytest<- conf[[4]]/(conf[[3]]+conf[[4]])

#Confusion matrix with error TEST####
cbind(conf,"Performance metrics"=c(sensitivitytest,specificitytest))



#PLOTS####
#BEST MODEL####
source('xgb_calculations_v3.R')

xgbGrid <-  expand.grid(max_depth = (1:10),
                        eta = seq(0.1 ,1, by=0.1),
                        gamma = c(0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
0.8, 0.9, 1),
                        subsample = 1,
                        min_child_weight = c(0,1,3,4,5,6,7,10,20,30),
                        colsample_bytree = 0.6,
                        nrounds= c(1, 2, 3, 4, 5, 6, 10))
source('find_best_model.R')
t_ini<-Sys.time()
```

```
bb<-find_best_model(xgb_train,xgb_test, TESTX, xgbGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t
#Time difference of 3.768923 hours
"""
max_depth eta gamma subsample min_child_weight colsample_bytree nrounds
F1_test
   8      0.6   0.7         1                0              0.6      10
0.8850575
F1_train
 0.9802891
"""

#Grid analysis####
#to save the results of the grid search in csv
#write.csv2(xgb_results_grid_search, 'xgb_dades_10092022.csv')
#to load these results on R Studio
xgb_results_grid_search<-read.csv2('xgb_dades_10092022.csv')
#for removing Na results in cases they exist
xgb_results_grid_search<-xgb_results_grid_search[!is.na(xgb_results_grid
_search$F1_test),]
xgb_results_grid_search$max_depth<-factor(xgb_results_grid_search$max_de
pth)
xgb_results_grid_search$eta<-factor(xgb_results_grid_search$eta)
xgb_results_grid_search$gamma<-factor(xgb_results_grid_search$gamma)
xgb_results_grid_search$min_child_weight<-factor(xgb_results_grid_search
$min_child_weight)
xgb_results_grid_search$nrounds<-factor(xgb_results_grid_search$nrounds)
xgb_overfit<-xgb_results_grid_search$F1_test-xgb_results_grid_search$F1_
train
xgb_overfit_percent<-(xgb_results_grid_search$F1_test-xgb_results_grid_s
earch$F1_train)*100/xgb_results_grid_search$F1_train
summary(xgb_results_grid_search)
summary(xgb_overfit)
summary(xgb_overfit_percent)

#Plot
layout(matrix(c(1:3), nrow=1, byrow=TRUE))
hist(xgb_results_grid_search$F1_test,main="F1 test", xlab='')
hist(xgb_results_grid_search$F1_train, main="F1 train", xlab='')
hist(xgb_overfit_percent, main="Overfitting (%)", xlim=c(-27,
125),xlab='')

#MAX DEPTH  Using For loop to identify the right max.depth for model####

t_ini<-Sys.time()
xgbGrid <-  expand.grid(max_depth = (1:10),
                        nrounds = 10,    # number of trees
                        # default values below
                        eta = 0.3,
                        gamma = 0,
                        subsample = 1,
```

```
                        min_child_weight = 5,
                        colsample_bytree = 0.6)

source('xgb_calculations_v3.R')
results<- xgb_calculation(xgb_train,xgb_test, TESTX,xgbGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t #4.3 s
x<-seq(1,10, by=1)
paste("Optimal value for max depth:", x[which.max(results$F1_test)]) #6
paste("Optimal value for max depth:", x[which.max(results$Gmean_test)])
#6


#PLOT 1####
Test_sensitivity<-results$senstest
Test_specificity<-results$spetest
Train_sensitivity<-results$senstrain
Train_specificity<-results$spetrain
x<-seq(1,10, by=1)

df <- data.frame(x,Test_sensitivity,
                 Train_sensitivity ,Test_specificity,
                 Train_specificity) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c( "red","orange", "green","blue"))+
  labs(title='Performance metrics by max depth', x='Max depth',
       y='Sensitivity and Specificity')




#PLOT 2####
False_class_error_test<-results$eftest
False_class_error_train<-results$eftrain
True_class_error_test<-results$ettest
True_class_error_train<-results$ettrain
x<-seq(1,10, by=1)



df <- data.frame(x,False_class_error_test,
                 False_class_error_train ,True_class_error_test,
                 True_class_error_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
```

```
  scale_color_manual(values=c("red","orange", "green","blue"))+
  labs(title='Error by max depth', x='Max depth',
       y='Class error')

#PLOT 3####
x<-seq(1,10, by=1)

