Master Thesis

Master's degree in Management Engineering

Decision Tree Statistical Learning Models. An application to New Customer Scoring

Report

Author:	Macià Comella Barbé
Director:	Ignasi Puig de Dou
Term:	Q1 2019-2020

Barcelona School of Industrial Engineering
ETSEIB

(Page left blank for printing purposes)

Summary

The aim of this thesis is to explore, understand and apply statistical learning methods based on decision trees, specifically individual decision trees and bagging, random forests and gradient boosting methods. In order to do this, a research has been done and the theory behind each one of these methods understood. The main sources of information are the books “Introduction to Statistical Learning” and “The Elements of Statistical Learning” by T. Hastie, R. Tibshirani and J. Friedman.
Afterwards this theory is put to practice using a real case data set and the R programming language to experiment with models of the mentioned methods. The data used comes from a real case project in which a business wishes to predict whether a new customer will be a good one based only in the information from its three first purchases.
The tools used are also presented, consisting in the different R packages and functions used and its tuning parameters. The strategy used in order to obtain representative results that make possible to understand the concepts presented in the theory is explained. As well as how these results have been extracted. The sensitivity analysis has been done with the Minitab v18 software, provided by the Universitat Politècnica de Catalunya for research purposes.

Finally the results are analysed. This analysis is divided in three sections.
The first one is focused in a sensitivity analysis of parameters. The results show that, with the used dataset, for gradient boosting the tree depth allowed is critical to obtain a good quality of fit and prevent overfitting, and the number of iterations allowed needs to be correctly aligned with the learning parameter used. The results for bagging and random forests (merged as one is a particular case of the other) prove the lack of overfitting intrinsic of these models and discovers that if the number of variables is high and these are strongly correlated the recommended number of variables to choose at each tree node does not lead to optimum results. An initial hypothesis to guide the analysis of this fact is proposed but it is not inside the scope of the project to analyse and prove this hypothesis.
The second section of the analysis consists in selecting the best performing method and apply it to the available dataset. The gradient boosting method is chosen as the best one due to higher quality of fit obtained and a more consistent selection of variables among all scenarios.
The third section compares the results obtained with gradient boosting versus the logistic regression model done by the student P. Casas in his bachelor thesis “New customers’ classifier” based on the same dataset. The results show that gradient boosting performs better in terms of prediction in two of the three models created, though the difference is small, and obtains the same quality of fit in the other case. Comparing variable relevance the most important one is shared among both methods (the total value of the purchase). Other secondary variables are shared and some of them not. Therefore it can be said there is similarity in general terms but gradient boosting and logistic regression are not totally close between them, as it happens with the decision tree methods used in the project.

(Page left blank for printing purposes)

Index

1 Introduction

Data science and statistical learning are becoming a part of everyone’s daily life. Knowing it or not, these models suggest the next video or film to watch in video streaming services or select the best advertises to display in the side of a webpage. And it is expected that the range of possibilities to provide will grow exponentially in the future.

Among the numerous types of statistical methods, some of them have increased their popularity lately. An example of these are methods based on decision trees, which have become relevant as they have been used to win various data science competitions like the Higgs Machine Learning Challenge and several Kaggle competitions (Kaggle is an important global online community of data scientists and machine learning engineers that holds data science competitions among other activities).

For these reasons the scope of this project is to study, learn and experiment with decision tree based statistical learning methods. Specifically the methods selected are bagging, random forests and gradient boosting. This study will contain theory explanations and afterwards this theory will be put into practice with the creation of models based on real case data using R language. The structure followed will be:

- A theoretical study will be done where the concepts of these three decision tree methods will be explained. As well as other intermediate statistical concepts needed for the development of the project.
- The available dataset will be presented, explained and basically analysed. This will allow to understand the data being used. This data has been provided by the department of Applied Statistics at the Industrial Engineering School of Barcelona and the origin of the data was a company (a mechanical spare parts distributor) which requested an algorithm to predict customer purchasing potential based on the information of the first three purchases done by the customer. Hence this is a classification problem and for this reason the whole project will be focused in classification models (models with binary response outcome).
- The methodology and tools for the creation of the models with R, data extraction and analysis will be explained in a dedicated chapter.
- The obtained results will be analysed considering two main characteristics: quality of fit (prediction power) and variable importance analysis. At this point it will also be possible to compare the best decision tree model obtained with a logistic regression model developed in a previous project [1] of the same school based on the used data.
- Finally conclusions will be presented and next steps will be proposed.

2 Theoretical framework

2.1 Basic concepts of statistical learning

The aim of this theory section is to present the research done in the main topic of this project, statistical learning models based on decision trees. This theory will underpin the experimental analysis based on a real case data explained afterwards. However, before starting this explanation some basic concepts will be introduced as they will be used later.

2.1.1 Validating statistical learning models

A basic concept of statistical learning is assessing the quality of fit of a model. A usual method in regression models is the Mean Standard Error (MSE), given by

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

where y_i is the real value for observation i and $\hat{f}(x_i)$ is the predicted value for the same observation. This is calculated for all n observations. In this formula the higher the difference between the real value and the predicted value the higher the MSE will be.

Usually a model is created based on a dataset of observations and then its quality is assessed by calculating the MSE of the same observations compared to the values predicted by the model. The main issue with this procedure is that the MSE used is assessing the quality of the model based on the same data the model has been trained. Hence the model could perfect fit the data if it is complicated enough, and actually is quite common if one is not careful. This is known as overfitting.

This is not good at all as in this situation the model will be really good predicting the already known observations, but it will perform poorly when faced with new data. And this is the actual objective of creating a model, since one wants to predict unknown data and not the already available.

This issue is solved by separating the dataset in two different samples. One will be used to create, or train, the model. This sample will be named training sample. The other will remain unknown to the model until the moment of assessing the quality of fit. Then the MSE, or any other error indicator, will be calculated using this second sample. This second sample will be named test sample.

This split in the dataset could be done in a myriad of proportions. 50/50, 25% test sample and 75% training sample or any other partition possible. In this project though a slightly more complicated approach has been used. Instead of splitting the dataset only once, several splits are made. For instance, an example with five samples will be explained here, but any other number is possible with the maximum value of the total number of observations available. These five samples would contain 20% of the observations each and each observation would be randomly assigned to one group. This random selection appears in figure 2.1 as the number randomly sorted from all n observations. Then, to train the model four of these samples would be used, for example samples 2, 3, 4 and 5. This is represented in figure 2.1 by the blue sections in the first row of data. Afterwards the sample number 1 would be used to test the model and obtain an error indicator, like the MSE. This is represented in figure 2.1 by the initial orange area.

After this, the process will be repeated but this time samples 1, 3, 4 and 5 would be used to train the model and sample 2 would be the test one (second coloured row of figure 2.1). This process would be done five times until all samples have been used as test sample. At this point five error indicators would be available for the five samples used as test. The returned error indicator would be the average of all of them.

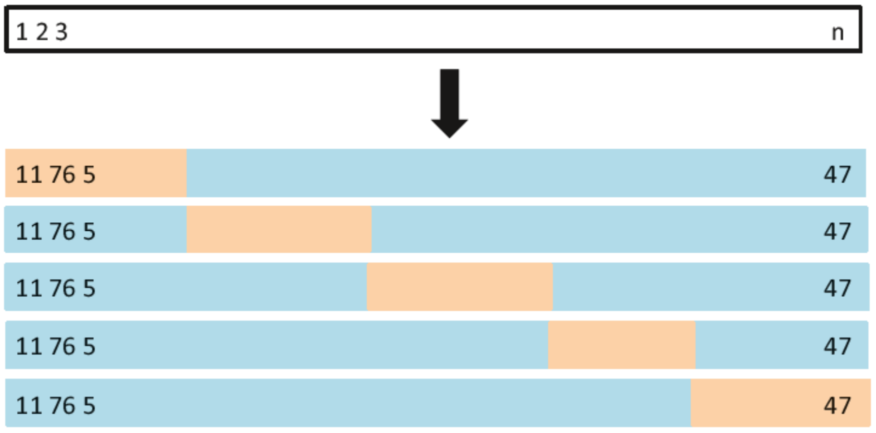


Figure 2.1: example of 5-fold cross validation. Source: [2]

This approach allows a robust estimate of the test error even in reduced datasets as all observations are used both for test and training. This reduces the variance of the indicator because if only a small number of observations are used to calculate the test error indicator the value of this indicator can vary widely depending on the observations selected as test.

This method is called k-fold cross-validation, where k indicates the number of samples, or folds, created from the dataset. The example above consists of a 5-fold cross-validation.

2.1.2 Error indicator used: ROC curve and AUC

The second concept that needs to be explained in this introductory theory chapter is the indicator used in this project to evaluate the quality of fit of the models created, which needs to be usable also with other statistical methods not based on decision trees for comparing purposes.

The dataset available for this project consists in the classification of good customers vs non-good customers of a business (for simplicity the term bad customers will be used although almost all customers are good for a company). Hence this is a classification problem with a binary response. The MSE cannot be used here. Instead the Area Under the ROC Curve (AUC) will be used. But to explain the AUC one needs to introduce first the concepts of **sensitivity** and **specificity**.

In a binary classification problem an observation can be either 0 or 1 (or any other binary classification like good/bad or yes/no), and the model created can classify the same observation as 0 or 1. Therefore there are 4 possible scenarios.

Prediction/observation	0	1
0	(1) Correct	(2) Wrong
1	(3) Wrong	(4) Correct

Table 2.1: possible situations in a binary classification problem

The **sensitivity** of a model is defined as the percentage of 1's identified as 1's by the model, (4) in table 2.1. The sensitivity of a model measures the ability to identify that "something" happens (predicted value = 1) when "something" actually happens (observed value = 1).

The **specificity** consists of the percentage of 0's identified as 0's by the model, (1) in table 2.1. In this case would be to identify that "something" is not happening (predicted value = 0) when actually that "something" is not happening (observed value = 0).

This indicators provide extra information versus the commonly used misclassification error (observations incorrectly classified over the total) as it takes into account that both classes (0's and 1's) are correctly predicted by the model. This is important as in certain problems one class might be more interesting than the other. For example it might be important to identify a good customer as good, but it might not be critical if some bad customers are misclassified as good customers. This is important also because if the dataset is biased and contains only a few observations of a certain class (for example patients with a disease over the total population) a trivial model like classifying all observations in the major class would obtain a good misclassification error but of course will be considered a poor model. As usually the objective is to identify the minority of the dataset, i.e. the patients with a disease over all the sane individuals.

The values of sensitivity and specificity are not fixed for a given model and dataset, they depend on the chosen classification threshold. Let's explain why. Usually a classification model returns a probability per each observation predicted, after introducing the prediction variables X_i the model returns a probability that the response is 1, for example. Table 2.2 shows an example of 5 predictions made by a model. It contains the predicted probability of the response Y being 1 $\hat{f}(X)$, based on the predictor variables x_1 and x_2 and the real value of the observation y_i.

i	**x_1**	**x_2**	**$\hat{f}(X)$**	**y_i**
1	1	9	0.9	1
2	3	5	0.6	1
3	4	5	0.4	0
4	1	2	0.3	1
5	2	3	0.2	0

Table 2.2: example of binary classification prediction problem

If the problem faced has a binary nature, each response y_i can only be 0 or 1, good or bad for example. It is not valid to predict that the given observation has a y of 0.6, it must be either 0 or

1. One “natural” way to solve this (the first that comes to one’s mind) is that any returned probability higher than 0.5 will be predicted as 1, and below 0.5 will be predicted as 0. If this 0.5 threshold is used on the previous observations, observations 1 and 2 will be classified as 1 (as $\hat{f}(X) \geq 0.5$), hence correctly classified, but observation 4 will be classified as 0 though it should be 1, so it will be incorrect. Therefore sensitivity will be 2/3 or 66%. Specificity will be 100% as all 0’s will be classified correctly.

However, if another threshold is used sensitivity and specificity will change. For example if all returned values with 0.3 or higher are classified as 1’s sensitivity would be 100% (the result would be three 1’s correct out of 3) but specificity would be 50% (one 1 correct out of 2). Using an extreme threshold of 0.0, that is classifying all observations as good, would deliver a sensitivity of 100% and a specificity of 0%. As it is possible to observe an increase of sensitivity results in a decrease in specificity.

If all possible combinations of sensitivity and specificity obtained for all classification thresholds are considered it is possible to create a curve. This is called ROC curve, (ROC comes from telecommunications theory and means Receiver Operating Characteristics). The ROC curve for the observations in table 2.2 is represented in figure 2.2 below. Typically the y axis of the ROC curve displays the sensitivity, or the true positive rate, and the x-axis displays 1-specificity, or the false positive rate.

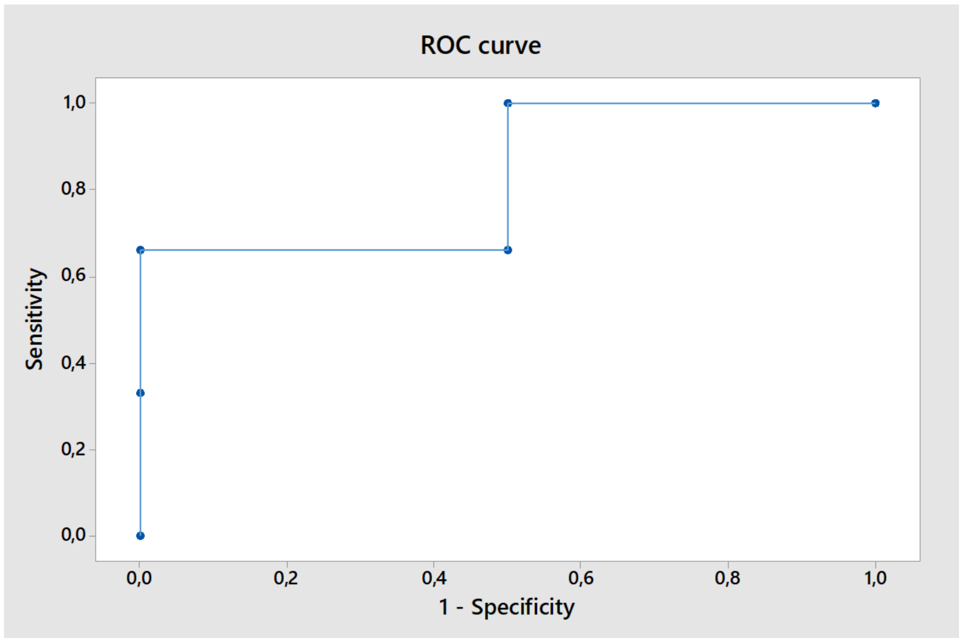


Figure 2.2: example of ROC curve based on table 2.2. Source: own creation

Every model has a unique ROC curve for the same predicted data. The overall performance of a classifier, summarized over all possible thresholds, is given by the Area Under the ROC Curve (AUC). An ideal ROC curve will reach the top left corner, so the larger the AUC the better the classifier. The AUC for the model in table 2.2 is 83% which is certainly good.

The AUC value (83% in the example) can be understood like this: given 2 random observations each from a different class, one 1 and one 0, the model will be able to distinguish correctly the good from the bad the (AUC value) percent of the times. In the example the model will distinguish correctly good from bad given two individuals 83% of the times.

ROC curves are useful for comparing different classifiers, since they take into account all possible thresholds. For this reason the AUC will be the indicator used to compare the models obtained in this project with the ones obtained by the student P. Casas using a logistic regression model based on the same real case dataset.

2.2 Decision trees

2.2.1 Introduction

Tree based methods involve stratifying or segmenting the predictor space (X's) into a number of simple regions. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision tree methods.

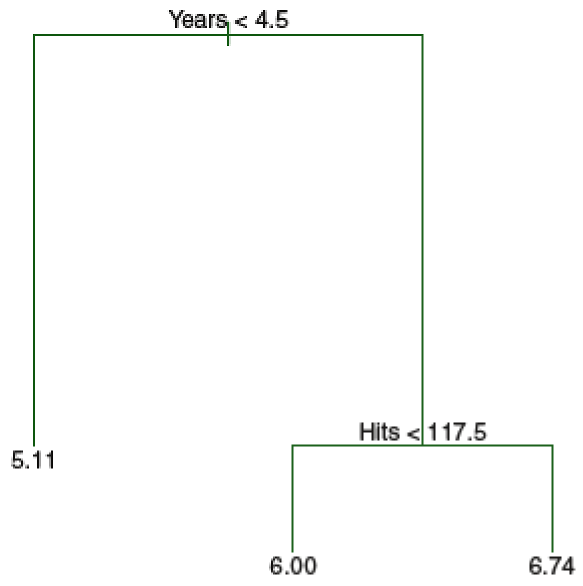


Figure 2.3: example of a regression decision tree. Source: [2]

Figure 2.3 shows an example of a regression tree. The data related to this tree classifies the salary of baseball players, in millions of dollars, based on the Years they have been playing and the number of Hits they did the last season. The tree consists of a series of splitting rules, starting at the top of the tree. In this example, the top split assigns players having Years<4.5 to the left branch. Since there are no more splits in that branch the predicted value of all observations with Years<4.5 is 5.11 million $. Players with Years>=4.5 are assigned to the right branch, and then that group is further subdivided by Hits, depending whether Hits is higher or lower than 117.5.

Overall, the tree segments the players into three regions of the predictor space: players who have played for four or fewer years earn an average of 5.11M$, players who have played for five or more years and who made fewer than 118 hits last year earn 6M$, and players who have played for five or more years and made at least 118 hits last year earn 6.74M$.

