BACHELOR'S DEGREE THESIS

Degree in Statistics

Title: Guide to Spark Machine Learning for credit scoring

Author: Álvaro Orgaz Expósito

Advisors: Ana María Pérez Marín, Catalina Bolancé Losilla

Department: Econometrics, Statistics and Applied Economics

Academic year: 2017-2018

In God we trust, all others bring data.

W. Edwards Deming

SUMMARY AND KEYWORDS

This bachelor's degree thesis aims to develop a predictive analytics guide for credit fraud detection using the *Big Data* tool *Spark*. Thus, the essence of this project is structured in three main linked sections which combine theory and practice.

The first part is a description of the problem and concepts about credit risk as well as its historical context. The second section contains a theoretical research in predictive algorithms, frequently known as *machine learning*[1] or *artificial intelligence*[2] models. The third part is a real case practical application of the studied models for predicting the probability of default for a given dataset.

However, nowadays one of the most common problems in predictive analytics is the huge amount of available data, and it gives meaning to the concept of *Big Data*. Thus, this project will use the tool *Spark*, which is an engine for processing *Big Data*.

The main keywords of the thesis are *data science, machine learning, artificial intelligence, credit scoring, Big Data, Spark, analytics, linear and non-linear models.*

Summary and keywords in the official language

El títol del projecte és: Guia sobre l'aprenentatge automàtic amb Spark pel risc creditici.

Aquest projecte final de grau pretén desenvolupar una guia sobre algoritmes predictius aplicats a la detecció del frau creditici utilitzant una eina de *Big Data* anomenada *Spark*. Així doncs, l'essència d'aquest projecte s'estructura en tres seccions enllaçades les quals combinen teoria i pràctica.

La primera part és una descripció del problema i conceptes sobre el risc creditici així com el seu context històric. La segona secció conté una investigació teòrica en algoritmes predictius, freqüentment vinculats als conceptes d'aprenentatge automàtic o models d'intel·ligència artificial. La tercera part és una aplicació pràctica dels models estudiats a un cas real per predir la probabilitat d'impagament per a un determinat conjunt de dades.

Malgrat això, actualment un dels problemes més comuns en els projectes d'algoritmes predictius és la gran quantitat de dades disponibles, la qual cosa dóna sentit al concepte del *Big Data*. Així doncs, aquest projecte utilitzarà l'eina *Spark*, la qual és un motor de processament de grans quantitats de dades.

En conclusió, aquesta tesi final de grau serà un manual per aquells usuaris que vulguin aprendre sobre: la detecció i gestió del risc creditici amb algoritmes predictius; la teoria que hi ha darrere els principals algoritmes en l'àrea de l'aprenentatge automàtic i la intel·ligència artificial; i

[1] Field of computer science that uses statistical techniques to give computers the ability to learn with data.
[2] Theory and development of computer systems able to perform tasks that require human intelligence.

l'aplicació d'aquests a un cas real des del plantejament del problema fins a la presa de decisions (incloent-hi el codi de programació necessari).

Les principals paraules clau de la tesi són: *ciència de dades, aprenentatge automàtic, intel·ligència artificial, detecció del risc creditici, grans quantitats de dades, Spark, anàlisis de dades, models lineals i no lineals.*

American Mathematical Society classification

The thematic classification of this thesis according to the American Mathematical Society corresponds to the section of Computer Science (68-XX), concretely:

- Artificial intelligence (68Txx)
- Algorithms (68Wxx)

Official AMS classification document at https://mathscinet.ams.org/mathscinet/msc/pdfs.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Definition of the problem

In this thesis, the problem in question arises from the business model of the credit sector: clients borrow money from financial service companies (typically banks but nowadays also *peer-to-peer Fintech start-ups*[3] which are gaining ground in the sector) with a certain interest rate which is the profit for the companies and/or investors. For this reason, customer loyalty and good risk indicators are the keys to the success of credit businesses.

Although in this sector there are several different types of methodologies depending on the business model, from classic banks with almost no digitalization to 100% digital *peer-to-peer start-ups* that may not even have a banking license and only connect lenders with borrowers, the typical customer journey map is:

- First, customers apply for a loan and provide the required data.
- Secondly, clients are pre-accepted or not based on the declarative data.
- Then, companies contact pre-accepted customers and check their documentation for having the verified information.
- Consecutively, customers are financed or not based on the checked data.
- Finally, companies manage the credit risk during the life of the loan.