F1_test<- results$F1_test
F1_train<- results$F1_train
Gmean_test<- results$Gmean_test
Gmean_train<- results$Gmean_train

df1 <- data.frame(x,F1_test, F1_train,
                  Gmean_test, Gmean_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df1, aes(x=x, y=value, group=Measurement,
                colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by max depth', x='Max depth',
       y='F1 & Gmean')

#ETA Using For loop to identify the right eta for model####

t_ini<-Sys.time()
xgbGrid <-  expand.grid(max_depth = 6,
                        nrounds = 10,    # number of trees
                        # default values below
                        eta = seq(0.1,1, by=0.1),
                        gamma = 0,
                        subsample = 1,
                        min_child_weight = 5,
                        colsample_bytree = 0.6)

source('xgb_calculations_v3.R')
results<- xgb_calculation(xgb_train,xgb_test, TESTX,xgbGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t #6.71 s
x<-seq(0.1,1, by=0.1)
paste("Optimal value for eta:", x[which.max(results$F1_test)]) #0.3
paste("Optimal value for eta:", x[which.max(results$Gmean_test)]) #0.3

#PLOT 1####
Test_sensitivity<-results$senstest
Test_specificity<-results$spetest
Train_sensitivity<-results$senstrain
Train_specificity<-results$spetrain
x<-seq(0.1,1, by=0.1)

df <- data.frame(x,Test_sensitivity,
```

```
                  Train_sensitivity ,Test_specificity,
                  Train_specificity) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
                colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c( "red",
                               "orange", "green","blue"))+
  labs(title='Performance metrics by eta', x='Eta',
       y='Sensitivity and Specificity')

#PLOT 2####
False_class_error_test<-results$eftest
False_class_error_train<-results$eftrain
True_class_error_test<-results$ettest
True_class_error_train<-results$ettrain


df <- data.frame(x,False_class_error_test,
                  False_class_error_train ,True_class_error_test,
                  True_class_error_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
                colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red","orange", "green","blue"))+
  labs(title='Error by eta', x='Eta', y='Class error')

#PLOT 3####
x<-seq(0.1,1, by=0.1)

F1_test<- results$F1_test
F1_train<- results$F1_train
Gmean_test<- results$Gmean_test
Gmean_train<- results$Gmean_train

df1 <- data.frame(x,F1_test, F1_train,
                   Gmean_test, Gmean_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df1, aes(x=x, y=value, group=Measurement,
                 colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by eta', x='Eta',
       y='F1 & Gmean')

#GAMMA Using For loop to identify the right gamma for model####
t_ini<-Sys.time()
xgbGrid <-  expand.grid(max_depth = 6,
```

```
                        nrounds = 10,     # number of trees
                        # default values below
                        eta = 0.3,
                        gamma =c(0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
0.8, 0.9, 1),
                        subsample = 1,
                        min_child_weight = 5,
                        colsample_bytree = 0.6)

source('xgb_calculations_v3.R')
results<- xgb_calculation(xgb_train,xgb_test, TESTX,xgbGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t#4.55 secs
x<-c(0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)
paste("Optimal value for gamma:", x[which.max(results$F1_test)]) #0
paste("Optimal value for gamma:", x[which.max(results$Gmean_test)]) #0

#PLOT 1####
Test_sensitivity<-results$senstest
Test_specificity<-results$spetest
Train_sensitivity<-results$senstrain
Train_specificity<-results$spetrain


df <- data.frame(x,Test_sensitivity,
                 Train_sensitivity ,Test_specificity,
                 Train_specificity) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by gamma', x='Gamma',
       y='Sensitivity and Specificity')

#PLOT 2####
False_class_error_test<-results$eftest
False_class_error_train<-results$eftrain
True_class_error_test<-results$ettest
True_class_error_train<-results$ettrain
c(0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

df <- data.frame(x,False_class_error_test,
                 False_class_error_train ,True_class_error_test,
                 True_class_error_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
```

```
  scale_color_manual(values=c("red","orange", "green","blue"))+
  labs(title='Error by gamma', x='Gamma', y='Class error')

#PLOT 3####
c(0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

F1_test<- results$F1_test
F1_train<- results$F1_train
Gmean_test<- results$Gmean_test
Gmean_train<- results$Gmean_train

df1 <- data.frame(x,F1_test, F1_train, Gmean_test, Gmean_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df1, aes(x=x, y=value, group=Measurement,
                colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by gamma', x='Gamma',
       y='F1 & Gmean')