Keeping with the tree analogy, the three regions of the predictor space are known as terminal nodes or leafs of the tree. Typically decision trees are displayed upside down as leafs are placed in the bottom of the figure. The points along the tree where the predictor space is split are referred to as internal nodes. In Figure 2.3, the two internal nodes are indicated by the text Years<4.5 and Hits<117.5. The segments of the trees that connect the nodes are named branches.

The interpretation of the overall figure would be: Years is the most important factor in determining a player's Salary. Players with less experience earn lower salaries than more experienced players. Given that a player is less experienced, the number of hits he made in the previous year has minimum weight in his salary (for this tree the weight is actually zero). Among players who have been in the major leagues for five or more years, the number of hits made in the previous year does influence salary. Therefore players who made more hits last year tend to have higher salaries.

The strong points of decision trees are ease of interpretation and nice graphical representation, as long as the number of nodes is not too high.

2.2.2 Regression tree

Roughly speaking, there are two steps to build a decision tree:

1. The predictor space is divided. So, the set of possible values for every variable X_1, X_2, …, X_p is grouped into J distinct and non-overlapping regions, R_1, R_2, …, R_j.
2. For every observation that falls into the region R_j, the same prediction is made, which is simply the mean of the response values for the training observations in R_j.

For instance, suppose two regions are obtained, R_1 and R_2, and that the response mean of the training observations in the first region is 10, while the response mean of the training observations in the second region is 20. Then for a given observation X = x, if $x \in R_1$ it will be predicted a value of 10, and if $x \in R_2$ a value of 20.

How are regions R_1, …, R_J constructed? In theory, the regions could have any shape. However, it is chosen to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model. The goal is to find boxes R_1, …, R_J that minimize the Residual Sum of Squares (RSS), given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

where $\hat{y}_{Rj}$ is the mean response for the training observations within the jth box.

Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes. For this reason, it is taken a top-down, greedy approach that is known as recursive binary splitting. This approach is top-down because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space. It is binary because each split is indicated via two new branches further down on the tree. Finally, it is greedy because at each step of the tree-building process, the best split is made considering that specific step rather than looking ahead and picking a split that will lead to a better tree in a future step.

In order to perform recursive binary splitting, first the predictor X_j and the cutpoint s are selected so that splitting the predictor space into the regions $\{X \mid X_j < s\}$ and $\{X \mid X_j \geq s\}$ leads to the greatest possible reduction in RSS. That is, all predictors $X_1, \ldots, X_p$, and all possible values of the cutpoint s for each of the predictors are considered, then the predictor and cutpoint that results in the lowest RSS is chosen.

Then the process is repeated, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions. However, this time, instead of splitting the entire predictor space, one of the two previously identified regions is split. There are now three regions. Again, one of these regions is further split to minimise the RSS. The process continues until a stopping criterion is reached; for instance, it may continue until no region contains more than five observations.

One of the main drawbacks of decision trees is that they tend to overfit the sample if tree size is too high. As displayed in figure 2.4 when tree size is initially increased, both training error and test error are reduced. But at a certain point (at the size or depth of three in this example), further increasing tree depth does not further reduce the test error, although training error continues to decrease.

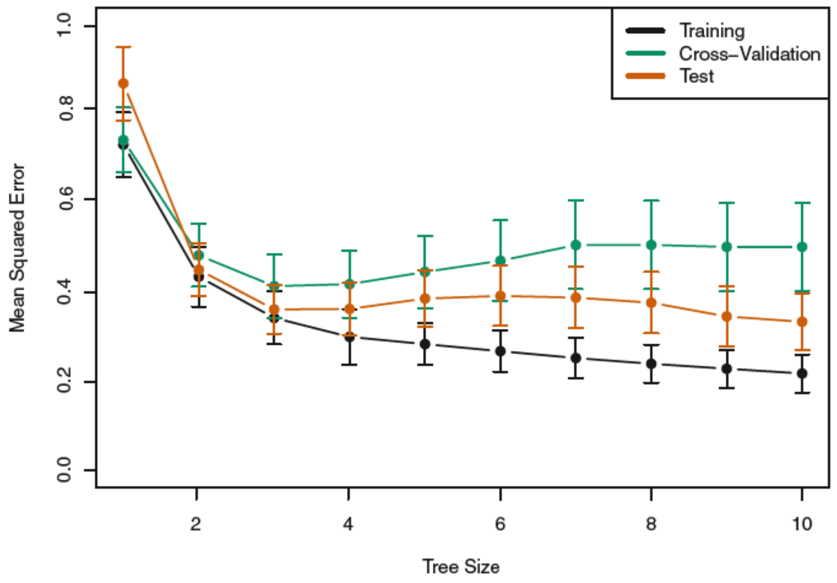


Figure 2.4: example of MSE vs tree size. MSE's displayed: training, test and cross-validation. Source: [2]

2.2.3 Classification trees

A classification tree is very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one. In contrast to regression trees, for a classification tree it is assumed that each observation belongs to the most commonly occurring class, based on training observations, in the region to which it belongs.

Growing a classification tree is quite similar to growing a regression tree. Just as in the regression setting, a recursive binary splitting is used to grow a classification tree. Since it will be assigned every observation in a given region to the most common class of training observations in that region, the classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k (\hat{p}_{mk})$$

Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. However, it turns out that classification error is not sufficiently sensitive for tree-growing, and in practice another measure is preferable [2], this is the Gini index. The Gini index is defined by:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

The Gini index takes a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason the Gini index is referred to as a measure of node purity, a small value indicates that a node contains predominantly observations from a single class. This would be equivalent to say that the error for that node is low. As low mixture of classes inside a node implies that most of the observations in that node are correctly classified, as observations in a node are classified all in a single class.

Tree-based methods are simple and useful for interpretation. However, they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy. Hence it is introduced here the concept of bagging, random forests, and boosting. Each of these more advanced approaches involve producing multiple trees which are then combined to yield a single consensus prediction.

2.3 Bagging

2.3.1 Bootstrap

The core of bagging models is the bootstrap process. The bootstrap is a powerful statistical tool used originally to estimate the standard deviation of a quantity of interest. It consists in randomly re-sampling the data available to obtain different datasets without the cost of obtaining additional data. This process allows to obtain infinite new datasets from the original one. The key point of bootstrap is re-sampling **with replacement.** An example of this procedure is shown in figure 2.5 below. The first new dataset obtained from the original 3 observations contains only observation number 1 once and number 3 twice. In this first re-sampled set, observation number 2 does not appear. This process is repeated three times in this example but could be extended at will.

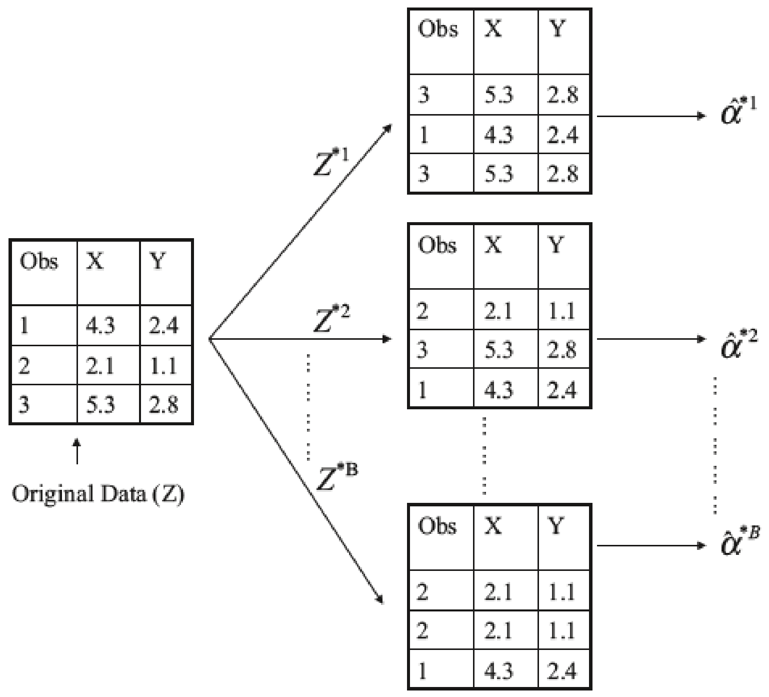


Figure 2.5: example of bootstrap sampling. Source: [2]

2.3.2 Building a bagging model

One of the drawbacks of decision trees is the high variability of the models obtained. If the training dataset is changed the resulting tree may be quite different from the initial, especially if the tree is grown deep. This results in low performance when predictions using new data are done. One option to reduce this variability of the model is to apply the bootstrap concept and resample n times the available dataset. Then, with this n datasets n trees can be created. Hence each individual tree has high variance, but low bias. Since the dataset used will be different each time, the obtained tree will differ from the rest. The final model consists of a decision committee composed by the n trees created. Averaging these n trees reduces the variance of the overall model.

When a new observation is given the committee of n trees provides n different predictions of the new data. The final result is calculated by averaging the prediction of all trees in case of regression models. In case of classification the result is given either by the majority vote of all trees or by the percentage of votes of each option over the total number of votes. It has been proved that, overall, the sum of a considerable number of highly variable but low biased decision makers obtains better results than an excellent individual decision maker. This principle is used in bagging but it can be applied in various situations.

2.3.3 Lack of overfitting

Unless a single tree or boosting methods, described later, bagged trees do not overfit the data independently of the number of trees used or the depth of these trees. Of course one single tree would overfit the training dataset, but since each tree is grown independently of the previous one and all trees are averaged this overfitting is cancelled. As a result, when creating a bagging model one needs only to ensure enough trees are grown to cancel the variability of each single tree, hence introducing diversity in the model and obtaining the maximum quality of fit. Once the test error has settled the model will not improve, nor worsen, if more trees are added. This behaviour is shown in figure 2.6.

2.3.4 Out-of-Bag error estimation

An interesting feature of bagging is that due to the bootstrap application it turns out that there is a very straightforward way to estimate the test error of a bagged model, without the need to perform cross-validation or the validation set approach. One can show that on average, each bagged tree makes use of around two-thirds of the observations [2]. The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations. Thanks to this, one can predict the response for the i^{th} observation using each of the trees in which that observation was OOB. This will yield around (nº of trees)/3 predictions for the i^{th} observation. In order to obtain a single prediction for the i^{th} observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal). This leads to a single OOB prediction for the i^{th} observation. An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB MSE (for a regression problem) or classification error (for a classification problem) can be computed. The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation. Although the absolute value of the OOB error estimation might differ, the important information is that it shows correctly the point where the error values become steady. OOB error estimation can be really useful in cases when the used dataset is really large, in which case performing cross-validation would be computationally expensive.

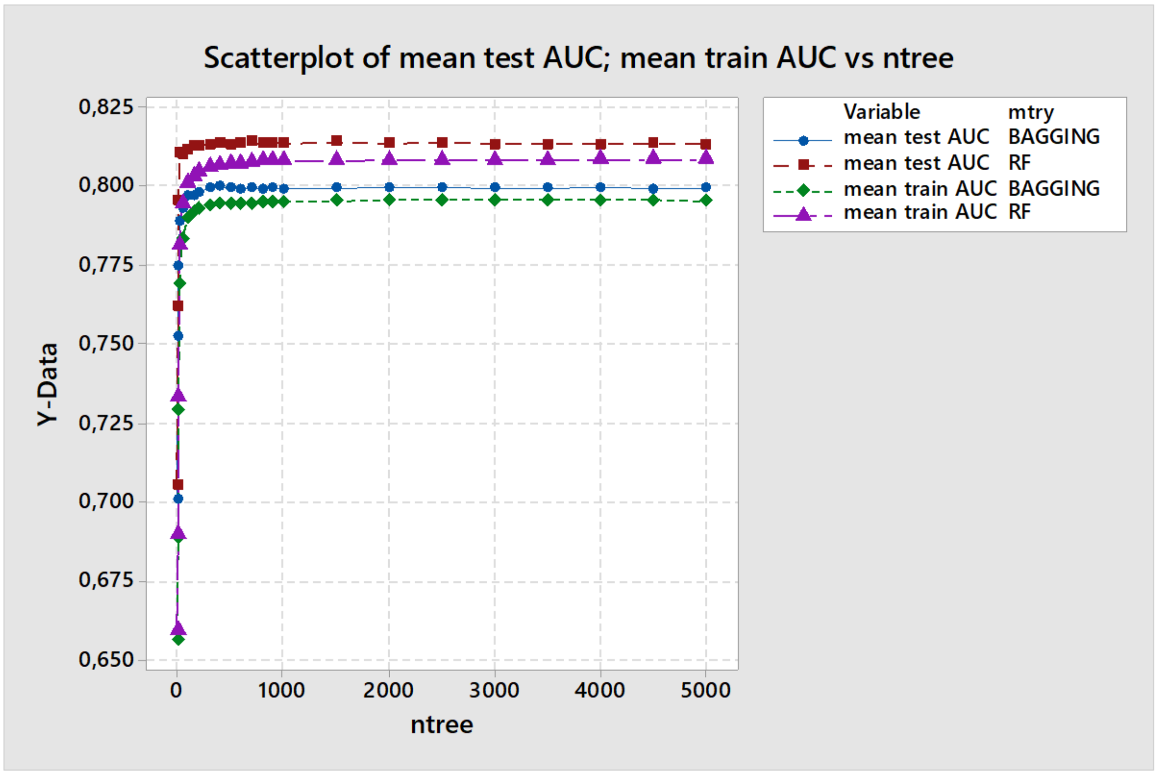


Figure 2.6: Test and training AUC vs number of trees in random forests and bagging models. Source: own creation

2.3.5 Interpretability and variable importance measures

Up to this point everything written about bagging are positive features. Where is the negative part of bagging? What is lost when aggregating hundreds or thousands of these decision trees? The main drawback is the loss of interpretability. Although the model is made of trees and a single decision tree can be clearly plotted and its decisions easily understood, when hundreds of trees are combined this plot is no longer reasonable.

However, there is still room for estimations of variable importance, although the level of interpretability might not be comparable to a single tree model or a linear model. To understand how this is achieved it is necessary to remember how a single tree is built. The predictor space is divided in J regions, R_J, for each possible value of each variable X_p. Then the splitting value of X_p that obtains the highest reduction in RSS (for regression) or Gini index (for classification) is chosen. To obtain a relative variable importance measure it is possible to sum, for each variable X_p, the error reduction obtained every time this variable was chosen to split the predictor space. This can be done for each tree. To obtain the variable importance in the overall model the average importance over all trees is calculated. Since these values are relative, it is usual to assign an importance of 100 to the largest value and calculate the other rest relative to this.

2.4 Random forests (RF)

Random forests provide an improvement over bagged trees using a small tweak that decorrelates the trees. As in bagging, a “forest” of decision trees is built on bootstrapped training samples. The difference is that when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The algorithm is allowed to choose from those m predictors, not from all predictors available in the dataset. A fresh sample of m available predictors is created at each node. Usually $m \approx \sqrt{p}$ is recommended for classification and $m \approx p/3$ for regression [2].

In other words, in building a random forest, at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors. This may sound senseless, but it has a clever rationale behind. Suppose that there is one very strong predictor in the data set, along with a number of other moderate predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split. Consequently, all of the bagged trees will look quite similar to each other. Then the predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.

Random forests overcome this problem by forcing each split to consider only a subset of the predictors. Therefore, on average $(p - m)/p$ of the splits will not even consider the strong predictor [2], and so other predictors will have a chance to add some value to the model. We can think of this process as decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable.

The main difference between bagging and random forests is the choice of predictor subset size m. For instance, if a random forest is built using $m = p$, then this amounts simply to bagging. Since the underlying process is the same as for bagging, the same properties stand for random forests. Random forests do not overfit the training set and out-of-bag error estimation is available. Interpretability is also strongly reduced in Random Forests, although the variable importance calculation as a workaround is still possible.

2.5 Gradient boosting methods (GBM)

Bagging and random forests tackled decision trees high variability by averaging hundreds or thousands of trees. Now, with boosting, another strategy to tackle this variability is explained. The idea behind boosting is a general concept of statistical learning that can be applied in various scenarios, however one of the main applications is combined with decision trees. For this reason this chapter is divided in two sections. First, the generic boosting concept is presented, afterwards its specific application with decision trees is explained.

2.5.1 The concept of boosting

The idea behind boosting is a model that learns slowly. Instead of trying to create the whole model in a single attempt, a first rough approximation is done. Then the model tries to improve the previous approximation by predicting the residuals between the original data and the last prediction made. This process is repeated until a stopping criteria is met. It is easy to see that as long as each steps improves the solution, even if only slightly, eventually the model will fit the data. Due to this approach the model will overfit the training dataset if the number of iterations is too high.

Now an illustrative example is presented to better understand how boosting works. The example consists in creating a model to predict the below function, represented in figure 2.7.