However, in the competitive market, customers have a lot of available options or credit companies to choose between different prices and products. For this reason, apart from managing the credit risk or miss payments, companies must invest a lot of money in marketing for attracting clients. Furthermore, lots of them are not financed because they do not comply with the risk requirements once the documents and personal information are checked, and for this reason, it is also important to have good ratios of financing over pre-accepted customers because if not, companies would be investing money in marketing with no profit since the major part of the clients are not financed.

In conclusion, the success of a credit company mainly lies in selecting and filtering well the customers (or rejecting determined profiles) that will be financed and will not default. Then, these are the key points to manage:

- Marketing costs balanced with the financing rate.
- Risk metrics balanced with the interest rate because, in the end, there is not too much risk (target miss payments rate assumed) but there is a too low-interest rate for the obtained risk.
- Digitalization and automatization of the whole process to enhance the user experience and reduce the operational costs.

For those who do not imagine how the customer selection is done, these are examples that companies currently use:

[3] Method of debt financing that enables individuals to borrow and lend money without an official financial institution.

- Establishing rules for rejecting customers with determined characteristics.
- Establishing a risk score model for punctuating every client and assessing the potential risk, then companies can offer a different pricing or interest depending on the risk level of the clients and rejecting those whose are potentially too risky.
- Rejecting customers that are in national public or private databases of defaulters.

1.1.1. *What is credit scoring?*

Credit scoring is a method of evaluating the credit risk of customers when they apply for a loan. Using historical financial information and statistical techniques credit scoring tries to identify the effects of customers characteristics on miss payments. The score can be used to rank the loan applicants or borrowers in terms of risk. A well-designed model should give higher scores to borrowers whose loans will perform well and lower scores to borrowers whose loans will not. However, no model is 100% accurate and some bad customers will receive higher scores than some good ones.

According to the data, information about borrowers is obtained from credit bureaus and from their loan application such as the applicant's monthly income, outstanding debt, financial assets, how long the applicant has been in the same job, whether the applicant has defaulted a previous loan, whether the applicant owns or rents a home, etc. In short, all potential variables that can be related to customer performance in terms of risk or defaults.

1.1.2. *Classic credit scoring methods: the role of statistics*

The question has always been the same: *Can I count on the borrower to repay?* Before credit scoring models or algorithms, lenders assessed the risk of customers based on subjective factors such as payment history, word-of-mouth and home visits. Thus, human judgment was the main factor in deciding who received the credit and it was a slow process and also unreliable because of human errors. However, those qualitative assessments have evolved into quantitative ones over the years.

Credit scores arose in the 1950s and the sector took a big step forward. These scores were statistical models based on correlations or linear models such as *logistic regression* proposed by Cox in (1958), and built using payment information from thousands of actual consumers, which made scores highly effective in predicting consumer credit behaviour. When combined with new technology, scoring models have made the credit granting process fast, efficient and objective, facilitating commerce and helping consumers quickly get the credit they need.

Concretely, the statisticians Bill Fair and Earl Isaac created an automated scoring system which they continued to refine combined with technology and computers to build what became the *FICO* score. They sold their credit scoring idea to banks and retailers around the world providing consumers with the most significant score factors. Apart from declared customer information, they used credit bureaus such as the companies *Experian* and *Equifax* to extract more information about the default state of clients.

1.1.3. *Modern credit scoring methods: the role of machine learning*

Accuracy is a very important consideration in using credit scoring. Even if the lender can reduce the costs of evaluating loan applications by using credit scoring, if the models are not accurate, these cost savings would be eaten away by poorly performing loans.

Over time, the technology has been steadily improving, more data is available and then, more complex algorithms can be applied. As a result, credit scores have also evolved in terms of predictive behaviour.