#MIN CHILD WEIGHT Using For loop to identify the right min_child_weight
for model####
t_ini<-Sys.time()
xgbGrid <-  expand.grid(max_depth = 6,
                        nrounds = 10,    # number of trees
                        # default values below
                        eta = 0.3,
                        gamma = 0,
                        subsample = 1,
                        min_child_weight =
c(0,2,3,4,5,6,7,10,20,30,40,50,52),
                        colsample_bytree = 0.6)

source('xgb_calculations_v3.R')
results<- xgb_calculation(xgb_train,xgb_test, TESTX,xgbGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t #5.73 s
x<-c(0,2,3,4,5,6,7,10,20,30,40,50,52)
paste("Optimal value for min_child_weight:",
x[which.max(results$F1_test)]) #0
paste("Optimal value for min_child_weight:",
x[which.max(results$Gmean_test)]) #0


#PLOT 1####
Test_sensitivity<-results$senstest
Test_specificity<-results$spetest
Train_sensitivity<-results$senstrain
Train_specificity<-results$spetrain
```

```
df <- data.frame(x,Test_sensitivity,
                 Train_sensitivity ,Test_specificity,
                 Train_specificity) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by min_child_weight', x=' Min child
weight',
       y='Sensitivity and Specificity')

#PLOT 2####
False_class_error_test<-results$eftest
False_class_error_train<-results$eftrain
True_class_error_test<-results$ettest
True_class_error_train<-results$ettrain

df <- data.frame(x,False_class_error_test,
                 False_class_error_train ,True_class_error_test,
                 True_class_error_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red","orange", "green","blue"))+
  labs(title='Error by min_child_weight', x='Min child weight',
       y='Class error')

#PLOT 3####
x<-c(0,2,3,4,5,6,7,10,20,30,40,50,52)

F1_test<- results$F1_test
F1_train<- results$F1_train
Gmean_test<- results$Gmean_test
Gmean_train<- results$Gmean_train

df1 <- data.frame(x,F1_test, F1_train, Gmean_test, Gmean_train) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df1, aes(x=x, y=value, group=Measurement,
                colour=Measurement ) ) +
  geom_line(size=0.75)+
  scale_color_manual(values=c("red",
                              "orange", "green","blue"))+
  labs(title='Performance metrics by min child weight', x='Min child
weight',
       y='F1 & Gmean')
```

xgb_calculations_v3

```
xgb_calculation<- function(xgb_train,xgb_test, TESTX,xgbGrid){
  library(ggplot2)
  library(lattice)
  library(xgboost)
  library(caTools)
  library(dplyr)
  library(caret)
  library(Matrix)
  library(xgboost)
  df<-data.frame(xgbGrid,
                 F1_test=rep(-1, nrow(xgbGrid)),
                 F1_train=rep(-1, nrow(xgbGrid)),
                 senstest=rep(-1, nrow(xgbGrid)),
                 senstrain=rep(-1, nrow(xgbGrid)),
                 spetest=rep(-1, nrow(xgbGrid)),
                 spetrain=rep(-1, nrow(xgbGrid)),
                 precisiontest=rep(-1, nrow(xgbGrid)),
                 precisiontrain=rep(-1, nrow(xgbGrid)),
                 Gmean_test=rep(-1, nrow(xgbGrid)),
                 Gmean_train=rep(-1, nrow(xgbGrid)),
                 eftest=rep(-1, nrow(xgbGrid)),
                 eftrain=rep(-1, nrow(xgbGrid)),
                 ettest=rep(-1, nrow(xgbGrid)),
                 ettrain=rep(-1, nrow(xgbGrid)))
  set.seed(123)
  for (i in seq(1:nrow(df))){
    senstest<-c()
    senstrain<-c()
    spetest<-c()
    spetrain<-c()
    precisiontest<-c()
    precisiontrain<-c()
    F1_test<-c()
    F1_train<-c()
    Gmean_test<-c()
    Gmean_train<- c()
    eftest<-c()
    eftrain<-c()
    ettest<-c()
    ettrain<-c()
    for (j in c(1,2,3,4,5)){
      set.seed(j*30+58)
      bst1 <- xgboost(data = xgb_train, max.depth =
xgbGrid$max_depth[i],
                      gamma=xgbGrid$gamma[i], eta =xgbGrid$eta[i],
                      min_child_weight=xgbGrid$min_child_weight[i],
                      nthread = 2, nrounds = xgbGrid$nrounds[i],
objective = "binary:logistic",
```