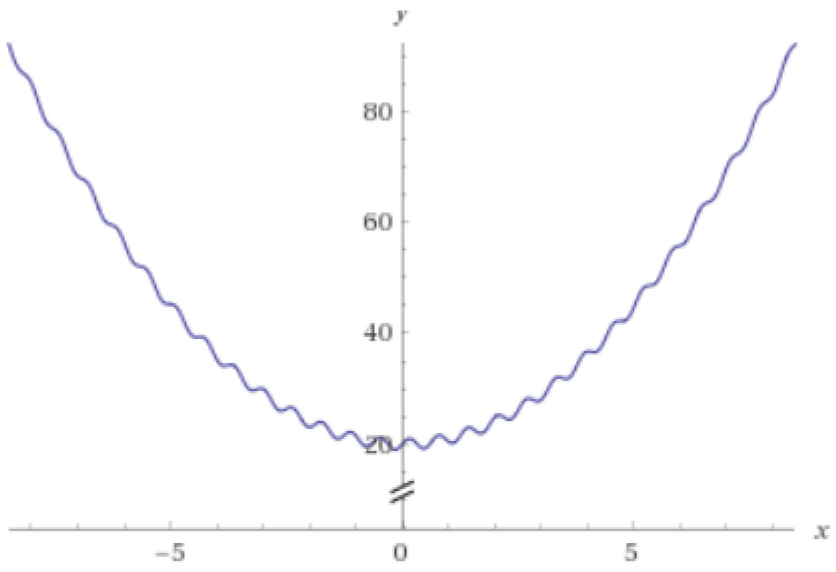


Figure 2.7: function F(x) =?? Source: own creation using Wolframalpha.com

This is how the boosting model will fit the data:

1. A rough approximation will be done, for example f(x) = 60. The model would look like this:

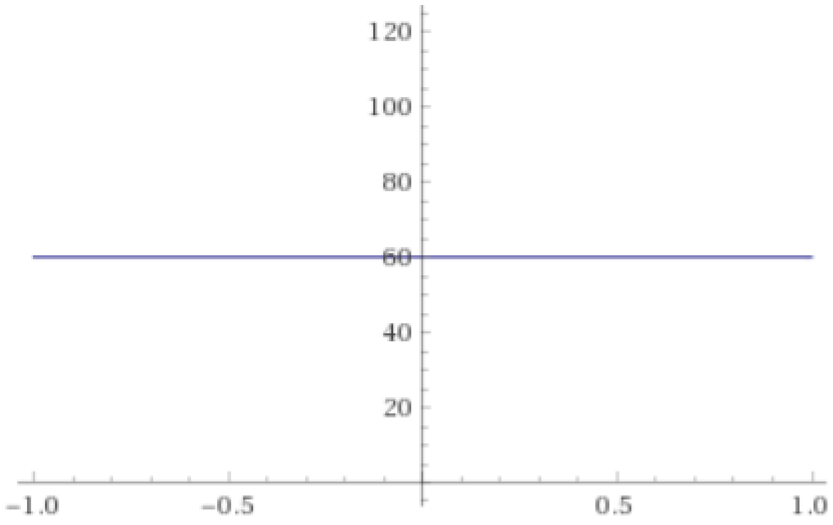


Figure 2.8: f(x) = 60 Source: own creation using Wolframalpha.com

2. In the next step the model will try to improve the results taking into account the residuals (difference between real data and predicted result) of the previous iteration. For example, since the residuals will have a U shape the algorithm will include this in the model, for example adding x^2. The model would now be $f(x) = 60 + x^2$ and it would look like this:

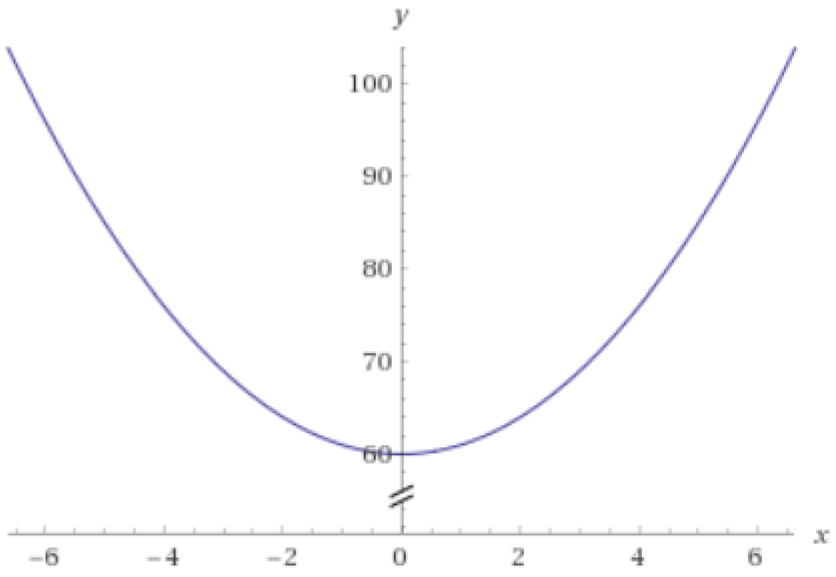


Figure 2.9: $f(x) = 60 + x^2$ Source: own creation using Wolframalpha.com

3. Here the model starts to look like the original data. In the next iteration the algorithm might identify the oscillatory nature of the data and decide to add a sinus function with specific frequency. The model would be $f(x) = 60 + x^2 + \sin(10x)$ and it would look like this:

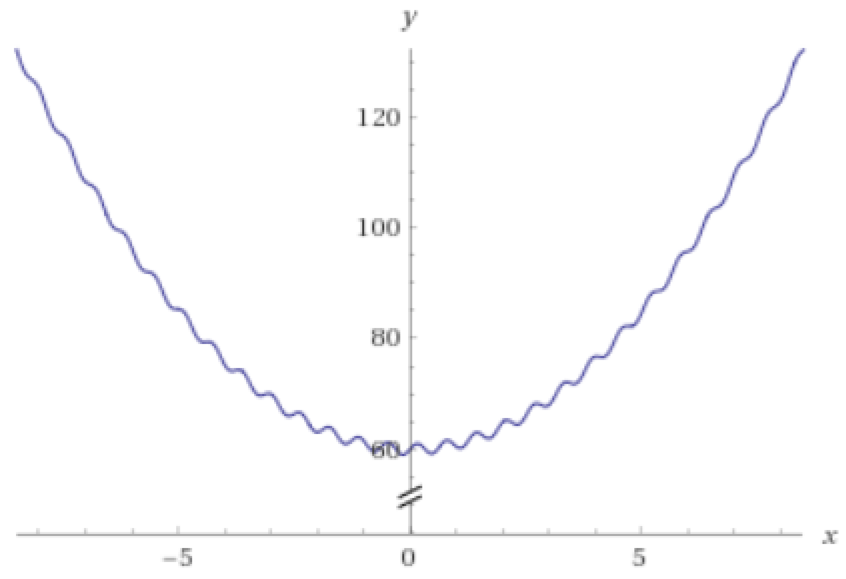


Figure 2.10: f(x) = 60 + x2 + sin(10x) Source: own creation using Wolframalpha.com

4. At this point the model might look really similar to the one in figure 2.7, but the residuals would show a constant offset of 40 in all points, due to the starting approximation of f(x)= 60 although the data cuts the y-axis at 20. The initial approximation of 60 in the first iteration was not accurate enough, and that is not a problem if the model is allowed to iterate enough to correct the residuals in the following iterations. Remember the model learns slowly, but learns. At this point the algorithm would try to correct this deviation and would subtract 40 to the constant value. After this the model would be f(x) = 60 - 40 + x^2 + sin(10x) and it would perfectly fit the data as displayed in figure 2.7.

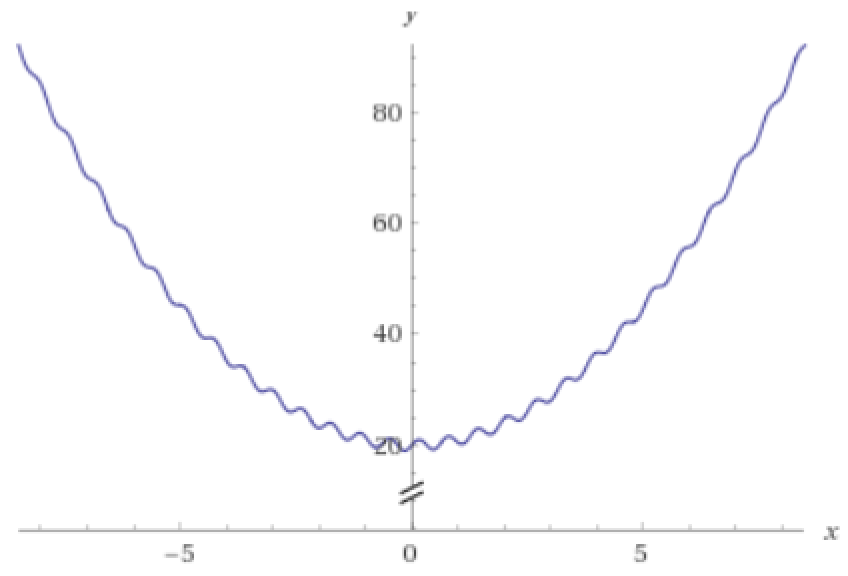


Figure 2.11: final model f(x)= 20 + x2 + sin(10x) Source: own creation using Wolframalpha.com

2.5.2 Why is it called GRADIENT boosting?

The method is called gradient boosting to explain the “tool” the model uses to improve at each step. To understand this it is necessary to introduce the concept of loss function. The loss function is an indicator the model uses to understand how far its predictions are from the real observations. Therefore, it measures the improvement on the fit at each step by computing the reduction in the loss function of the new model proposal. An example of a loss function is the quadratic one: $(y_i, f(x_i)) = \frac{1}{2}[y_i - f(x_i)]^2$. For each observation i the squared difference between the observation (y_i) and the prediction of the model ($f(x_i)$) is calculated (the ½ is only used so that when the function is derivated the 2 disappears).

When looking for the direction of model improvement at each iteration, the model uses the gradient of the loss function to discover the fastest direction to decrease the Loss function. It is possible to think of the gradient as a compass the model uses. In mathematical terms this is the partial derivative of the Loss function:

$$-\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

with

$$L(y_i, f(x_i)) = \frac{1}{2}[y_i - f(x_i)]^2$$

is

$$y_i - f(x_i) \text{ or equivalently } y_i - \hat{y}_i$$

So, the model is calculating the difference between the observed value and the predicted value of the previous iteration (m-1) to guide the creation of the current iteration (m), which has been introduced in section 2.5.1 as the residuals left by the previous iteration that the algorithm was trying to minimise. This indicates in which direction should the model move (positive or negative sign) and how far it is from the actual value (value of the difference).

This is an example of loss function for regression models but other loss functions can be used with different properties providing different model building characteristics. The loss function also depends on the type of problem at hand. If the data follows a classification model the loss function needs to be selected accordingly. Two common options are available in that case, an exponential loss function:

$$L(y_i, f(x_i)) = e^{-y_i f(x_i)}$$

or the binomial deviance [2]

$$L(y_i, f(x_i)) = \log(1 + e^{-2y_i f(x_i)})$$

The exponential loss function gives rise to the well-known AdaBoost.M1 algorithm but has the drawback of trying too hard to correct large misclassifications. The Binomial deviance mitigates this behaviour.

The derivative of the binomial deviance is [2]:

$$y_i - p(x_i)$$

where $p(x_i)$ is the predicted probability of the response being class 1 returned by the model per each observation i. This assumes a classification problem with only two responses {0, 1}.

2.5.3 Learning coefficient or shrinkage

At this point is also convenient to introduce the concept of learning coefficient (λ) or shrinkage parameter. This coefficient, between 0 and 1 multiplies the prediction created at each iteration of the model. This forces the model to approach the real data y_i but without doing it too fast. Recommended learning coefficients are usually between 0.01 and 0.001. The learning coefficient would be introduced like this:

$$\hat{F}^b(x) = \hat{f}^{b-1} + \lambda \hat{f}^b$$

where, at each iteration, the final model $\hat{F}^b$ consists of the model up to the previous iteration $\hat{f}^{b-1}$ plus the new model, in our case a tree, created at the current iteration $\hat{f}^b$ multiplied by the learning coefficient (or shrinkage) λ.

The rationale behind this is that if the model is allowed to approach the data without being slowed (λ = 1) it will almost perfectly fit the data really fast. The model would predict really good the training data but would perform poorly when faced with new data, in other words it would overfit the data.

This can be observed in figure 2.12 where the red square and the purple triangle show the test and training AUC (remember higher AUC means better adjustment) of a Gradient Boosting Machine with a quite high λ of 0.1. It is clear that the higher the number of trees allowed the higher the training AUC and the lower the test AUC becomes. On the other hand, if one looks at figure 2.12 to the blue circles and the green diamonds that show a model created with λ = 0.001 it will see that the best model is obtained with 5000 iteration but this model is quite better at predicting new data than the model which uses λ = 0.1.

The side effect of using this a small shrinkage parameter is that more iterations will be needed to reach the best possible model. This results in increased computational resources, but almost always this price is totally acceptable compared to the increased quality of fit.

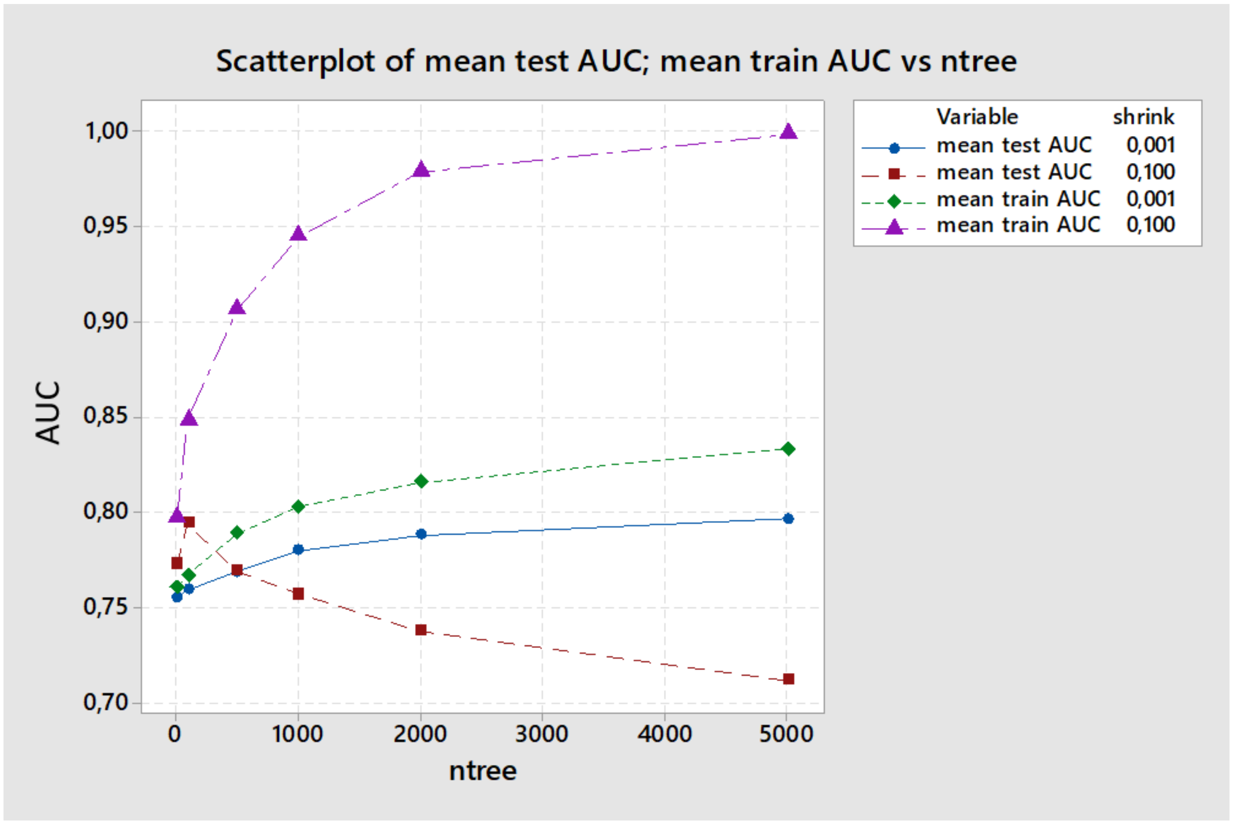


Figure 2.12: GBM models fitting the same data using λ = 0.1 and λ = 0.001. Source: own creation

It is important to consider also that the smaller the learning coefficient (λ closer to 0) the higher the number of iterations needed. Similar ratios of data fit will be obtained with λ = 0.01 and 100 iterations than with λ = 0.001 and 1000 iterations. This relation will be demonstrated empirically when the model obtained is analysed in section 5.1.1.

2.5.4 Gradient boosting using decision trees

In the previous sections the concept of gradient boosting has been explained without mentioning decision trees at any point. And that is totally correct as gradient boosting can be applied in several scenarios, one of which involves decision trees. But, since this project is focused in using decision tree based methods to fit a dataset, additional details will be explained here about the specifics of using decision trees to apply gradient boosting.

When creating the model several hyperparameters need to be fine-tuned, (hyperparameters are the coefficients selected by the user on the model before starting the model fit). The correct tuning of these parameters is really important in methods that tend to overfit the data like gradient boosting. These parameters are number of iterations and learning coefficient for gradient boosting. When decision trees are involved an additional parameter is available: tree depth.

Usually shallow trees are recommended (typically trees with one node, called stumps), as too deep trees tend to overfit the data. For example when the number of levels is higher than 9-10. This will be shown in chapter 5.1.1, during the analysis of the results applied to a real dataset.