Currently, although it is beginning to be accepted by banking regulators, *machine learning* and *artificial intelligence* algorithms are gaining traction in the credit sector. For example, some studied algorithms in this thesis (tree-based models and neural networks) allow discerning the relationship between borrower characteristics and the probability of default in a more flexible way than the standard statistical techniques. Although in some cases these algorithms do not beat the traditional statistical models, for example, if there is not too much data, a new era is beginning with *machine learning.*

1.2. Aims

Once the introduction to the problem is done, it is important to define the aims of this bachelor's degree thesis by sections. The main objectives of the theoretical part are introducing the credit sector and doing research about the theory of most common algorithms used nowadays in the areas of *machine learning*, *artificial intelligence* and predictive analytics. The main aims of the practical part are applying the studied algorithms for predicting the probability of default with a dataset of customer characteristics and comparing the predictive behaviour between models. Thus, it is a case with a binary target variable (default or not default) and different types of explanatory features. In conclusion, this thesis will provide the necessary information for developing a predictive analytics project, from the business problem definition to the application of the optimal model.

Beyond this, the Spanish banking regulator only authorizes to use as a maximum the model complexity of the *logistic regression*. It means that black-box models such as tree-based models or neural networks are not currently allowed. That is why banks have to justify their credit scores and rules for financing and rejecting customers, and it is a way of simply understanding what the entities are doing. For this reason, laws have to progress and be adapted to all these new available and widely used techniques.

1.3. Justification

There is no reason to beat around the bush with the thesis justification. Basically, the idea of this project surges from the confusions about *machine learning* and *artificial intelligence* that users have in mind nowadays, which are easy to clarify with a bit of succinct information. Also, the digitalization of the industry is playing a big role and the major part of people do not understand anything about what is behind, for example, this kind of slogans: *"How artificial* *intelligence will impact our lives?"*. For this reason, the credit sector is a perfect candidate for giving sense and a good application example for the studied *machine learning* algorithms. In short, it would be great that a person, without knowledge about these technical fields, understands better what is *machine learning* and its potential uses through its application in the credit sector.

1.4. Structure

This bachelor's degree thesis is structured into five parts. The first one is the introduction which contains information about the problem that this project aims to resolve or research, the objectives and aims, traditional risk management methods versus the newest ones, and the hypothesis that the author proposes before developing the thesis. The second part is the methodology, which explains how this project has been conducted: data sources, general classification of studied and used algorithms, and the computer resources used for developing the practical case. The third section is a theoretical description of the algorithms. The fourth section is the real case application of models with the corresponding analysis of predictive performance, followed by the global conclusions. At the end of the body, you can find the references, the list of figures and tables, and the annexes which contains the *Spark R* code for reproducing the whole project as well as the metadata of the real case dataset and its pertinent analysis.

1.5. Hypothesis

According to the theoretical section about the studied algorithms, the hypothesis is that although the black-box models (so complex that the interpretability becomes difficult) such as neural networks or tree-based models appear harder to understand, maybe they follow basic rules or algorithmic theories that are not as much complicated or ingenious than the simpler models.

According to the practical part of the thesis, one of the hypothesis is that the optimal model depends on the objective of the researcher. At first sight, it seems that when the complexity of the model increases, the predictive power or accuracy is higher but the interpretability decreases. For this reason, probably the *logistic regression* model will give a better interpretation of how every variable affects the outcome, and the neural network, as well as complex tree-based models, will give a better predictive behaviour at the cost of interpretability loss. Nevertheless, the dataset dimensions can influence these hypotheses because it has not a lot of observations and maybe the most complex algorithms do not have enough data for learning and beating the simple models. Let see what happens.

Furthermore, apart from the thesis contents, there is an important point about the coding language. One of the aims of this project is developing the whole predictive analytics project using the *Big Data* tool *Spark* in *R* statistical software. It means that the typical functions that users use in *R* working in local mode do not run. Thus, another hypothesis is that learning the necessary coding skills will be very-time consuming in the thesis development.

2. METHODOLOGY

2.1. Data sources and references

Currently, there are a lot of free books, courses, lectures, repositories, etc. for learning everything, and especially the data science sector is becoming more and more open-source. For developing this bachelor's degree thesis, several data sources from books to data science online platforms have been used for researching about the algorithms and for learning the necessary programming language skills. In the reference section at the end of the thesis, you can find the list of used references.

Furthermore, the author of this thesis tries to capture his particular view of the field in question, and for this reason, the experience of the author is another important data source.

2.2. Classification of used *machine learning* techniques

Machine learning is mainly divided into two fields: *supervised* and *unsupervised.* This thesis mainly works on the *supervised machine learning,* and it refers to techniques where the computer learns without the exhaustive supervision of humans but there is a target variable to predict. In the real case approach the objective is to predict the defaults and basically, it will be done by discriminating the target binary variable using the space of explanatory features. In short, *supervised machine learning* refers to predictive algorithms.