```
                verbose = 0, eval_metric='error')

y_pred <- predict(bst1,xgb_test)
lvl = c('False','.True')
y_pred_label <- lvl[as.numeric(y_pred>0.5)+1]
y_actual_label <- lvl[as.numeric(TESTX$Churn)+1]
#Confusion matrix TEST
conf<-table(y_pred_label,y_actual_label)

if (dim(conf)[1] == 1){
  if(y_pred_label[1]=='False'){
    #Error de la class False TEST
    errorfalsetest<- conf[[1]]/(conf[[1]]+conf[[2]])
    #Error de la class True TEST
    errortruetest<- 1
    #Sensitivity TEST
    sensitivitytest<- 0
    #Specificity TEST
    specificitytest<- 1
    #Precision TEST
    precisiontest1<- 0
  } else {
    #Error de la class False TEST
    errorfalsetest<- 1
    #Error de la class True TEST
    errortruetest<-conf[[2]]/(conf[[1]]+conf[[2]])
    #Sensitivity TEST
    sensitivitytest<- 1
    #Specificity True TEST
    specificitytest<-0
    #Precision TEST
    precisiontest1<-conf[[1]]/(conf[[1]]+conf[[2]])
  }
} else {
  #Error de la class False TEST
  errorfalsetest<- conf[[2]]/(conf[[2]]+conf[[4]])
  #Error de la class True TEST
  errortruetest<- conf[[3]]/(conf[[1]]+conf[[3]])
  #Sensitivity TEST
  sensitivitytest<- conf[[1]]/(conf[[1]]+conf[[2]])
  #Specificity TEST
  specificitytest<- conf[[4]]/(conf[[3]]+conf[[4]])
  #Precision TEST
  precisiontest1<- conf[[1]]/(conf[[1]]+conf[[3]])
}


y_pred <- predict(bst1,xgb_train)
lvl = c('False','.True')
y_pred_label <- lvl[as.numeric(y_pred>0.5)+1]
y_actual_label <- lvl[as.numeric(TRAINX$Churn)+1]
#Confusion matrix TRAIN
conf<-table(y_pred_label,y_actual_label)
```

```
if (dim(conf)[1] == 1){
  if(y_pred_label[1]=='False'){
    #Error de la class False TRAIN
    errorfalsetrain<- conf[[1]]/(conf[[1]]+conf[[2]])
    #Error de la class True TRAIN
    errortruetrain<- 1
    #Sensitivity TRAIN
    sensitivitytrain<- 0
    #Specificity TRAIN
    specificitytrain<- 1
    #Precision TRAIN
    precisiontrain1<-0
  } else {
    #Error de la class False TRAIN
    errorfalsetrain<- 1
    #Error de la class True TRAIN
    errortruetrain<-conf[[2]]/(conf[[1]]+conf[[2]])
    #Sensitivity TRAIN
    sensitivitytrain<- 1
    #Specificity TRAIN
    specificitytrain<-0
    #Precision TRAIN
    precisiontrain1<-conf[[1]]/(conf[[1]]+conf[[2]])
  }
} else {
  #Error de la class False TRAIN
  errorfalsetrain<- conf[[2]]/(conf[[2]]+conf[[4]])
  #Error de la class True TRAIN
  errortruetrain<- conf[[3]]/(conf[[1]]+conf[[3]])
  #Sensitivity TRAIN
  sensitivitytrain<- conf[[1]]/(conf[[1]]+conf[[2]])
  #Specificity TRAIN
  specificitytrain<- conf[[4]]/(conf[[3]]+conf[[4]])
  #Precision TRAIN
  precisiontrain1<-conf[[1]]/(conf[[1]]+conf[[3]])
}
eftest<-c(eftest,errorfalsetest)
eftrain<-c(eftrain, errorfalsetrain)
ettest<-c(ettest, errortruetest)
ettrain<-c(ettrain, errortruetrain)
senstest<-c(senstest,sensitivitytest)
senstrain<-c(senstrain, sensitivitytrain)
spetest<-c(spetest, specificitytest)
spetrain<-c(spetrain, specificitytrain)
precisiontest<-c(precisiontest, precisiontest1)
precisiontrain<-c(precisiontrain, precisiontrain1)
F1_test<-c(F1_test,((2*(precisiontest1)*sensitivitytest/
                       ((precisiontest1)+sensitivitytest))))
F1_train<-c(F1_train, (2*precisiontrain1*sensitivitytrain/
                         (precisiontrain1+sensitivitytrain)))
Gmean_test<-c(Gmean_test, sqrt(((sensitivitytest))*
                                  (specificitytest)))
```