But tree depth has also other implications. Tree depth is also referred to as interaction depth, as when only one level tree is allowed an additive model is obtained (the model is a sum of order 1 parameters, similar to a linear model). When two levels are allowed two variables can get involved in each tree and variable interaction is possible. This will result in interactions of order two at maximum. This relation can continue to higher level interactions. Interactions are required if it is suspected that the population from which the data is obtained may have them.

3 Dataset

The aim of this project is to apply and understand the presented theory: understand how decision trees work, if different types of decision trees models behave differently and how they perform compared to other statistical methods. The best way to do this would be using a real scenario with real data, instead of simulated data. This has been possible thanks to the department of Applied Statistics from the School of Industrial Engineering of Barcelona (ETSEIB) at the Universitat Politècnica de Catalunya, who has provided the study “New customers’ classifier” done by the student Pau-Ramon Casas Cachinero [1].

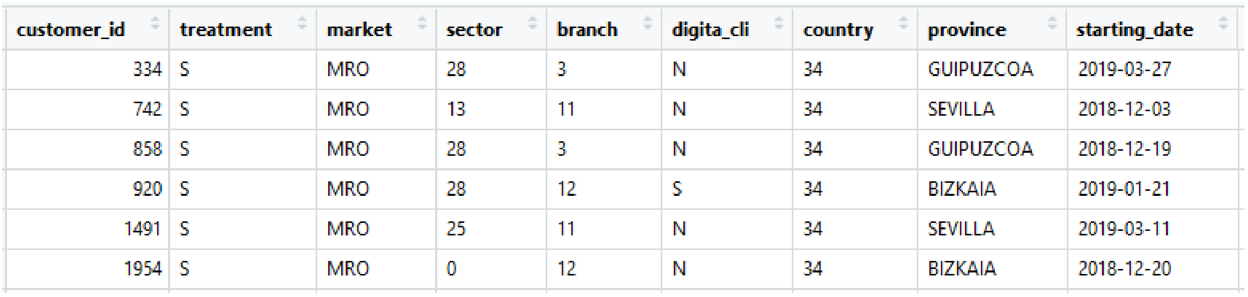
The study of P. Casas creates a model to identify good customers from an industrial wholesaler based on their few initial purchases. The challenging part of the project is the need to assess whether a customer who starts transacting with the company will become a good or bad one in the future based on his/her initial purchase features. Specifically, the model built needs to predict whether a customer is “good” (commercially interesting) or “bad” (not commercially interesting) **based only on the information from the first, second or third purchases of that customer.**

This study used a logistic regression model to make these predictions. Therefore, using the same dataset allows to use real-case observations plus being able to compare the current decision tree models with another statistical method as logistic regression. To allow the reader to understand the used data without needing to consult external documents the description, treatment and analysis of the dataset done by P. Casas is summarized in this section with minor variations.

3.1 Initial dataset: Customer Master

The original datasets contain nearly 1 million line items from purchases done by around 15,000 customers of the company. From these initial data about 70,000-line items and 6,000 customers will be used. These data correspond to customers who have started their relationship with the company in the observation period going from May the 2nd, 2016 to March the 29th, 2019, hence all customers with starting date before the 2nd of May are not included in the used dataset.

This sections explains briefly the variables available in the used dataset:



customer_id	treatment	market	sector	branch	digita_cli	country	province	starting_date
334	S	MRO	28	3	N	34	GUIPUZCOA	2019-03-27
742	S	MRO	13	11	N	34	SEVILLA	2018-12-03
858	S	MRO	28	3	N	34	GUIPUZCOA	2018-12-19
920	S	MRO	28	12	S	34	BIZKAIA	2019-01-21
1491	S	MRO	25	11	N	34	SEVILLA	2019-03-11
1954	S	MRO	0	12	N	34	BIZKAIA	2018-12-20

Table 3.1: master customers dataset structure

Table 3.1 shows the structure of masterCustomers dataset listing all available customers. Its field contents are:

- **Customer_id:** Customer identification number
- **Treatment**: It can take two values: Gold or Silver. These two values are given by the company depending on the customer annual revenue and its reputation. That variable is assigned by the company in hindsight and it is not available when a customer places its first purchases.
- **Market:** It can take three levels: MRO, OEM or SI. It depends on the customer's type:
    - MRO: Maintenance and Repair.
    - OEM: Original Equipment Manufacturing.
    - SI: Industrial Systems.
- **Sector:** It can take 29 different levels depending on the client activity sector.
- **Branch:** Provides information about the company's branch with which the customer contacted on its first purchase.
- **Digita_cli:** It can take two values: Yes or No. Yes customers are customers who made its registration using digital methods, basically the company's webpage.
- **Country:** Country in which the customer is located.
- **Province:** Spanish province in which the customer is located.
- **Starting_date:** Date in which the company made the customer's registration.

3.2 Exploratory analysis

In the master Customers dataset there are 5,862 lines which corresponds to the same amount of different customers. Although the details will not be explained here, the original dataset was cleared to ensure the data used is consistent. In this section an exploratory analysis is presented to have a background knowledge about the data used.

Spain hosts 98.7% of all customers. Figures 3.1.a and 3.1.b show the **customers per branch** from where they are being served and **province** where they are located respectively. 36% of them are assigned to branch 1, located in Barcelona. There are 4 other big branches that have 57% of the customers. These are Sevilla, Donostia, Bilbao and Madrid. The six remaining ones only represents 7% of the total.

With regards to the **province** variable the distribution is very similar. 33% of the customers are located in Barcelona and 49% in Sevilla, Guipúzcoa, Bizkaia and Madrid. This sounds reasonable as these are the areas with higher population density of Spain. It is also clear that customer location and branch location percentages match. It seems that most of the customers are assigned to branches in their location. With the creation of the variable *same.zone* (in chapter 3.4 Predictor variables) it will be possible to test that 85.5% of customers use the branch in their own location.

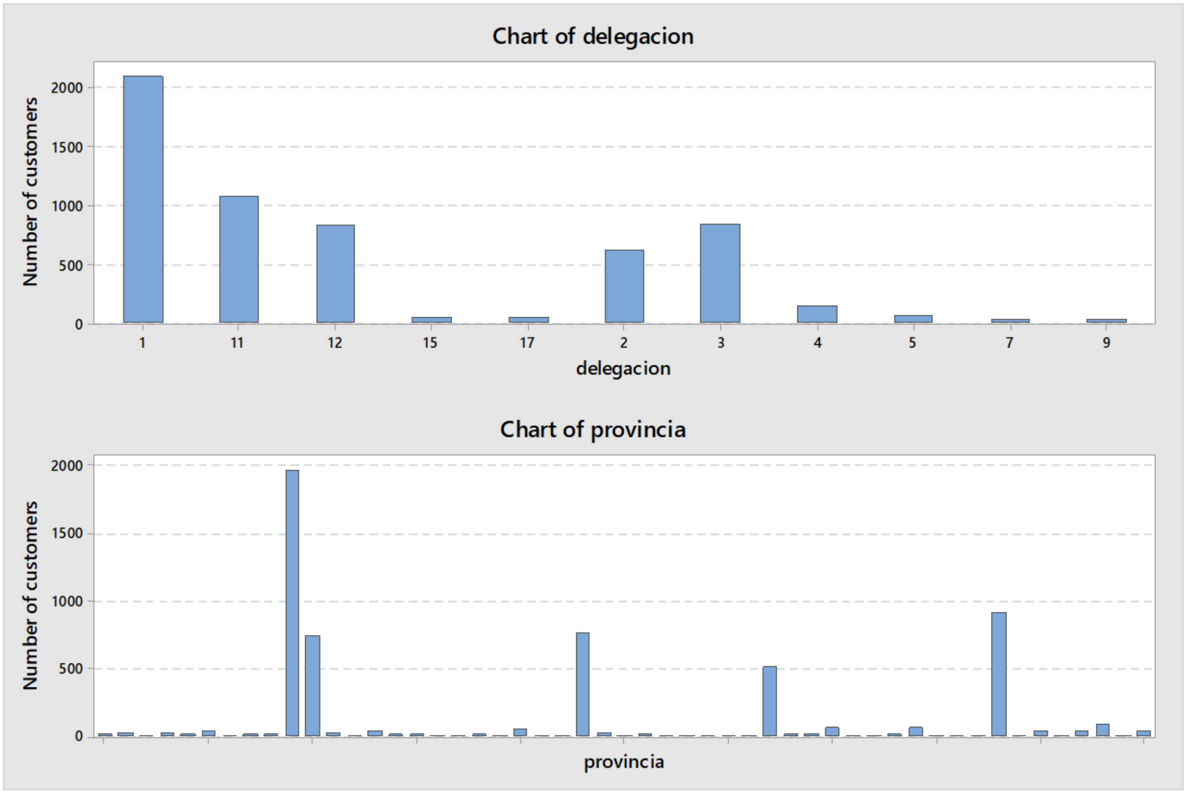


Figure 3.1: a) Company's branch bar chart, b) Customer's province bar chart. Source: own creation

Sector variable has been analysed and 41.5% and 34.4% observations belong to sectors 0 and 28 respectively. An issue has been found here as sector 28 is labelled as "Others" and sector 0 is equivalent to NA (not assigned). This means 75.9% of the observations fall into the not identified category. This would not be a problem if the data were to be missing at random. Missing at random means that NA observations do not have any information. However, if a table containing the distribution of good/bad customers among sectors is created, it is possible to see that known sectors (different from 0 or 28) tend to have considerably higher ratio of good customers (see table 3.2).

For this reason it has been decided not to use the variable sector in the model, as this data is not trustful. The hypothesis behind this behaviour is that only interesting customers (for example with a certain time in the company or with high expenditure) drawn enough attention to have certain data filled into their profile, like the sector. Unfortunately it has not been possible to test this hypothesis with the provider of the data.

Sector	Bad customers (%)	Good customers (%)	Sum
0	93,80	6,20	100,00
1	77,78	22,22	100,00
10	75,56	24,44	100,00
11	64,29	35,71	100,00
12	66,67	33,33	100,00
13	69,86	30,14	100,00
14	82,61	17,39	100,00
15	68,63	31,37	100,00
16	68,00	32,00	100,00
17	82,76	17,24	100,00
18	73,12	26,88	100,00
19	43,75	56,25	100,00
2	68,42	31,58	100,00
20	70,97	29,03	100,00
21	78,95	21,05	100,00
22	65,85	34,15	100,00
23	66,67	33,33	100,00
24	83,33	16,67	100,00
25	86,09	13,91	100,00
26	47,62	52,38	100,00
27	76,92	23,08	100,00
28	93,16	6,84	100,00
29	56,25	43,75	100,00
3	76,24	23,76	100,00
4	80,95	19,05	100,00
5	62,86	37,14	100,00
6	89,47	10,53	100,00
7	85,71	14,29	100,00
8	72,33	27,67	100,00
9	81,82	18,18	100,00
All	88,60	11,40	100,00

Table 3.2: distribution of good/bad customer among sector

Figure 3.2.a shows distribution among **markets**. MRO customers represent the 85.6% of the total. OEM and SI customers are 7.7% and 6.6% respectively. 94% of customers are not **“digital”**, as displayed in figure 3.2.b.

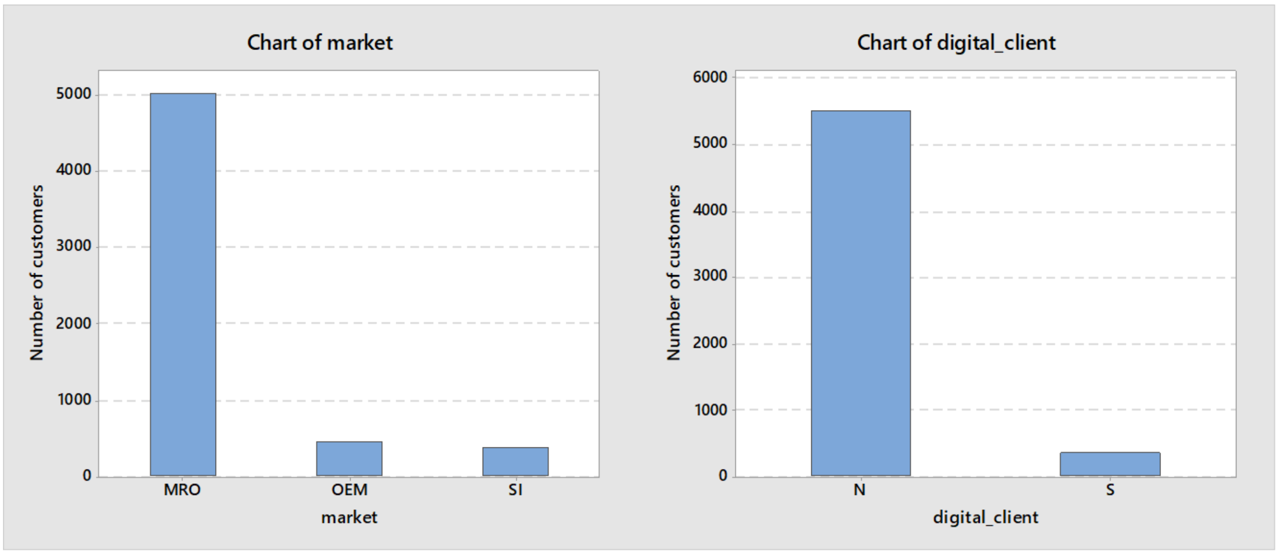


Figure 3.2: a) Market bar chart, b) Digital client bar chart. Source: own creation

After doing master customers exploratory analysis it can be said that a standard customer is a customer which is located in Spain, is not digital and belongs to market MRO. There are not a clear major group in province and branch variables.

3.3 Response variable: good customer

3.3.1 Definition of a good customer

As stated before, the main objective of this dataset is to determine whether a new customer will be “good” or “bad” (commercially) given the features of his first purchases with the company. For this classification to work it is key to clearly classify already existing customers as good or bad from the company’s point of view.

Several meetings were done by P. Casas with the marketing teams of the company and two parameters were defined to determine a good customer: 1) whether the customer is “dead” or “alive” and 2) the monetary value for to the company. A good customer is one who is “alive” and has a monetary value above average.

3.3.2 Alive or dead customers

To study which features determine whether a new customer is going to be good, the behaviour of the ones from which recorded data during the observational period is available will be analysed.

The first step is to decide when a customer is “dead” or “alive”, this has been called its activity status.

A “dead” customer is one which has ended his relation with the company and will not buy from them again. On the other hand, an “alive” customer is a customer who is active and is likely to place orders in the near future.

The company’s relationship with the customers is known as non-contractual. Customers do not formally communicate to the company when they plan to stop transacting with it. On the contrary, settings like insurance’s, banks or phone companies are known as contractual settings as their relationship with customers is based on contracts. In this examples the end of a customer relationship is known for sure. In non-contractual settings, there is never a 100% security that the customer is done operating with the company. There will always be the possibility that he/she is just going through a long relapse in his commercial activity. This makes classification of dead or alive customers harder.

To approach this issue, customers have been differentiated in three groups depending on the amount of purchases done. One group includes customers born in the observation period and that have made only 1 purchase, the second customers that have done 2 or 3 purchases, and finally, customers with 4 or more purchases.

Two variables have also been created to be used in the definition of a “dead” or “alive” customer:

- **Global behaviour.** This variable compares the individual behaviour of the customer versus the behaviour of the rest of the customers. It can take 2 values: TRUE or FALSE. It is TRUE if the customer’s recency[1] is smaller than the 80% quantile of the distribution of all customers maximum time between purchases (312 days as will be seen below). The period of time within 312 days prior to the last day of the observation period is named “Recency zone”. This term is going to be used in the monetary value explanation.
- **Individual behaviour.** This variable is an indicator of the behaviour of the customer individually. It is a binary variable. It is TRUE if the recency of the customer is smaller than its own maximum time between purchases.

Depending to which group the customer belongs it is defined whether it is dead or alive using the following criteria:

1. **Customers with one purchase.** A one purchase customer can never be considered as an “alive” customer, they can be either “dead” or uncertain about their status (NA customer). They will be considered dead if their “Global behaviour” variable is false and NA otherwise. That is, they are dead if their first and only purchase was done before the “Recency zone” and NA if it was done within it.

1: Recency = Time between customer’s last purchase and the end of the observational period (29-03-2019).

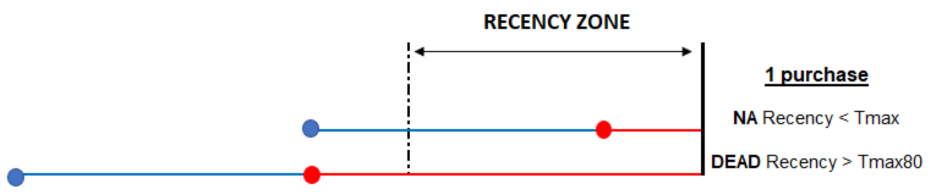


Figure 3.3: Representation of a dead and an alive one purchase customer. Source: [1]

2. **Customers with two or three purchases.** These are considered “alive” if their “Global behaviour” or “Individual behaviour” variables are TRUE. In any other cases they are considered “dead”.