Apart from this field, there is the *unsupervised machine learning.* This term usually refers to techniques where the computer learns without the exhaustive supervision of humans but there is not a target variable. Then, the main objective is exploring the data without trying to discriminate a concrete variable using others. For example, two well-known techniques are: *clustering* which is a segmentation of observations by calculating the mathematical distance between them and creating groups according to the similarities; and *principal components analysis (PCA),* proposed by Hotelling in 1930, which is the creation of a new artificial variables space where the new axis or principal components are a linear combination of the original features and for building it the objective function maximizes the projected variance in fewer axis. This second technique will be used in the models' comparison of the real case approach, and although *PCA* will not be explained in the theoretical section, see Hastie et al. (2008) and Tibshirani et al. (2013) for more information about these techniques.

Once it is explained, this thesis will research and will use six concrete and well-known algorithms. Just for giving a clear structure to the project, they will be divided into four groups according to their algorithmic type. In theoretical section 3, you can find this scheme when describing the theoretical concepts:

Algorithmic group	Model
Tree-based models	Decision tree Random forest Gradient boosted trees

Discriminative classification models	Naive Bayes
Generalised linear models	Logistic regression
Neural networks	Multilayer perceptron classifier

Table 1: Classification of algorithms included in the thesis.

2.3. Computer resources

This bachelor's degree thesis will work with the statistical software *R* but with a particularity that consists of using the *Big Data* engine called *Spark*. The whole project, including the data pre-processing, the training of models, as well as the model validation phase where performance measures are calculated, will be implemented using *Spark* in *R Studio* with the version 3.5.0 of *R* statistical software. In the annexes, you can find the code with comments for developing and reproducing the entire project.

2.3.1. *What is the problem of Big Data?*

Nowadays, the amount of data that is being created and stored on a global level is almost inconceivable, and it just keeps growing exponentially. The volume is as massive that processing data it is becoming a problem in terms of storage and computational capacity. In general, statistics means working with samples of data for doing inference about the whole population, and it enables to work in computers with not such a sophisticated storage and computational capacity. However, when working with millions of observations and several features, the problem of *Big Data* erases because most computers have not the required hardware and functionalities.

Typically, the three Vs of *Big Data* are:

1. *Volume*. The amount of data matters and with *Big Data*, you will have to process high volumes of structured and unstructured data. This can be data of unknown value, such as *Twitter* feeds, clickstreams on a webpage or a mobile app, or sensor equipment.

2. *Velocity*. It is the fast rate at which data is received and acted on. Some internet-based products even operate in real-time and will require real-time evaluation and action.

3. *Variety*. It refers to the many types of data that are available. Traditional data types were structured and fit into a relational database. However, with the rise of *Big Data*, data comes in new unstructured types such as text, audio, and video.

Currently, as *Big Data* has become a capital, there are a lot of available tools for managing it and basically, they consist of two main branches:

a. Getting more powerful hardware for improving the storage and computational capacity of computers. For example, the technological brand *NVIDIA* is investing in their business line of graphics cards (GPUs) for developing a data science package called

Cuda. It integrates the *Python* language with the GPUs for computing *machine learning* with the graphics card instead of working with RAM, improving so much the potential.

b. Distributing data in a cluster service for computing the code and storing data in a remote computer or server, which is more powerful than a single machine. For instance, common products are: sending the code and data for receiving the outputs or connecting to a cluster from a data science language such as *R* or *Python.* The advantage of using these methods is the efficiency because it is only necessary to invest money when you need to execute tasks. Some well-known tools are *Spark*, *Hadoop, MongoDB, Cloudera, Hive* and *Amazon Web Service.* In this thesis, *Spark* is used in *R* when developing the real case section 4.

2.3.2. *Spark, a Big Data engine*

In simple words, *Spark* is a fast cluster computing engine that is being optimized for speed of computations. *Spark* is written in *Scala* programming language but it provides high-level APIs in *Scala, Python* and *R* which means that the *Spark* engine can be run from these softwares. Although it is not possible using the functions of these softwares if they have not an equivalent in *Spark*, it supports a rich set of higher-level functions and packages such as *SparkSQL* for data processing, *sparklyr* for manipulating data and *MLlib* for *machine learning.*

How does *Spark* enhance *machine learning? Python* and *R* are popular languages for data scientists due to a large number of functions or packages that are readily available. But traditional uses of these tools are often limiting, as they process data on a single machine where it becomes time-consuming, the analysis requires sampling, and moving from development to production environments requires extensive re-engineering. To help address these problems, *Spark* provides with a powerful and unified engine that is both fast and easy to use.