```
      Gmean_train<- c(Gmean_train, sqrt((sensitivitytrain)*
                                          (specificitytrain)))
  }
    df$eftest[i]<-mean(eftest)
    df$eftrain[i]<-mean(eftrain)
    df$ettest[i]<-mean(ettest)
    df$ettrain[i]<-mean(ettrain)
    df$senstest[i]<-mean(senstest)
    df$senstrain[i]<-mean(senstrain)
    df$spetest[i]<-mean(spetest)
    df$spetrain[i]<-mean(spetrain)
    df$precisiontest[i]<-mean(precisiontest)
    df$precisiontrain[i]<-mean(precisiontrain)
    df$F1_test[i]<-mean(F1_test)
    df$F1_train[i]<-mean(F1_train)
    df$Gmean_test[i]<-mean(Gmean_test)
    df$Gmean_train[i]<-mean(Gmean_train)

  }

  return (df)

}
```

find_best_model

```
find_best_model<-function(xgb_train,xgb_test, TESTX,xgbGrid){
  library(ggplot2)
  library(lattice)
  library(xgboost)
  library(caTools)
  library(dplyr)
  library(caret)
  library(Matrix)
  library(xgboost)


  df<-data.frame(xgbGrid,
  F1_test=rep(-1, nrow(xgbGrid)),
  F1_train=rep(-1, nrow(xgbGrid)))
  set.seed(123)
  for (i in seq(1:nrow(df))){
      F1_test<-c()
      F1_train<-c()
      for (j in c(1,2,3,4,5)){
          set.seed(j*40+58)
          bst1 <- xgboost(data = xgb_train, max.depth =
xgbGrid$max_depth[i],
                          gamma=xgbGrid$gamma[i], eta =xgbGrid$eta[i],
                          min_child_weight=xgbGrid$min_child_weight[i],
                          nthread = 2, nrounds = xgbGrid$nrounds[i],
objective = "binary:logistic",
                          verbose = 0, eval_metric='error')
          y_pred <- predict(bst1,xgb_test)
          lvl = c('False','.True')
          y_pred_label <- lvl[as.numeric(y_pred>0.5)+1]
          y_actual_label <- lvl[as.numeric(TESTX$Churn)+1]
          #Confusion matrix TEST
          conf<-table(y_pred_label,y_actual_label)

          if (dim(conf)[1] == 1){
           if(y_pred_label[1]=='False'){
             #Sensitivity TEST
             sensitivitytest<- 0
             #Precision TEST
             precisiontest1<- 0
           } else {
             #Sensitivity TEST
             sensitivitytest<- 1
             #Precision TEST
             precisiontest1<-conf[[1]]/(conf[[1]]+conf[[2]])
           }
          } else {
```

```
            #Sensitivity TEST
            sensitivitytest<- conf[[1]]/(conf[[1]]+conf[[2]])
            #Precision TEST
            precisiontest1<- conf[[1]]/(conf[[1]]+conf[[3]])
           }


           y_pred <- predict(bst1,xgb_train)
           lvl = c('False','.True')
           y_pred_label <- lvl[as.numeric(y_pred>0.5)+1]
           y_actual_label <- lvl[as.numeric(TRAINX$Churn)+1]
           #Confusion matrix TRAIN
           conf<-table(y_pred_label,y_actual_label)

           if (dim(conf)[1] == 1){
            if(y_pred_label[1]=='False'){
              #Sensitivity TRAIN
              sensitivitytrain<- 0
              #Precision TRAIN
              precisiontrain1<-0
            } else {
              #Sensitivity TRAIN
              sensitivitytrain<- 1
              #Precision TRAIN
              precisiontrain1<-conf[[1]]/(conf[[1]]+conf[[2]])
            }
           } else {
            #Sensitivity TRAIN
            sensitivitytrain<- conf[[1]]/(conf[[1]]+conf[[2]])
            #Precision TRAIN
            precisiontrain1<-conf[[1]]/(conf[[1]]+conf[[3]])

           }
           F1_test<-c(F1_test,((2*(precisiontest1)*sensitivitytest/
                                ((precisiontest1)+sensitivitytest))))
           F1_train<-c(F1_train, (2*precisiontrain1*sensitivitytrain/
                                   (precisiontrain1+sensitivitytrain)))

         }

       df$F1_test[i]<-mean(F1_test)
       df$F1_train[i]<-mean(F1_train)

  }

  best_model<-df[which(df$F1_test== max(df$F1_test, na.rm=t)),]
  return (list(df, best_model))
}
```

comparing_best_models

```
#RANDOM FOREST####

####LOADING DATA####
setwd("C:/Users/pc/Desktop/tfm")
library(tidyverse)
library(randomForest)
library(ggplot2)
TRAIN<-read.csv("churn-bigml-80.csv")
TEST<-read.csv("churn-bigml-20.csv")
TOTAL<-rbind(TRAIN,TEST)