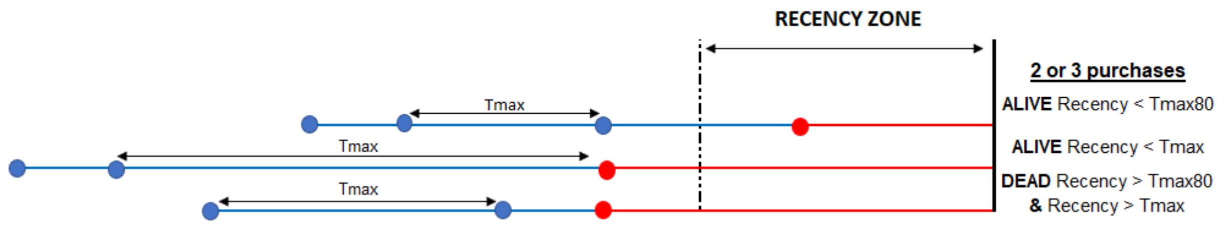


Figure 3.4: Representation of two alive and one dead two purchases customers. Source: [1]

3. **Customers with four or more purchases.** These customers are considered “alive” if the “Individual behaviour” variable is TRUE. Otherwise they are “dead”.

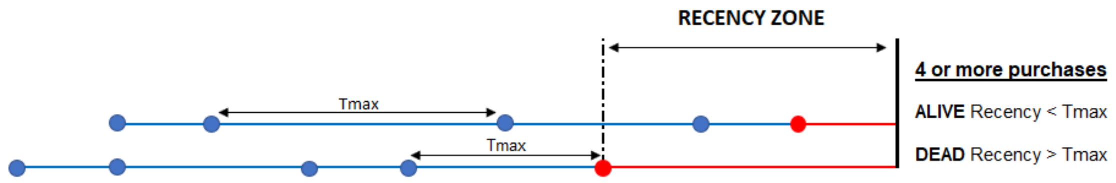


Figure 3.5: Representation of a dead and an alive three purchases customers. Source: [1]

The rationale behind this assignation is that the more information available about a customer the more it is possible to rely on their own purchase behaviour and it is less needed to rely on global behaviour data.

The **recency zone** is the time lapse between May 21st 2018 and March 29th 2019 as the 80% quantile of the maximum time between purchases for all customers is 312 days. It can be seen in Figure 3.6.a the distribution of maximum time between purchases. It can be observed that this variable follows a negative exponential law.

Figure 3.6.b shows a bar chart of the distribution of customers depending on their activity status. Of the total customers born in the observation period 3,308 are assumed dead, 1,392 alive and 1,398 it is not possible to say anything of them (NA’s). Dead customers are 54.3% of the total number while alive and NA customers represent nearly 23% each one.

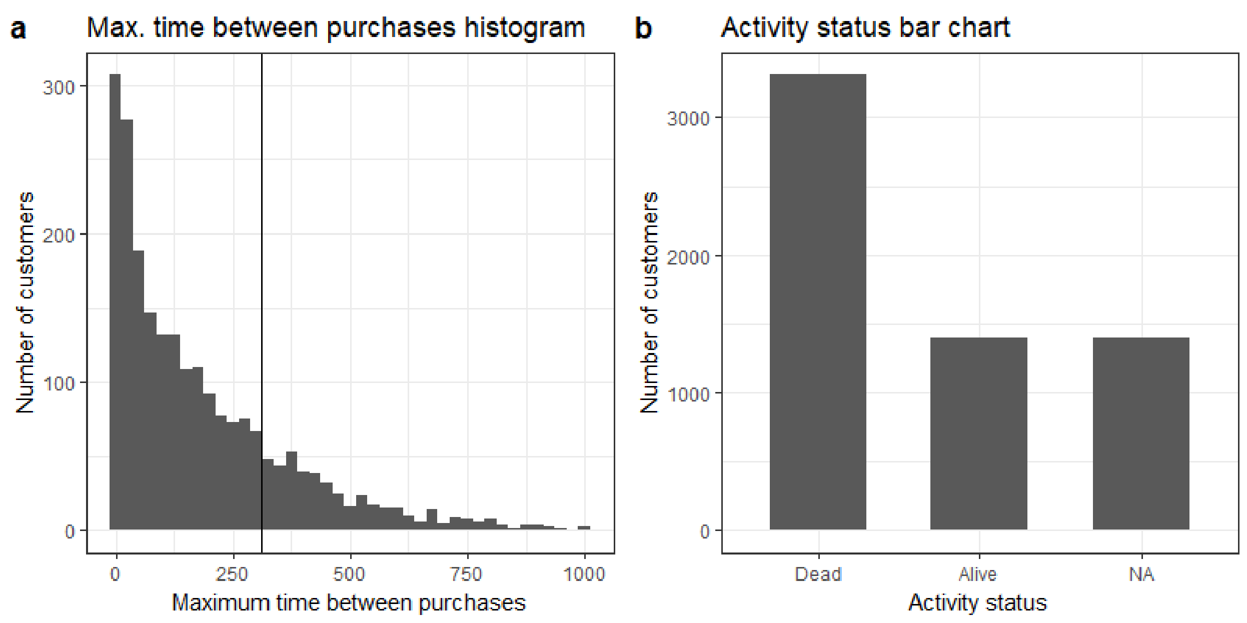


Figure 3.6: a) Maximum time between purchases histogram, b) Activity status bar chart. Source: [1]

3.3.3 Monetary value rate

Once the dead or alive status of a customer is assessed it is time to estimate the second key parameter: the monetary value. It has been defined monetary value as the total value spent by the customer by time unit. It is computed as follows:

$$Monetary\ value = \frac{totalvalue}{t.firstpurchase}$$

Totalvalue is the amount spent by the customer since its first purchase and up to the end of the observation period, minus its refunds if any. *t.fisrtpurchase* is the lapse of time between the first purchase and the last day of the observational period.

The higher the total value spent is, the higher the customer monetary value variable. The lower the time lapse in which this amount has been spent, the higher the customer monetary value. For example, a customer who spent 300 € in two purchases done in two months is better than one who spent the same amount in two purchases done during 4 months.

A risk with the proposed monetary value is that it could overestimate customers that were “born” close to the end of the observation period. In this case their purchase value would be divided by the small elapsed time. In order to solve this problem, customer have been divided in two groups depending on the period they were “born”. Then the monetary value has been re-defined depending on these groups.

1. Customers that have born inside the “recency zone”:

$$Monetary\ value = \frac{totalvalue}{T_{MAX}80\%}$$

   Where $T_{MAX}80\%$ is the time that corresponds to the 80% percentile of the maximum time between purchases distribution for all customers, in this case 312 days.

2. Customers that have born outside the recency zone:

$$Monetary\ value = \frac{totalvalue}{t.firstpurchase}$$

For the second group the original formula is used but for customers whose first purchase was done inside the recency zone their *totalvalue* is divided by the overall population T_{MAX}80%. It is shown in Figure 3.7.

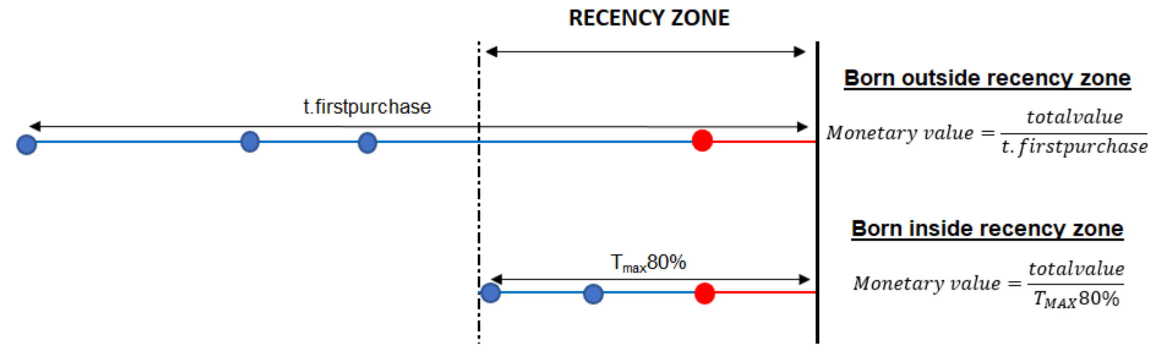

RECENCY ZONE
t.firstpurchase
Born outside recency zone
$Monetary\ value = \frac{totalvalue}{t.firstpurchase}$
Born inside recency zone
$T_{max}80\%$
$Monetary\ value = \frac{totalvalue}{T_{MAX}80\%}$

Figure 3.7: Representation of the two groups of monetary values. Source: [1]

Thanks to this change a potential overestimation of customer monetary value due a “short” active time is expected to be reduce.

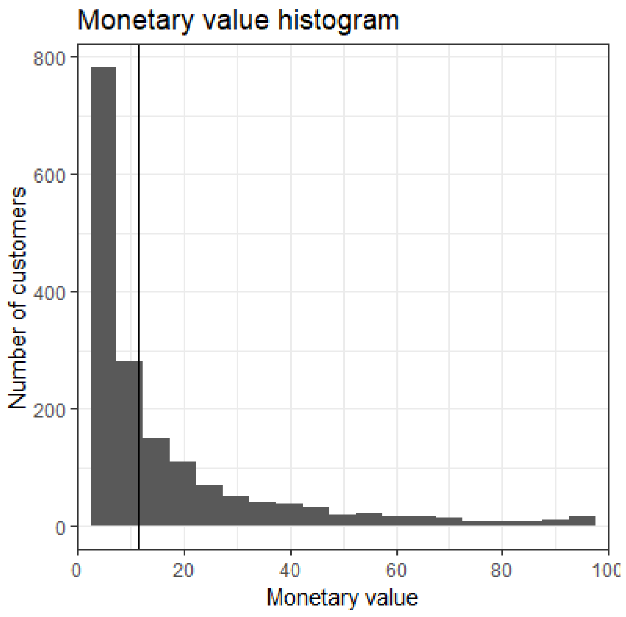


Figure 3.8: Monetary value histogram. Monetary value limited between 0 and 100 €/month. Source: [1]

Once the monetary value variable is defined, it is calculated for each customer. Figure 3.8 shows the distribution of the variable and its 60 % percentile for customers with more than one purchase. This percentile will be the threshold for good monetary value customers. The distribution is positively skewed with a long tail for high values. Median monetary value is 7.63 €/month. The 60% percentile of the monetary value distribution is used as a threshold for a good monetary value customer and it corresponds to 11.50 €/month or 138 €/year.

3.3.4 Good customers

Once both parameters are defined and calculated for each customer in the dataset there are four main possible customer groupings. These are represented in Figure 3.9. Based on the previous criteria, a good customer is a customer that belongs to the 4th group. It is alive and its monetary value is above 138 €/year (the 60% percentile of the monetary value distribution for all the more than one purchase customers in the dataset).

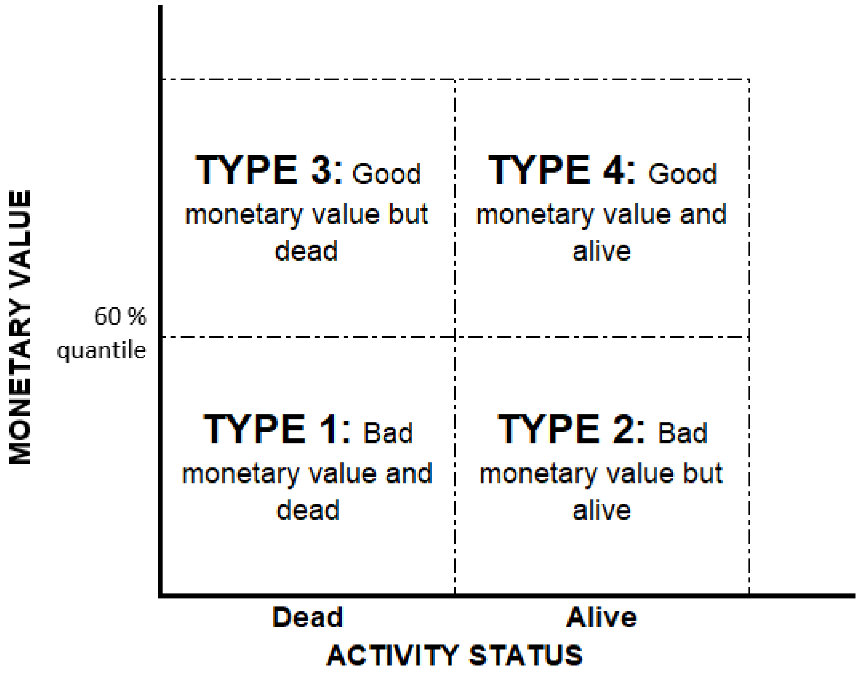

MONETARY VALUE
60 % quantile
TYPE 3: Good monetary value but dead
TYPE 4: Good monetary value and alive
TYPE 1: Bad monetary value and dead
TYPE 2: Bad monetary value but alive
Dead
Alive
ACTIVITY STATUS

Figure 3.9: Main types of customers' representation. Source: [1]

For customers with NA active status two groups can also be defined depending on their monetary value. If their monetary value is below 138 €/year they are classified as bad customers. If their monetary value is higher than 138 €/year they cannot classified as good customers given their NA active status so they are classified as NA customers. These customers are removed from the dataset.

Figure 3.10 and Table 3.3 show the distribution of customers in the above mentioned groups and their total number.

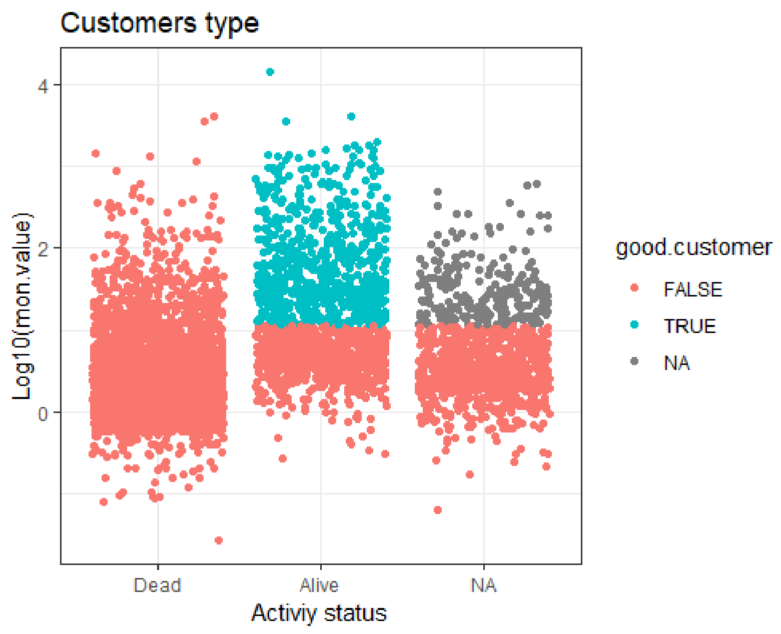


Figure 3.10: Types of customers' representation. Source: [1]

		MONEATARY VALUE	
		FALSE	TRUE
ACTIVITY STATUS	DEAD	2914	394
	ALIVE	722	**670**
	NA	1170	228

Table 3.3: Type of customers' table

1,292 (21.2%) of the total customers have good monetary value. Of these good monetary value customers 670 of them (52%) are alive, and the remaining 622 (48%) are dead or not classified. The 670 alive customers with good monetary value represent 11% of all the customers in the dataset and are the good customers used in the models.

Customers that are alive have significantly better monetary value as can be seen in Figure 3.11. Monetary value median for alive customers is 10.53 €/month and for dead ones is 1.75 €//month.

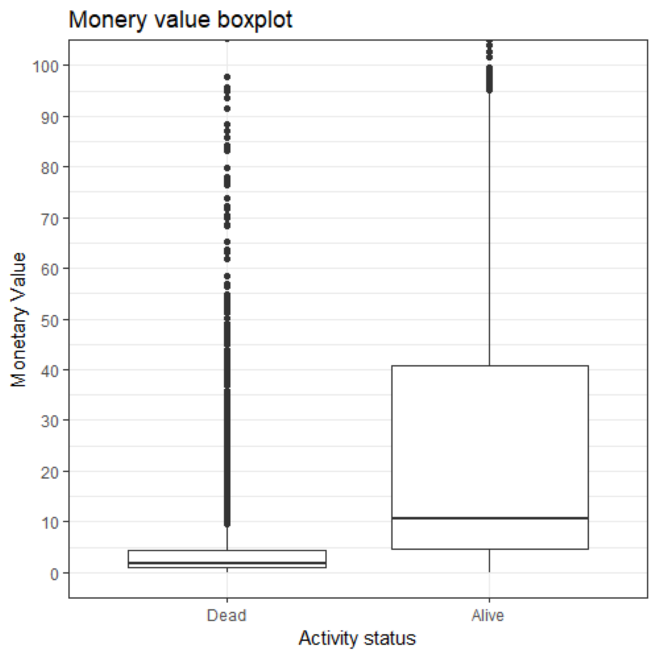


Figure 3.11: Boxplot of monetary value for alive and dead customers. Monetary value limited between 0 and 100 €/month. Source: [1]

3.4 Predictor variables

Predictor variables are a fundamental part of the model construction process. Predictors are used to obtain the models response, in this case determine whether a customer will be good or bad. Predictors are also named X's while Response is named Y. In this section, the X variables created and used to fit the model are going to be described and how they are calculated is going to be explained.

Some variables included in the original client dataset can be directly used as an X variable but most of them come from transformations of the original ones. All of the new variables created were chosen before the start of the model construction process. At that point, it was not known whether they would become significant. Their significance will be checked in future stages.

These variables have been classified in two groups: Purchasing behaviour variables and Demographic variables.