Figure 1: Apache Spark official logo.

3. METHODS FOR BINARY CLASSIFICATION

In many situations applying predictive algorithms, the response variable is qualitative or categorical instead of quantitative. For example, gender is qualitative, taking qualitative on values male or female. In this section, we study models for predicting qualitative responses, a process that is known as classification, although these algorithms can predict quantitative variables too. Predicting a qualitative response involves assigning the observation to a class of the response variable and the basis for making the classification is predicting the probability of each category. There are many possible classification techniques and in this chapter, the most widely-used classifiers will be discussed.

Citing Tibshirani et al. (2013), some classification examples include:

A person arrives at the emergency room with a set of symptoms that could possibly be attributed to one of three medical conditions. Which of the three conditions does the individual have?

An online banking service must be able to determine whether or not a transaction being performed on the site is fraudulent, on the basis of the user's IP address, past transaction history, and so forth.

On the basis of DNA sequence data for a number of patients with and without a given disease, a biologist would like to figure out which DNA mutations are deleterious (disease-causing) and which are not.

3.1. Generalised linear models

In statistics, linear regression is used to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). In linear regression, the relationships are modelled using linear predictor functions whose unknown model parameters are estimated from the data. In short, the model takes the form:

$$y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i = x_i^T \beta + \epsilon_i \,, \qquad i = 1, \ldots, n$$

Notation and terminology:

- n is the number of observations (in the real case approach it is the number of customers)
- y is a vector of observed values of the response or dependent variable
- x is the matrix of independent variables with p n-dimensional columns
- β is a (p+1)-dimensional parameter vector where β_0 is the intercept and its elements are the effects or partial derivatives of the y with respect to the x
- ϵ is a n-dimensional vector with the error term which captures all other factors that influence the y other than the regressors x

However, linear regression is not appropriate in the case of a qualitative response. Why not? Suppose that we are trying to predict the final status of a customer in a credit company. In this example, there are three categories in the status response: not pre-accepted, pre-accepted but not financed, financed. We could index these categories as a quantitative response variable, 1 if not pre-accepted; 2 if pre-accepted but not financed; 3 if financed. Unfortunately, this coding implies an ordering on the response classes creating a false difference between categories and if we use linear regression, some of our estimates might be outside the [0,1] interval, making them hard to interpret as probabilities!

Fortunately, making some adjustments to simple linear models we can obtain generalised linear models for pure classification which provides a predicted probability by response classes. Concretely, this thesis will explain and use the generalised linear model *logistic regression* for binary classification.

3.1.1. *Logistic regression*

Consider a defaults data set as in the real case approach of this thesis, where the binary response is one of two categories, 1 if default or 0 if not default. Rather than modelling this y response directly, *logistic regression* models the probability of a particular category.

Now, let see the simple steps for understanding how the *logistic regression* works:

1. Let $P(y = 1 \mid x)$ be the probability that the binary output y is 1 given the features x.

2. Use the logarithm of the odds ratio, instead of directly the probability as another way of interpreting probabilities, which can take on any value between -∞ and ∞. For example, if the probability of default is 0.9, on average nine out of every ten people with an odds of 9 will default, since $P(y = 1|x) = 0.9$ implies an odds of $0.9/0.1 = 9$.

$$ln\left(\frac{P(y = 1 \mid x)}{1 - P(y = 1 \mid x)}\right)$$

3. Model the logarithm of the odds ratio using the linear regression explained above:

$$\ln\left(\frac{P(y_i = 1 \mid x_i)}{1 - P(y_i = 1 \mid x_i)}\right) = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p\, x_{ip} + \epsilon_i$$

4. Isolate the probability of default from the equation getting the *logit activation function* or *logit link function,* which means that applying the function to the simple linear predictor we can get the output in the probability range [0,1]:

$$P(y_i = 1 \mid x_i) = \frac{e^{\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p\, x_{ip} + \epsilon_i}}{1 + e^{\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p\, x_{ip} + \epsilon_i}}$$