####DATA CLEANING FOR RANDOM FOREST####
TEST$State<- factor(TEST$State)
TEST$International.plan<- factor(TEST$International.plan)
TEST$Voice.mail.plan<- factor(TEST$Voice.mail.plan)
TEST$Churn<- factor(TEST$Churn)
TEST$Area.code<- factor(TEST$Area.code)
TRAIN$State<- factor(TRAIN$State)
TRAIN$International.plan<- factor(TRAIN$International.plan)
TRAIN$Voice.mail.plan<- factor(TRAIN$Voice.mail.plan)
TRAIN$Churn<- factor(TRAIN$Churn)
TRAIN$Area.code<- factor(TRAIN$Area.code)
TOTAL$State<- factor(TOTAL$State)
TOTAL$International.plan<- factor(TOTAL$International.plan)
TOTAL$Voice.mail.plan<- factor(TOTAL$Voice.mail.plan)
TOTAL$Churn<- factor(TOTAL$Churn)
TOTAL$Area.code<- factor(TOTAL$Area.code)




#randomforest default model####
"""
mtry ntree nodesize   F1_test  F1_train
 4   500         1 0.8176191 0.8248909
"""
rfGrid <-  expand.grid(mtry = 4,
                       ntree = 500,
                       nodesize = 1)

t_ini<-Sys.time()
source('rf_calculations.R')
rf_results_1<- rf_calculation(TRAIN,TEST,rfGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t
set.seed(189)
source('find_best_rf_model.R')
rf_results_2<- find_best_rf_model(TRAIN,TEST,rfGrid)
```

```
set.seed(189)
TEST<- TEST[,-1]  #ejecutar si TEST y TRAIN tienen 20 variables
TRAIN<- TRAIN[,-1] #ejecutar si TEST y TRAIN tienen 20 variables
model <- randomForest(as.factor(Churn) ~ .,
                      data = TRAIN,
                      ntree = 500 ,
                      mtry = 4,
                      nodesize= 1,
                      max_depth= 99,
                      keep.forest = TRUE,
                      xtest = TEST[,-19],
                      ytest = as.factor(TEST$Churn),
                      importance=1,
                      verbose=2)


#randomforest best model####
"""
mtry ntree nodesize   F1_test  F1_train
  7   175        3 0.8463146 0.8272176
"""
rfGrid <-  expand.grid(mtry = 7,
                       ntree = 175,
                       nodesize = 3)
set.seed(189)
t_ini<-Sys.time()
source('rf_calculations.R')
rf_best_results_1<- rf_calculation(TRAIN,TEST,rfGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini

set.seed(189)
source('find_best_rf_model.R')
rf_best_results_2<- find_best_rf_model(TRAIN,TEST,rfGrid)



set.seed(189)
TEST<- TEST[,-1]  #ejecutar si TEST y TRAIN tienen 20 variables
TRAIN<- TRAIN[,-1] #ejecutar si TEST y TRAIN tienen 20 variables
rf_best_model <- randomForest(as.factor(Churn) ~ .,
                        data = TRAIN,
                        ntree = 175 ,
                        mtry = 7,
                        nodesize= 3,
                        max_depth= 7,
                        keep.forest = TRUE,
                        xtest = TEST[,-19],
                        ytest = as.factor(TEST$Churn),
                        importance=1,
                        verbose=2)
```

```
#IMPORTANCE PLOTS####

varImpPlot(rf_model, sort=TRUE)
varImpPlot(rf_best_model, sort= TRUE)

#XGBOOSTING####
library(caret)
library(Matrix)
library(xgboost)
TRAIN<-read.csv("churn-bigml-80.csv")
TEST<-read.csv("churn-bigml-20.csv")
TOTAL<-rbind(TRAIN,TEST)