3.4.1 Purchasing behaviour variables

Purchasing behaviour variables are created based on the items purchased information. They give information about each client purchasing behaviour.

All of the purchasing behaviour variables except the month and the season of the first purchase will be different for the three models to be constructed (model using the first purchase only, the two first purchases or the first three purchases). Each one of them is based in a different number of transactions and therefore in different amounts. In the following explanation it is shown separated by // the number purchases entered in each of the three models.

The variables are:

- **Total value:** Numeric Variable. It is the total amount spent by the customer in the 1st// 1st and 2nd //1st, 2nd and 3rd purchases minus its refunds if any.
- **Total Entropy:** Numeric variable. It measures the variability of each customers' purchases. Entropy can take values between 0 and 1, been 0 null variability and 1 maximum variability. This model's entropy is based on families (groups of products) purchased and amount spent in each family. In the model, entropy is used for the 1st //1st and 2nd // 1st, 2nd and 3rd purchase. Entropy is calculated using the following formula.

  $$Entropy = \sum_{i=1}^{n} p_i \cdot log_n(p_i)$$

  Where p is the total amount spent in each family divided by the total amount of the purchase. n is the number of families in the purchase.

- **Total lines:** Numeric variable. It indicates how many lines or different products a customer bought in its 1st// 1st and 2nd // 1st, 2nd and 3rd purchases.

- **Total families:** Numeric variable. It is the number of different families (groups of products) included in the 1st// 1st and 2nd // 1st, 2nd and 3rd purchases.
- **First purchase month:** Categorical variable. It shows the month in which the first purchase was placed.
- **First purchase season:** Categorical variable. It shows the season in which the first purchase was placed.
- **Any disc:** Dichotomous variable. It is TRUE if there was any discount in the 1st // 1st and 2nd// 1st, 2nd and 3rd purchases.

The following purchasing variables are only available in the two and three purchases models:

- **Value's increase:** Numeric variable. It is the delta in the value spent between the 2nd and 1st // 3rd and 1st purchase
- **Lines' increase:** Numeric variable. It is the delta in lines purchased between the 2nd and 1st // 3rd and 1st purchase
- **Families' increase:** Numeric variable. It is the delta in families purchased between 2nd and 1st // 3rd and 1st purchase
- **Entropy's increase:** Numeric variable. It is the delta in the entropy between the 2nd and 1st // 3rd and 1st purchase.
- **Any refund:** Logical variable. It is TRUE if the customers have made any refund between the 2nd and 1st// 3rd and 1st purchase.

3.4.2 Demographic variables

Demographic variables are based on the master Customers dataset. They do not give information about purchase behaviour but about the profile of the customer. The first two of these demographic variables have been computed based on the information in the master Customer dataset. The last two of them are directly extracted from the dataset.

- **Same zone:** Dichotomous variable. It is TRUE if the location of the customer is the same than the company's branch assigned to the customer. It is FALSE otherwise. 85.5% of customers use the branch in their own location.
- **Dist:** Numeric variable. It is the distance between the company's branch and customer's province in kilometres.
- **Market:** Categorical variable. It has 3 values "MRO", "OEM" or "SI".
- **Digital client:** Dichotomous variable. It is TRUE if the customer made its registration using the company's webpage. It is FALSE in any other case.

Unlike most of the purchasing behaviour variables, demographic variables will not vary between the models using one, two or three purchases, as they are constant per customer.

Looking at the overall picture of the predictor variables, it can be seen that the majority of variables have been created from the dataset and not directly extracted from them. Only two of the 16 variables were directly available in the original dataset: market and digital client variable.

4 Building the models

The data described in the previous sections has made it possible to experiment with the different models based on decision trees explained in section 2, Theoretical Framework, using real data. In this section it is going to be explained how these models have been created, which software has been used, what data has been extracted to evaluate how the models behave in different scenarios and which statistical tools have been used to analyse the data, including the indicators to compare among the different models discussed in this project.

4.1 Model creation

The software used to arrange the data and create all the models has been the computer language R. Specifically the R 3.6.1 is the version of the language employed. The interface used has been Rstudio.

R is an open-source language for statistical learning, data mining and data analytics widely used both in academic and professional fields. This computer language is able to manage large amounts of data and state-of-the-art statistical techniques are being used and developed with it. One of its strengths, due to its open source nature, is the amount of content and functionalities developed by the users, included in what is called R packages (R packages can be either native or user developed). This high activity makes that it is almost always possible to use any wanted technique as it has already been developed by someone. Together with the availability of information in internet forums like Stackoverflow.com and others makes it possible to easily learn how to use the language and how to solve any problem that might appear.

R-studio has been the interface used as R IDE (Integrated Development Environment). R-studio is also open source and helps to improve the visualization of data when programming or analysing. It makes this possible thanks to different panels that show in a single window the different sections necessary for programming.

4.1.1 Package for bagging and random forests

The package used to create random forests models in this project has been the **randomForests package** by Andy Liaw (andy_liaw@merck.com) and Matthew Wiener (matthew_wiener@merck.com) based on Brieman's *Random Forests* article (Breiman, L. (2001). *Random Forests*. Machine Learning 45, 5–32), referenced in additional bibliography.

To create the bagging models this same package has been used as it is possible to tune the number of variables to use at each split. Therefore if this parameter is equal to the number of predictor variables of the data a bagging model will be created.

The main function of the package is the *randomForests* function which takes the following parameters:

- Formula: as the standard method in R to define the response variable Y and the predictor variables X.
- Data: to define the dataset to use.
- Ntree: defines the number of trees that the model is allowed to grow.
- Mtry: Number of variables randomly sampled as candidates at each split. This variable is defaulted to square root of the number of predictors but it has been tuned for this project. To create a bagging model this parameter has been set to the total amount of predictor variables.
- Nodesize: Minimum size of terminal nodes or equivalently minimum number of observations in each terminal node. Setting this number larger causes smaller trees to be grown, thus taking less time to create the model. It is defaulted to 1 but other numbers have been used.
- Importance: binary variable, if set to TRUE the importance of predictors is assessed. This feature has only been used (importance = TRUE) when creating the final model as it increases the calculation time.

The R function offers the option others parameters but they have not been explained here as either were performing functions done separately, like cross validation, or were not influencing the results.

4.1.2 Package for gradient boosting method

The package used to create gradient boosting models in this project has been the **gbm package** by Greg Ridgeway (gregridgeway@gmail.com). The main function of the package is the *gbm* function which takes the following parameters:

- Formula: as the standard method in R to define the response variable Y and the predictor variables X.
- Data: to define the dataset to use.
- Distribution: the distribution the model will assume the data has. If nothing is defined the model will try to guess. If the response is binary, Bernoulli is assumed. If the response is continuous Gaussian distribution is assumed. Other distributions are also available. For this project the distribution used has been Bernoulli.
- N.trees: defines the number of trees that the model is allowed to grow. It is defaulted to 100 but other values will be used.
- Shrinkage: shrinkage parameter or learning rate. Parameter between 0 and 1. The smaller the value of this parameter the slower the model will learn, thus more iterations (or trees) will be needed. It is recommended to use between 0.1 and 0.001. It is defaulted to 0.1 but other values will be assessed.
- Interaction.depth: Integer specifying the maximum depth of each tree. As explained in the theoretical framework this value also defines the maximum level of interactions allowed in the model. It is defaulted to 1 but other values will also be used.

- Bag.fraction: the fraction of the training set observations randomly selected to propose the next tree in the expansion. Parameter between 0 and 1. It introduces randomness in the model if <1 and it is defaulted to 0.5. In this project it has been set always to 1 as it showed better results when used with the dataset used.

By fixing the bag.fraction parameter the number of parameters to change in GBM models is set to 3 (Ntrees, shrinkage and interaction depth) making the number combinations reasonable to manage, as the graphs and analysis to be considered grows exponentially with the number of parameters to change.

The R package offers others variables to use but they have not been explained here as either were performing functions done separately, like cross validation, or were not influencing the results.

4.1.3 Other packages used

On top of the main R packages necessary to fit the models other support packages have been used. These are explained below:

- **pROC package** has been used to calculate ROC curves and the area under the ROC curve (Area Under the Curve or AUC) in order to compare models between them.
- **Caret package** has been used for one specific function called createFolds. This function facilitates the sub-sampling of the dataset to easily perform K-fold cross validation by dividing the dataset in k equal and random subsets.
- **Openxlsx package** has allowed to export the results into an Excel spreadsheet to easily work with the obtained data.

4.2 Obtain the results

Once all the tools needed have been set, it is the time to run the software to create the models necessary to answer the questions presented. At this point it is necessary to test the different models, bagging, random forests and gradient boosting using different parameters to understand how they behave.

This is because the objective is multiple here. On one hand the project seeks to obtain the optimum model so that it can be compared to the results obtained using logistic regression. On the other hand extreme values are wanted to observe the characteristics described in the theory (for example at what point gradient boosting overfits the data). It is thought that extreme scenarios might be useful to compare and understand the behaviour of the models and also what are the key point a user needs to take care of in order to obtain a model with good quality.

After understanding how the needed R packages work, the author has created an iterative procedure to test multiple combinations of parameters in only one code run. This code has been created for the different models of one, two and three purchases and the two different packages of R, bagging plus random forests running together and gradient boosting machine.

In the case of **gradient boosting** the code creates all possible combinations with the following parameters:

- **Number of trees** to be grown (called ntree in the code and graphics): {10, 100, 500, 1000, 2000, 5000}
- Learning rate or **shrinkage parameter** (called shrink in the code and graphics): {0.1, 0.01, 0.005, 0.001}
- **Interaction.depth**, integer specifying the maximum depth of each tree: {1, 3, 5, 7, 9, 15}

When all possible combinations are taken into account it results in 144 combinations for each type of model with one, two or three purchases. All results are available in Annex 4.

In the case of **bagging/random forests** the code creates all possible combinations with the following parameters:

- **Number of trees** to be grown (called ntree in the code and graphics): {10, 100, 500, 1000, 2000, 5000}
- Mtry. **Number of variables randomly sampled** as candidates at each split. When mtry=(all variables) the model created is a bagging model. This value depends on the model used as the number of variables is different:
  - 1 purchase: {1, 2, 3, 4, 5, 7, 9, 12}
  - 1 and 2 purchases: {1, 3, 5, 6, 10, 13, 17, 20, 23, 27}
  - 1, 2 and 3 purchases: {1, 3, 5, 6, 8, 10, 15, 20, 25, 30}
- **Nodesize**. Minimum size of terminal nodes:
  - 1 purchase: {1, 5, 10, 20}
  - 2 and 3 purchases: {1, 5, 10, 100}

When all possible combinations are taken into account it results in 192 combinations for the one purchase model, and 240 combinations for the two and three purchases models. All results are available in Annex 3.

On top of the different configurations of parameters explained, in every combination it has been used a 10-fold cross validation to ensure the results obtained are representative and they are not biased by the random numbers used. For example when creating the Gradient boosting model with 1000 trees, 0.01 shrinkage parameter and 5 maximum tree depth the dataset has been split in 10 random samples. Then the model has been trained using samples 1 to 9 altogether and sample 10 has been left out and used only to test the model. This has been repeated then with samples 1 to 8 plus sample 10 to fit the model and sample 9 as test, and repeated until the 10 samples have been used as test. This means that for every combination of parameters 10 different models have been created, having 10 values of AUC from test data and 10 AUC's from training data. Then the mean and standard deviation of this AUC's have been calculated, graphically displayed and analysed (all available in Annexes 1, 2, 3 and 4). Summing the models with one, two and three purchases, this project has created 4320 GBM models and 6720 random forests models.

4.3 Preparing the analysis

To obtain representative information, the data analysed has been directly the one obtained after grouping the results of all 10 models obtained after the 10-fold cross validation, these are the 10 models created per each combination of parameters. Hence, for each combination four indicators are available: the mean and standard deviation of AUC values obtained by using the test data and the mean and standard deviation of the AUC using the training data. The complete information is available in Annexes 3 and 4. This allows to assess the quality of fit using the test information and also to evaluate if the model is overfitting the data using the training information.

To analyse all this data the Minitab software has been used. After displaying some descriptive statistics to see basic information, contour plots have been used to understand the behaviour of the model when the parameters change. Logarithmic transformations have been applied to number of trees variable (GBM and RF) and shrinkage (GBM) to avoid concentration of data in one small area of the graphic. These graphics are shown and analysed in the next section "Analysis of the results" and all the graphics are available in Annexes 1 and 2.

5 Analysis of the results

After understanding the theory and the rationale behind the different decision tree models, obtaining a real dataset to apply this theory to real-life data and creating and processing different models, now it comes the moment to analyse this data and let the project provide conclusions.

This section will be divided in three chapters, answering the three questions presented at the beginning of the thesis.

- Which are the key parameters that influence the construction of these tree-based models and how do they react to them? (Also understanding the theory behind)
- How the three decision tree models behave? Comparison between bagging, random forests and gradient boosting.
- How does the best decision tree model behave compared to the logistic regression model created by P. Casas in the previous project using the same data?

The next pages answer this questions.

5.1 Sensitivity analysis of parameters

5.1.1 Gradient Boosting

The best performing models using gradient boosting are:

- **1 purchase model: Test AUC= 0.79658** (0.015); Train AUC= 0.83344. ntree= 5000 (log=3.7); shrink= 0,001 (log=-3); int.depth= 3
- **2 purchases model: Test AUC= 0.81700** (0.024); Train AUC= 0.83686. ntree= 1000 (log=3); shrink= 0.005 (log=-2.30); int.depth= 1

- **3 purchases model: Test AUC= 0.79029** (0.044); Train AUC= 0.82474. ntree= 500 (log=2.7); shrink= 0.01 (log=-2); int.depth= 1

Gradient boosting tends to overfit the data, for this reason is really important to adjust properly the parameters to avoid losing quality of fit. However, unlike other statistical methods that require a step-by-step building of the model, gradient boosting is quite fast to construct. Either way, in this project an extensive sensitivity analysis has been conducted to understand how these models behave in a wide range of conditions.

In a more day to day work, in order to fit only a model for prediction (the typical case) this process can be shortened significantly. One strong point of decision trees is that data does require only a minimum treatment (like response being 0-1 values or a factor/string class depending on the package), but no dummy variables or similar are needed. When this part is complete, good results can be obtained by doing several trials with a range of low interaction depths (between 1 and 5) while choosing a combination of number of trees and shrinkage that maximises the test AUC and keeping the training AUC under control. It is important to consider that the higher the test AUC, the higher the training AUC will naturally be. This process can be quite fast compared to other statistical methods like logistic regression, while not being limited by the number of categorical variables with large amount of classes or the presence of outliers.

With this being said, the next paragraphs show the main highlights of how the different parameters behave based on all the data obtained from the fit of all models using GBM.

1. <u>Interaction depth is a key parameter</u>

In all scenarios an interaction depth between 1 and 3 is providing the best results. Good results can also be obtained with interaction depths up to 7 as the test AUC is not dropping drastically, but only in some specific combination of parameters and always providing worse performance compared with 1-3 tree depths.

ntree	shrink	int.depth	mean test AUC	sd test AUC
5000	0,001	3	0,79658	0,01582
1000	0,005	3	0,79652	0,01584
500	0,01	3	0,79643	0,01614
5000	0,001	1	0,79544	0,01478
1000	0,005	1	0,79531	0,01496
500	0,01	1	0,79528	0,01501
100	0,1	3	0,79458	0,02076
100	0,1	1	0,79446	0,01411
1000	0,01	3	0,79441	0,01937
2000	0,005	1	0,79431	0,01409

Table 5.1: 10 best 1 Purchase Models (GBM)

ntree	shrink	int.depth	mean test AUC	sd test AUC
1000	0,005	1	0,81700	0,02438
500	0,01	3	0,81697	0,02046
5000	0,001	1	0,81691	0,02432
500	0,01	1	0,81690	0,02439
5000	0,001	3	0,81679	0,02102
1000	0,005	3	0,81670	0,02077
1000	0,01	1	0,81599	0,02194
2000	0,005	1	0,81590	0,02198
100	0,1	1	0,81510	0,02226
2000	0,01	1	0,81450	0,02043

Table 5.2: 10 best 2 Purchases Models (GBM)

ntree	shrink	int.depth	mean test AUC	sd test AUC
500	0,01	1	0,79029	0,04473
1000	0,005	1	0,79025	0,04480
5000	0,001	1	0,79018	0,04475
500	0,005	1	0,78903	0,04034
2000	0,001	1	0,78734	0,04060
2000	0,001	3	0,78720	0,03496
500	0,005	3	0,78668	0,03782
100	0,1	1	0,78506	0,04457
1000	0,01	1	0,78482	0,04469
2000	0,005	1	0,78476	0,04460

Table 5.3: 10 best 3 Purchases Models (GBM)

From that point onwards the model tends to overfit progressively and quality of fit is lowered if the trees are allowed to grow deeper. If enough trees are created, eventually all models will reach a point when the training data is fit with no mistake having a training AUC of 100%, as visible in figure 5.1. On the other hand, if depth is controlled better results are obtained. This can be observed in figure 5.2 as the zone with lower depths provide the best results.