#DATA CLEANING FOR XGBOOSTING####

TESTX<-TEST %>% mutate(Churn=recode(Churn, 'False'=0, 'True'=1))
TESTX<-TESTX %>% mutate(Voice.mail.plan=recode(Voice.mail.plan,
                                               'No'=0,
                                               'Yes'=1))
TESTX<-TESTX %>% mutate(International.plan=recode(International.plan,
                                                  'No'=0,
                                                  'Yes'=1))
TESTX<- TESTX %>% mutate(Area.code=recode(Area.code, '408'=0, '415'=1,
                                          '510'=2))
TESTX<-TESTX  %>%  mutate(State=recode(State,  'AK'=0,  'AL'=1,    'AR'=2,
'AZ'=3,  'CA'=4,
                                       'CO'=5, 'CT'=6,  'DC'=7,  'DE'=8,
'FL'=9,
                                      'GA'=10, 'HI'=11, 'IA'=12, 'ID'=13,
'IL'=14,
                                      'IN'=15, 'KS'=16, 'KY'=17, 'LA'=18,
'MA'=19,
                                      'MD'=20, 'ME'=21, 'MI'=22, 'MN'=23,
'MO'=24,
                                      'MS'=25, 'MT'=26, 'NC'=27, 'ND'=28,
'NE'=29,
                                      'NH'=30, 'NJ'=31, 'NM'=32, 'NV'=33,
'NY'=34,
                                      'OH'=35, 'OK'=36, 'OR'=37, 'PA'=38,
'RI'=39,
                                      'SC'=40, 'SD'=41, 'TN'=42, 'TX'=43,
'UT'=44,
                                      'VA'=45, 'VT'=46, 'WA'=47, 'WI'=48,
'WV'=49,
                                     'WY'=50))
TRAINX<-TRAIN %>% mutate(Churn=recode(Churn, 'False'=0, 'True'=1))
TRAINX<-TRAINX %>% mutate(Voice.mail.plan=recode(Voice.mail.plan,
                                                 'No'=0,
                                                 'Yes'=1))
TRAINX<-TRAINX %>% mutate(International.plan=recode(International.plan,
```

```
                                                    'No'=0,
                                                    'Yes'=1))
TRAINX<- TRAINX %>% mutate(Area.code=recode(Area.code, '408'=0, '415'=1,
                                                '510'=2))
TRAINX<-TRAINX  %>% mutate(State=recode(State,  'AK'=0,  'AL'=1,   'AR'=2,
'AZ'=3,  'CA'=4,
                                        'CO'=5, 'CT'=6,  'DC'=7,  'DE'=8,
'FL'=9,
                                                'GA'=10, 'HI'=11, 'IA'=12,
'ID'=13, 'IL'=14,
                                                'IN'=15, 'KS'=16, 'KY'=17,
'LA'=18, 'MA'=19,
                                                'MD'=20, 'ME'=21, 'MI'=22,
'MN'=23, 'MO'=24,
                                                'MS'=25, 'MT'=26, 'NC'=27,
'ND'=28, 'NE'=29,
                                                'NH'=30, 'NJ'=31, 'NM'=32,
'NV'=33, 'NY'=34,
                                                'OH'=35, 'OK'=36, 'OR'=37,
'PA'=38, 'RI'=39,
                                                'SC'=40, 'SD'=41, 'TN'=42,
'TX'=43, 'UT'=44,
                                                'VA'=45, 'VT'=46, 'WA'=47,
'WI'=48, 'WV'=49,
                                        'WY'=50))
train_set <- TRAINX
test_set <- TESTX
X_train<-data.matrix(train_set[-20])
X_test<-data.matrix(test_set[-20])
y_train<-data.matrix(train_set$Churn)
y_test<-data.matrix(test_set$Churn)
xgb_train <- xgb.DMatrix(data = X_train, label = y_train)
xgb_test <- xgb.DMatrix(data = X_test, label = y_test)

#xgboot model####


#xgboost default model####
"""
max_depth nrounds eta gamma subsample min_child_weight colsample_bytree
F1_test  F1_train
1            6          2  0.3       0            1                     1
0.6 0.8333333 0.8567416
"""
t_ini<-Sys.time()
xgbGrid <-  expand.grid(max_depth = 6,
                        nrounds = 2,
                        eta = 0.3,
                        gamma = 0,
                        subsample = 1,
                        min_child_weight = 1,
                        colsample_bytree = 0.6)
```

```
source('xgb_calculations_v3.R')
xgb_results_1<- xgb_calculation(xgb_train,xgb_test, TESTX,xgbGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t
source('find_best_model.R')
t_ini<-Sys.time()
xgb_results_2<-find_best_model(xgb_train,xgb_test, TESTX, xgbGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t
set.seed(189)
xgb_model <- xgboost(data = xgb_train, max.depth = 6,
                          gamma=0, eta =0.3,
                          min_child_weight=1,
                                  nthread = 2, nrounds = 2, objective =
"binary:logistic",
                          verbose = 0, eval_metric='error')
y_pred <- predict(xgb_model,xgb_test)
lvl = c('False','.True')
y_pred_label <- lvl[as.numeric(y_pred>0.5)+1]
y_actual_label <- lvl[as.numeric(TESTX$Churn)+1]
#Confusion matrix TEST
conf<-table(y_pred_label,y_actual_label)


#xgboost best model####
"""
 max_depth nrounds eta gamma subsample min_child_weight colsample_bytree
F1_test  F1_train
 8          10      0.6   0.7         1                0              0.6
0.8850575 0.9802891
"""
xgbGrid <-  expand.grid(max_depth = 8,
                        nrounds = 10,
                        eta = 0.6,
                        gamma = 0.7,
                        subsample = 1,
                        min_child_weight = 0,
                        colsample_bytree = 0.6)

t_ini<-Sys.time()
source('xgb_calculations_v3.R')
xgb_best_results_1<- xgb_calculation(xgb_train,xgb_test, TESTX,xgbGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t
source('find_best_model.R')
t_ini<-Sys.time()
xgb_best_results_2<-find_best_model(xgb_train,xgb_test, TESTX, xgbGrid)
t_fi<- Sys.time()
t<- t_fi-t_ini
t
```

```
set.seed(88)
xgb_best_model <- xgboost(data = xgb_train, max.depth = 8,
                gamma=0.6, eta =0.7,
                min_child_weight=0,
                            nthread = 2, nrounds = 10, objective =
"binary:logistic",
                verbose = 0, eval_metric='error')

y_pred <- predict(xgb_best_model,xgb_test)
lvl = c('False','.True')
y_pred_label <- lvl[as.numeric(y_pred>0.5)+1]
y_actual_label <- lvl[as.numeric(TESTX$Churn)+1]
#Confusion matrix TEST
conf<-table(y_pred_label,y_actual_label)

#PLOT 1####
xgb_model_1<-c(xgb_results_1$F1_test, xgb_results_1$F1_train)
rf_model_1<-c(rf_results_1$F1_test, rf_results_1$F1_train)
xgb_model_best<-c(xgb_best_results_1$F1_test,
xgb_best_results_1$F1_train)
rf_model_best<-c(rf_best_results_1$F1_test, rf_best_results_1$F1_train)
x<-c('F1_test', 'F1_train')

df    <-    data.frame(x,xgb_model_1,    xgb_model_best,    rf_model_1,
rf_model_best) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_point(size=2.5)+
  scale_color_manual(values=c( "red","blue", "green", "orange"))+
    labs(title='Performance  evaluation  of  F1  Score',  x='Measurement
paramethers')
"""
DEFAULT RANDOM FOREST
mtry ntree nodesize   F1_test  F1_train
 4   500        1 0.8176191 0.8248909
 CUSTOM RANDOM FOREST
mtry ntree nodesize   F1_test  F1_train
  7   175        3 0.8463146 0.8272176
  XGB DEFAULT
max_depth  nrounds  eta  gamma  subsample  min_child_weight  colsample_bytree
F1_test  F1_train
 6          2      0.3     0         1                 1              0.6
0.8333333 0.8567416
CUSTOM XGB
 max_depth nrounds eta gamma subsample min_child_weight colsample_bytree
F1_test  F1_train
 8         10      0.6   0.7         1                 0              0.6
0.8850575 0.9802891
"""
#PLOT 2####
```

```
xgb_model_1<-c(xgb_results_1$senstest,
xgb_results_1$senstrain,1-xgb_results_1$ettest, 1-xgb_results_1$ettrain)
rf_model_1<-c(rf_results_1$rfsenstest,
rf_results_1$rfsenstrain,1-rf_results_1$rfettest,1-rf_results_1$rfettrai
n)
xgb_model_best<-c(xgb_best_results_1$senstest,
xgb_best_results_1$senstrain,1-xgb_best_results_1$ettest,
1-xgb_best_results_1$ettrain)
rf_model_best<-c(rf_best_results_1$rfsenstest,
rf_best_results_1$rfsenstrain,1-rf_best_results_1$rfettest,
1-rf_best_results_1$rfettrain)
x<-c('Sensitivity test', 'Sensitivity train', 'Precision test',
"Precision train")

df <- data.frame(x,xgb_model_1, xgb_model_best, rf_model_1,
rf_model_best) %>%
  gather(Measurement, value ='value', 2:5)

ggplot(df, aes(x=x, y=value, group=Measurement,
               colour=Measurement ) ) +
  geom_point(size=2.5)+
  scale_color_manual(values=c( "red","blue", "green", "orange"))+
    labs(title='Performance evaluation', x='Measurement paramethers',
y='')

"""
rf_model_1
[1] 0.9499623 0.9406408 0.7178947 0.7345361
> rf_model_best
[1] 0.9650383 0.9293109 0.7536842 0.7453608
> xgb_model_1
[1] 0.7894737 0.7860825 0.8823529 0.9413580
> xgb_model_best
[1] 0.8105263 0.9613402 0.9746835 1.0000000
"""
```