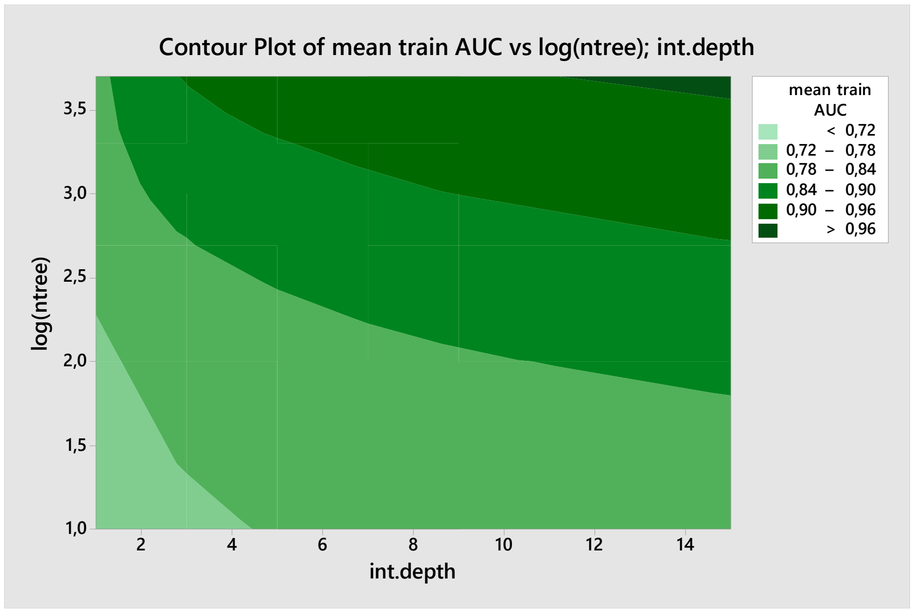


Figure 5.1: Training AUC of 1 purchase model. Source: own creation

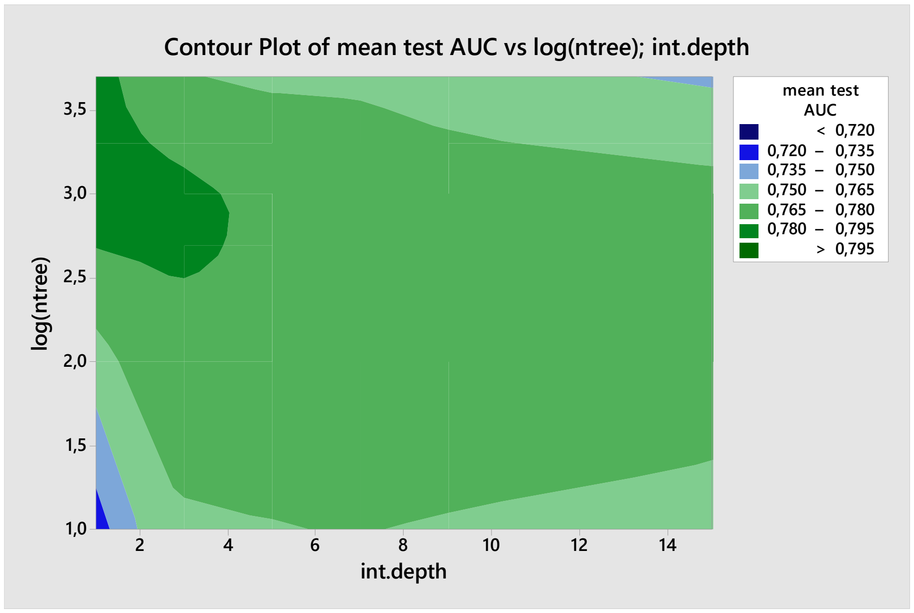


Figure 5.2: Test AUC of 1 purchase model. Source: own creation

Interaction depth parameter takes this name because it is also defining the maximum level of variable interaction possible in each tree. In the logistic regression model done by P. Casas, some low level interactions were present, but using gradient boosting method it has not been mandatory to allow interactions, hence int.depth higher than 1, to obtain the best results. The same quality of fit is obtained by including models with order 1 interactions or allowing them up to order 3 or higher. The hypothesis proposed to explain this is that all interactions present in the logistic model are related to variables which, as shown in the next chapter, have low importance in GBM. It can be said that these variables are almost not used by the GBM model, therefore its interaction must have an even lower importance, making the mtry=1 having no impact in the final result. However, this should be analysed in deep and tested thoroughly.

2. The relation between number of trees and shrinkage must remain constant

In chapter 2 Theoretical framework it was explained that the shrinkage parameter is key to avoid overfitting. It was also mentioned that if the shrinkage parameter is small the number of trees should be higher to allow the model enough iterations to assimilate all the available information in the data. This relation has been effectively appreciated when real data has been used and it is the reason why several models with interaction depth of 1 or 3 achieve similar AUCs. As long as the relation between trees and shrinkage is correct the results will have good and consistent quality. This is shown in figure 5.3 (example from 3 purchases model), where it is possible to see that the relation is linear and with slope of -1 in logarithmic scale.

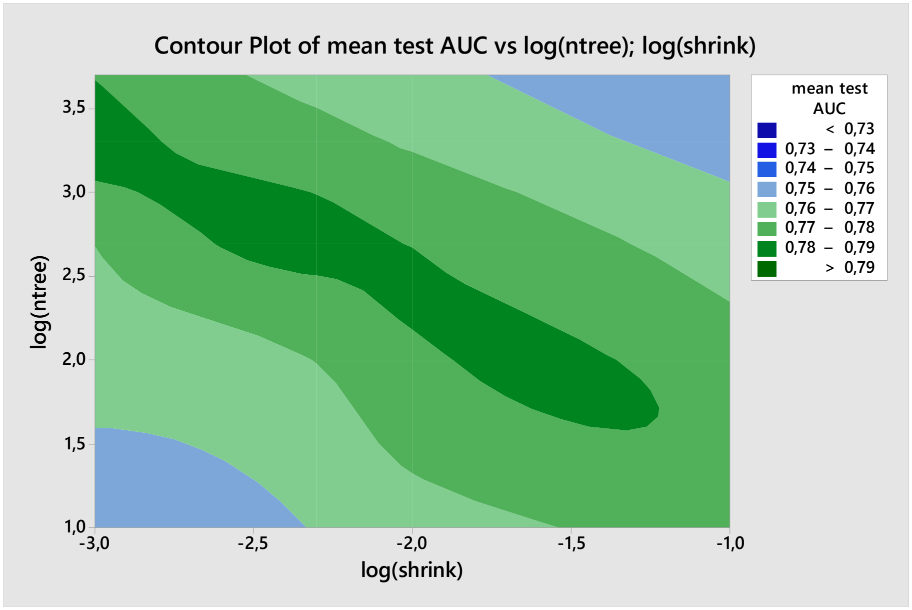


Figure 5.3: relation between number of trees and shrinkage (data from 3 purchases model). Source: own creation

5.1.2 Random Forests

The best performing models using random forests are:

- **1 purchase model: Test AUC= 0.77977** (0.0125); Train AUC= 0.77537. ntree=500, mtry=4, nodesize=20
- **2 purchases model: Test AUC= 0.81358** (0.0205); Train AUC= 0.80579. ntree=500, mtry=5, nodesize=100
- **3 purchases model: Test AUC= 0.79048** (0.0336); Train AUC= 0.78669. ntree=5000, mtry=1, nodesize=10

If treated like the same model, random forests consistently outperforms bagging in all situations, although in some cases the difference is not significant. This is natural as the first is an upgrade based on the second, but for sake of clarity the best bagging models are described below:

- **1 purchase model: Test AUC= 0.77389** (0.0100); Train AUC= 0.77271. ntree=1000, mtry=12 (all variables), nodesize=20

- **2 purchases model: Test AUC= 0.80672** (0.0218); Train AUC= 0.80289. ntree=5000, mtry=27 (all variables), nodesize=100
- **3 purchases model: Test AUC= 0.77790** (0.0380); Train AUC= 0.77203. ntree=5000, mtry=30 (all variables), nodesize=100

Random forests and bagging are easy models to create as there is no risk to overfit the training data. As decision trees are robust against outliers and do not require variable creation or similar the preparation work is low. The main tuning parameter is the number of variables to consider at each split, which will be analysed later, but if a bagging model is used that tuning is not even there. In summary creating a random forest model is quite easy compared to other statistical models. As apart from data cleansing, the only remaining thing to do is to try several mtry parameters (which are recommended by literature to be √p for classification or p/3 for regression, being p the number of variables [2]). And to ensure the number of trees to grow is big enough so the model is able to use all information in the data, and not so big that the calculation time becomes onerous. But even that is not critical as in no case a small variation of parameters will create important variations in the result, for example if 1500 trees are used instead of 2000 the quality of fit will be similar, as well as the calculation time. This makes this model adequate to be updated frequently with new data, hence improving the model systematically. On top of that, some comments will be given to explain evidences found in this study.

1. <u>Bootstrapping based trees do not overfit</u>

Random forests models, and consequently bagging models, do not overfit the data. As explained in the theory, since every new tree is averaged with the existing ones adding more trees will not lead to overfitting. This has been tested using real data, and in all cases the training AUC has not been higher than the test AUC. This is repeated among all models. Figure 5.4 is shown as an example.

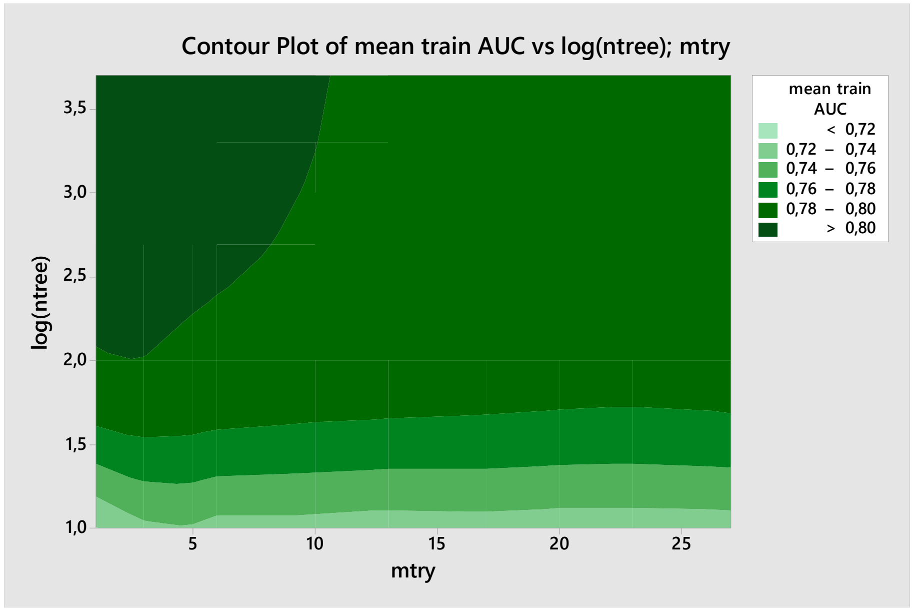


Figure 5.4: train AUC of 1 purchase model. Source: own creation

2. If number of variables is high, low mtry and high number of trees work better (with this dataset)

The behaviour of the number of variables to choose from at each split (mtry according to the name in the R package) has not been the expected. The model with data from only one purchase works as expected and the best results appear with values of mtry around √p (3.46 so 3 or 4 variables from the total of 12), see table 5.4.

ntree	mtry	nodesize	mean test AUC	sd test AUC
500	4	20	0,77977	0,01250
5000	5	20	0,77955	0,00927
1000	3	20	0,77949	0,00965
5000	3	20	0,77914	0,00968
2000	5	20	0,77909	0,01015
5000	4	20	0,77886	0,01045
2000	3	20	0,77885	0,00875
2000	4	20	0,77881	0,01091
1000	4	20	0,77881	0,01097
5000	3	10	0,77861	0,01108

Table 5.4: 10 best 1 purchase models (RF)

On the other hand, models with 2 and 3 purchases work differently and the best models tend to appear also with mtry's of 1 and 3 plus high number of trees like 2000 or 5000, see table 5.5, 5.6 and figure 5.5. This effect is clearer in the 3 purchases model, which has more variables than the one with 2 (30 vs 27 variables). This is strange as mtry of 1 means the model is not allowed to choose which variable to use at any split, hence the model can only choose the split point for that variable. It might seem counter intuitive that the best models are the ones where the system is not allowed to choose even which the best variables are.

ntree	mtry	nodesize	mean test AUC	sd test AUC
5000	1	10	0,79049	0,03359
2000	1	10	0,79022	0,03373
2000	1	5	0,79000	0,03465
5000	1	1	0,78986	0,03504
5000	1	5	0,78967	0,03465
5000	3	100	0,78961	0,03247
1000	1	100	0,78949	0,03334
2000	1	1	0,78946	0,03568
1000	6	100	0,78945	0,03576
5000	6	100	0,78945	0,03450

Table 5.5: 10 best 3 purchases models (RF)

ntree	mtry	nodesize	mean test AUC	sd test AUC
500	5	100	0,81358	0,02054
1000	3	10	0,81346	0,01448
2000	3	10	0,81339	0,01476
2000	1	1	0,81332	0,01733
5000	3	10	0,81322	0,01501
2000	5	100	0,81315	0,01967
500	3	10	0,81309	0,01408
1000	5	100	0,81298	0,01940
5000	5	100	0,81296	0,01947
5000	1	1	0,81287	0,01779

Table 5.6: 10 best 2 purchases models (RF)

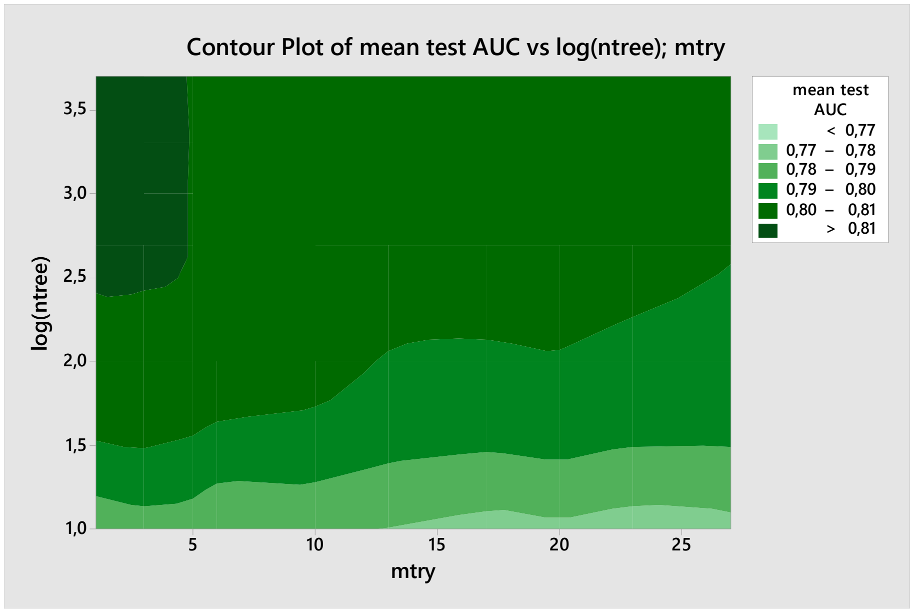


Figure 5.5: test AUC of 2 purchases model. Source: own creation

The proposed explanation for this fact is detailed here. The 2 and 3 purchases models have more variables as more information is available, making it possible to calculate, for example, the increase on expenditure from first to second purchase or the growth in the number of products bought between the first and the second or third purchase. But these added variables are also highly correlated among them. For instance, the higher the expenditure increase between the first and the second purchase, the higher the total value spent will be, for example. The result of this is that while it might seem the number of variables has doubled, the information added is not that much, so actually there are much less than 27 or 30 independent variables. Anyway, why should the model perform better by not being allowed to choose between these 30 correlated variables? The answer to this falls back again to the theory and the difference between random forests and bagging. The added value of RF is that, since the algorithm is not allowed to choose from all variables, different variables than the strongest ones must be chosen. This adds randomness to the model, correcting the tendency of trees to having high variability (making the model too much dependent of the used dataset).

In this particular case, since several variables are highly correlated in the 2 and 3 purchases models, letting the algorithm choose from the total value or the value increase is practically the same. For these reason the mtry value gets reduced to 1, selecting the variable at random at each tree node, but introducing many trees so the interesting variables appear by chance. Then the algorithm is forced to use them, adding value to the resulting model.

Of course this is only a hypothesis that should be proved with an in depth analysis, which will not be covered in this project. But as a first approach to prove this, the most important variables of the 1 and 3 purchase models have been compared in tables 5.7 and 5.8. In these tables it is possible to see that the most important variable (higher decrease of the Gini index) in the 1 purchase model is, by far, the total value of the purchase. The four most important variables in the 3 purchases model are the total value of the purchases and the individual values of the 1st, 2nd and 3rd purchases (firstp.value, secondp.value and thirdp.value). These three purchases sum up to the total value, so all these variables are highly correlated. What is happening is that, while the important information lies in the same place (value of the purchase) in the 3 purchases model this information is repeated in several variables. So if the algorithm is allowed (if mtry is too high) it will choose always the same equivalent variables. For this reason the best models have mtry of 3 or even 1, adding the necessary randomness to the model, instead of using the expected values of 5 or 6 for mtry.

Variables (1PM)	MeanDecreaseGini
totalvalue	241,23
provincia	92,18
fp.month	66,95
firstp.entropy	34,36
delegacion	31,87
firstp.lines	30,53

Variables (3PM)	MeanDecreaseGini
totalvalue	32,27
secondp.value	23,02
firstp.value	20,22
thirdp.value	19,07
provincia	17,67
valueincrease	15,83
fp.month	9,78
delegacion	9,21
total.lines	6,38
dist	5,62
linesincrease	5,60
entropyincrease	5,25
total.entropy	5,22

Tables 5.7 and 5.8: most important variables of 1 and 3 purchases models respectively (RF)

5.2 Comparison between gradient boosting, random forests and bagging

The objective of this project is twofold. The first one is to understand models based on decision trees, the second one is to compare these decision tree based models to the logistic regression model done by P. Casas. But before doing that in this chapter the three types of decision tree models will be compared.

5.2.1 Test AUC

	PM1	PM2	PM3
Bagging	0.7739 (0.010)	0.8067 (0.022)	0.7779 (0.038)
Random Forests	0.7798 (0.013)	0.8136 (0.021)	0.7905 (0.034)
Gradient Boosting	0.7966 (0.015)	0.8170 (0.024)	0.7903 (0.044)

Table 5.9: Test AUC of all models. Source: own creation

If the quality of the prediction is assessed, GBM model outperforms the rest in the 1 purchase model. In the 2 and 3 purchases models it can be seen that GBM and RF are technically equal as the difference is small compared to the variability of the value among the created models (the standard deviation appears in brackets).

5.2.2 Variable importance

The next tables show the variable importance for the different models created. It is relevant to mention that while bagging and RF use an absolute measurement of the importance with no value limitation (the decrease in the Gini index caused by that variable), GBM uses a relative influence measurement, hence representing a percentage over 100.

1PM Bagging	MeanDecreaseGini	1PM RF	MeanDecreaseGini	1PM GBM	rel.inf %
totalvalue	313,11	totalvalue	241,23	totalvalue	73,27
provincia	111,64	provincia	92,18	provincia	19,07
fp.month	85,30	fp.month	66,95	fp.month	3,14
firstp.entropy	37,68	firstp.entropy	34,36	mercado	2,75
firstp.lines	26,26	delegacion	31,87	firstp.lines	0,79

Table 5.10: variable importance of 1 purchase model

The variables for the 1 purchase model are quite homogeneous among the different tree models, being the first three the same and in the same order. The most important one is totalvalue by far. This is natural as the definition of a good customer is based in part on the amount spent by the customer. This is common among all models.

2PM Bagging	MeanDecreaseGini	2PM RF	MeanDecreaseGini	2PM GBM	rel.inf %
totalvalue	261,43	totalvalue	105,19	totalvalue	70,59
provincia	59,26	secondp.value	71,28	provincia	12,18
fp.month	23,63	firstp.value	49,88	secondp.value	7,16
secondp.value	16,14	provincia	36,95	fp.month	4,02
valueincrease	9,95	valueincrease	27,72	mercado	1,24
firstp.value	7,16	fp.month	12,70	total.families	0,89
mercado	5,64	delegacion	8,69	digita_cli	0,7

Table 5.11: variable importance of 2 purchases model

In the 2 purchases model it is possible to see that the bagging model and the GBM select similar variables up to the 4th choice, and also from that point on the importance starts to be quite reduced in comparison with the first variables. RF is more focused in the value spent, although the province (provincia) variable appears in the 4th position.

3PM Bagging	MeanDecreaseGini	3PM RF	MeanDecreaseGini	3PM GBM	rel.inf %
totalvalue	172,18	totalvalue	32,27	totalvalue	86,09
provincia	39,80	secondp.value	23,02	provincia	7,38
fp.month	14,88	firstp.value	20,22	secondp.value	2,46
thirdp.value	6,51	thirdp.value	19,07	valueincrease	1,22
valueincrease	6,23	provincia	17,67	mercado	0,86
secondp.value	6,03	valueincrease	15,83	total.families	0,76
firstp.value	4,67	fp.month	9,78	firstp.families	0,55
mercado	3,10	delegacion	9,21	fp.month	0,33
total.families	1,99	total.lines	6,38	thirdp.families	0,23
total.entropy	1,55	dist	5,62	total.lines	0,13

Table 5.12: variable importance of 3 purchases model

In the 3 purchase model a similar behaviour is observed related to GBM and bagging, focusing the importance in totalvalue and secondly in the geographical distribution (province). However, RF behave differently and variable importance is not concentred in 1 variable. It is important to remind here that this RF model uses mtry=1, so the variable used at each node has been selected at random. For this reason the importance is split among different "value" variables, which as it has already been explained are correlated to the totalvalue variable. What is interesting here is that if the first 4 variables are merged the result is not so different compared to the other models (see Table 5.13), but by distributing the variables RF achieves a test AUC equal to GBM (0.79) and leave bagging behind (with 0.77).

3PM Bagging	MeanDecreaseGini	3PM RF	MeanDecreaseGini	3PM GBM	rel.inf %
totalvalue	172,180	First 4 variables	94,57	totalvalue	86,09
provincia	39,802	provincia	17,67	provincia	7,38
fp.month	14,878	valueincrease	15,83	secondp.value	2,46
thirdp.value	6,512	fp.month	9,78	valueincrease	1,22
valueincrease	6,226	delegacion	9,21	mercado	0,86
secondp.value	6,028	total.lines	6,38	total.families	0,76
firstp.value	4,674	dist	5,62	firstp.families	0,55

Table 5.13: variable importance of RF merging the value of the first 4 variables

As a summary of this chapter it can be said that, although there are some minor differences when selecting the variables, all models choose similar variables to predict the data. This can be said both from the 1, 2 and 3 purchase models as well as of RF, bagging and GBM.

The most important variable by far is the total value of the purchases. The second most important variable is the province of the customer and the third is the month when the first purchase was done. It should be kept in mind that these indicators in decision trees show the most important variables (e.g. fp.month) but not the splitting points inside the variables (e.g. which months define a good customer) so this information is not available to be compared.

In the next chapter the GBM model will be compared to the logistic regression model of P. Casas project. GBM has been chosen as best method as it obtains the highest test AUC and has a more consistent choice of variables for all the models with 1, 2 or 3 purchases.

5.3 Comparison between gradient boosting and logistic regression

This chapter follows the same structure as the previous one. In this case the gradient boosting model and the logistic regression model made by P. Casas are compared.

5.3.1 Test AUC

The table below compares the test AUC for all three purchases models between GBM and logistic regression. Standard deviation of the test AUC indicator for the logistic regression models is not available in this case.

	PM1	PM2	PM3
Gradient Boosting	0.7966	0.8170	0.7903
Logistic Regression	0.8025	0.7797	0.7746

Table 5.14: comparison between GBM and logistic regression models for the three purchases models

In terms of prediction of new data it can be said that both models present quite a good quality with test AUC around 80%. If prioritization is to be done between the two one would say that for the 1 purchase model both present technically the same quality of fit. Logistic regression obtains minimal better results, but the difference is less than 1%. For the 2 and 3 purchases models GBM obtains better results. The difference is mildly important in the 2PM, being 3.7% of separation and minor in the 3PM with a difference of 1.6%.

This indicator tells about the quality of prediction made by these models but also interpretability might be of paramount importance when creating a statistical model. This is analysed in the next chapter.

5.3.2 Variable importance

The most important variables for the 1 purchase model (1PM) logistic regression (LR) and gradient boosting (GBM) are:

1PM GBM	rel.inf %	1PM LR	P-value
totalvalue	73.273	Log10(totalvalue)	<0.001
provincia	19.072	mercadoOEM x log10(totalvalue)	<0.001
fp.month	3.140	Market OEM	<0.001
mercado	2.748	Market SI	<0.001
firstp.lines	0.790	Firstp.families	0.001
		mercadoSI x log10(totalvalue)	0.002
		Fp.seasonWINTER	0.006
		Firstp.entropy	0.008

Table 5.15: variable importance comparison between GBM and logistic regression for the 1PM. Green show common variables, red show non-common variables

The most important variables for the 2 purchase model (2PM) logistic regression (LR) and gradient boosting (GBM) are:

2PM GBM	rel.inf %	2PM LR	P-value
totalvalue	70.588	Log10(totalvalue)	<0.001
provincia	12.179	mercadoOEM x log10(totalvalue)	<0.001
secondp.value	7.157	Market OEM	<0.001
fp.month	4.017	Market SI	0.001
mercado	1.244	Total.families	0.005
total.families	0.894	Digita_cli x log10(totalvalue)	0.008
digita_cli	0.701	mercadoSI x log10(totalvalue)	0.014
		Digita_cli	0021
		Fp.seasonWINTER	0.025

Table 5.15: variable importance comparison between GBM and logistic regression for the 2PM. Green show common variables, red show non-common variables

The most important variables for the 3 purchase model (3PM) logistic regression (LR) and gradient boosting (GBM) are:

3PM GBM	rel.inf %	3PM LR	P-value
totalvalue	86.089	Log10(totalvalue)	<0.001
provincia	7.380	mercadoOEM x log10(totalvalue)	<0.001
secondp.value	2.458	valueincrease	0.001
valueincrease	1.215	Market OEM	0.001
mercado	0.862	Fp.seasonWINTER	0.012
total.families	0.756	Market SI	0.026
firstp.families	0.549	Digita_clientS	0.028
fp.month	0.332		
thirdp.families	0.225		
total.lines	0.133		

Table 5.15: variable importance comparison between GBM and logistic regression for the 3PM. Green show common variables, red show non-common variables

What can be seen directly is that the logistic model requires much more treatment to really identify the relevant variables. This can be appreciated by the appearance of the logarithm of base 10 transformation applied to the total value, which is unnecessary in decision trees (actually this transformation was tested in decision trees and the results did not change). Also by the presence of divided categorical variables like market that needs to be split within 2 dummy variables “Market OEM” and “Market SI”. And finally interactions need to be also additionally created in the form of multiplications between existing variables. All this is present by default, if significant, in the decision tree models with the only singularity that for GBM the interaction.depth parameter regulates the maximum order of interactions (and it is actually set to one in the 2 and 3 PM). As a summary it can be said in favour of decision trees that they require much less treatment to obtain good results. And probably more important, they eliminate the risk of not creating a needed variable, which would lead to a sub-optimal model.

If one wants to know if different models lead to similar most important variables it can be said that total value appears in all cases, as it happened also with RF and bagging. The second most important variable for GBM is province (in all PM models), which does not appear in any logistic regression model, not even in any type of similarly geographic variable. On the other hand, the second most important variable for logistic regression is market, which appears in 4th or 5th position in GBM. Secondary variables related to the time of the year when the first purchase was done appear in both models. In GBM these information appears as the month of the first purchase (fp.month) while logistic regression evaluates whether the first purchase was done in winter or not (fp.seasonWINTER). Finally certain variables appear in both cases for certain models, like number of lines/families in the 1PM, digital_clients and total.families in the 2PM and value increase in the 3PM. It is worth mentioning that in GBM it is not possible to see which interactions are strong as the variable importance indicator shows the importance of a variable in the overall, including main effects plus interactions.

As a summary it can be said that both models agree that totalvalue is the most important variable. There is agreement that some indicator about the time when the first purchase was relatively important, although not strongly important. Finally, some important variables for one model are not mentioned in the other. All specific cases can be identified in the tables in dark red but the more notorious case is the province.

Conclusions

The conclusions of the project will be divided in the three areas of analysis: parameter sensitivity, comparison between decision tree models and comparison versus logistic regression model. But before entering in that level of detail some concepts from the theory have been proved when decision tree models have been created with real case data.

The data processing effort to adapt the data to the model's requirements is almost null. This does not refer to cleansing the dataset, creating wanted prediction variables (like season of first purchase based on month information) or exploratory analysis, as these are highly important and need to be done always. The time saved here is thanks to the intrinsic robustness of decision trees that avoid the need for additional categorical variables or identification of outliers. The models used process the data as it is and usually the only transformation needed refers to converting the response variable to a specific class or format, like 0-1 values or logic class (TRUE-FALSE).

All decision tree models created showed also powerful prediction power, equalling or overtaking the AUC obtained in logistic regression. But this comes at the expense of interpretability of the results.

Sensitivity analysis

The sensitivity analysis conducted on the parameters of the models showed that in gradient boosting models interaction depth has high influence, with the best results obtained with low values between 1 and 3, as it is recommended by literature. It was also demonstrated the linear balance needed between shrinkage parameter and number of trees. If one is increased the other needs to be decreased and vice versa to avoid decreasing the model prediction capability.

The best GBM models obtained for the 2 and 3 purchase models use tree depth of 1. This is expected as it is the recommended value by research papers and books. However, to use a depth of 1 means that interactions between variables are not allowed. It is interesting to see that GBM obtains the best results without allowing variable interaction although the logistic regression model does have relevant interactions. This can be attributed to the iterative creation of the boosting model that integrates this interaction in a different way, but an alternative proposed is that interactions present in the logistic models were related to variables (market and digital client) that had a low importance in the GBM model. Therefore if the main effect of the variable is close to zero it is reasonable to say that the effect of the interaction might also be close to zero in this case. This can be the reason why having an interaction depth parameter of 1 does not prevent to obtain the better models than with mtry>1, as this parameter is not actually blocking any significant interaction. It can also be that the root cause is a combination of both the way boosted trees are built with this lack of importance of interactions in this model. This fact has not been tested in deep in this project, for this reason it is proposed as research to be done in future projects.

Experimenting with random forests and bagging showed that these models are safer to use compared to gradient boosting as the risk of overfitting the data is absent. It was also discovered that the best results were obtained in random forests by not allowing the model to choose which variable to use at each split (mtry=1) and having a high number of trees (in the order of 2000 or 5000 trees). The reason is because in the datasets of 2 and 3 purchases almost all variables are highly correlated. Because of this an mtry=1 is necessary to introduce randomness in the model and avoiding that all trees start with the same strong variable. This extreme value of mtry=1 is explained as if half the variables are not actually independent, to use a higher mtry, √p for example, is not enough and the model will continue to use the strongest variables, hence not obtaining the randomness expected by limiting the number of variables available. This follows the same rationale as why bagging is worse than random forests and, in this case, to use an mtry higher than 1 is not obtaining the full benefits of random forest. However this hypothesis should be tested, and this is proposed as next steps to continue the line of research defined by this project.

Comparison of decision tree models

Comparing the models showed that gradient boosting was the best performer in the model with 1 purchase with test AUC of 79.6% and random forests and gradient boosting were tied in the 2 and 3 purchases with test AUC of 81.5% and 79.0% respectively. Bagging was left behind all cases but with a reduced difference, around 2% worse in test AUC.

Comparing the most important variables in all three models showed that these variables are shared among GBM, RF and bagging. The strongest variable by far is the total amount spent by the customer. The second most important variable is the province of the customer and the third is the month of the first purchase. Comparing results between the models with 1, 2 or 3 purchases also led to the same conclusions about the most important variables. This concludes that adding information about subsequent purchases after the first one does not add significant information that the models can use to improve the results.

Comparison with the logistic regression model

In order to benchmark the obtained models the best decision tree model (GBM) has been compared to the logistic regression model made by P. Casas based on the same dataset. Apart from the basic differences already explained at the beginning of this chapter regarding robustness of decision trees when it comes to data acceptance, performance and variable importance have also been assessed.

The AUC indicator has been used in both projects making the comparison easy. All three models in both methods obtained a test AUC around 80% which is quite high and provides good prediction capabilities. In the 1 purchase model both obtained similar values close to 80%, with logistic regression (80.25%) slightly outperforming GBM (79.66%) by 0.6%. However in the 2 and 3 purchases model this was inverted and GBM outperformed logistic regression in the 2PM by 3.7% (81.70% vs 77.97%) and 1.6% in the 3PM (79.03% vs 77.46%). In general it can be said that performance were similar in this dataset, although if prediction was to be the only goal GBM would be preferable due to the simplicity of feeding the model with new data and slightly better performance.

When variable importance have been evaluated both models agreed that the most important variable by far is the total value of the purchases done by the customer. This is also motivated by the fact that good customers are defined in part depending on the amount spent. Both models also agreed that some timing variable about when the first purchase was done is also important, with logistic regression focusing in whether the purchase was done in winter or not (this level of variable interpretation is not available in decision trees).

On the other hand, disagreements have also been found. For example the second most important variable in GBM is the province of the customer but this does not appear in logistic regression, or any similar variable referred to geographical information. And also while market is quite important for logistic model it has low relevance in GBM.

As a summary it can be said that both models use the same variables to extract the majority of information from the dataset, but when they look for secondary supplementary variables they differ in the choices. However, while GBM was better if prediction was the main goal of the project, logistic regression is preferable if variable importance wants to be deeply understood and applied.

Additionally to previous findings, both models demonstrate that adding information about customer purchases after the first one does not add significant information that the models can use to improve the results. This is equivalent to say that a customer is defined as good or bad since the first purchase and the second and third do not modify this behaviour. This was a surprising fact discovered by Casas' project and it has been confirmed with all decision tree models built in this project.

References

Bibliographic references

[1] CASAS, P (2019). *New Customer's Classifier,* Barcelona: ETSEIB - UPC.

[2] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An introduction to statistical learning with applications in R*. New York: Springer.

Additional bibliography

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning, second edition: data mining, inference, and prediction.* New York: Springer.

Breiman, L. (2001). *Random Forests*. Machine Learning 45, 5–32

Freund, Y., & Schapire, R. E. (1997). *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*. Journal of Computer and System Sciences, 55(1), 119–139.

Friedman, J. H. (2002). *Stochastic gradient boosting*. Computational Statistics & Data Analysis, 38(4), 367–378.

(Page left blank for printing purposes)