5. Consider the likelihood function for the response (a sequence of Bernoulli trials):

$$L\big(P(y = 1 \mid x)\big) = \prod_{i=1}^{n} P(y_i = 1 \mid x_i)^{y_i}\,(1 - P(y_i = 1 \mid x_i))^{(1-y_i)}$$

6. Estimate the parameters β by optimizing[4] the next *loss function* created from the logarithm of the function above*:*

$$\sum_{i=1}^{n} y_i \ln\big(P(y_i = 1 \mid x_i)\big) + (1 - y_i)\ln\big(1 - P(y_i = 1 \mid x_i)\big) =$$

$$\sum_{i=1}^{n} y_i \ln\left(\frac{\mathrm{e}^{\beta_0+\beta_1 x_{i1}+\ldots+\beta_p x_{ip}+\epsilon_i}}{1+\mathrm{e}^{\beta_0+\beta_1 x_{i1}+\ldots+\beta_p x_{ip}+\epsilon_i}}\right) + (1 - y_i)\ln\left(1 - \frac{\mathrm{e}^{\beta_0+\beta_1 x_{i1}+\ldots+\beta_p x_{ip}+\epsilon_i}}{1+\mathrm{e}^{\beta_0+\beta_1 x_{i1}+\ldots+\beta_p x_{ip}+\epsilon_i}}\right)$$

7. Once the optimal β are calculated, we finally have the model *logistic regression*. Now, we can predict the default probability of the customer i with explanatory variables x_i:

$$P(y_i = 1 \mid x_i) = \frac{e^{\beta_0+\beta_1 x_{i1}+\ldots+\beta_p x_{ip}+\epsilon_i}}{1+e^{\beta_0+\beta_1 x_{i1}+\ldots+\beta_p x_{ip}+\epsilon_i}}$$

Note: In the next sections, the abbreviation LR means *logistic regression.*

3.2. Discriminative classification models

The objective of the discriminant analysis is to find a direction or linear combination of the original numerical variables that best separate categories from a categorical response variable, in the real case approach the binary variable *Defaulted.* In this way, the space of explanatory variables is divided into regions, and we can predict which class a new individual will belong depending on which of the regions of the space is projected.

In this algorithmic group, two well-known models are *naive Bayes* and *support vector machine.* However, this thesis will explain and use only the first one because *support vector machine* is not implemented in *Spark R* environment for what this thesis needs. More information can be found in Hastie et al. (2008) and Tibshirani et al. (2013).

3.2.1. *Naive Bayes*

The *logistic regression* involves directly modelling $P(y = 1 \mid x)$ and in statistical jargon, it is known as the conditional distribution of the response y given the p predictors x. In the case of the *navie Bayes*, introduced into the community under a different name in the early 1960s, it uses a less direct approach to estimating these probabilities.

Now, let see the simple steps for understanding how the *naive Bayes* works:

1. Establish the prior probability of the response classes which means the probability of that a randomly chosen observation comes from every class. If we have a random

[4] The implemented model function in the packages of statistical softwares such as *R* have a determined method of optimization explained in the manuals.

sample of y from the population, simply compute the fraction of the observations that belong to the k class.

$$\pi_k\,, \qquad k = 1, \dots, K$$

2. Establish the probability distribution of the p predictors x separately in each of the response K classes. In the real case approach, it means establishing the probability distribution of each x for customers defaulters $(y = 1)$ and for non-defaulters $(y = 0)$ separately.

$$f(x_j \mid y = k)\,, \qquad j = 1, \dots, p\,, \qquad k = 1, \dots, K$$

3. Establish the probability distribution of the p predictors x jointly in each of the response K classes. But it tends to be more challenging, unless we make assumptions for these functions. Then, assuming that $x_1, \dots, x_p$ are statistically independent given the response class k:

$$f(x \mid y = k) = \prod_{j=1}^{p} f(x_j \mid y = k)$$

4. Use the Bayes theorem to computing the posterior probability that an observation with explanatory variables x belongs to the class k:

$$P(y = k \mid x) = \frac{\pi_k\, f(x \mid y = k)}{f(x)}$$

   but as the denominator $f(x)$ is a constant we only need to develop a classifier that approximates the Bayes theorem for the posterior probability that an observation with explanatory variables x belongs to the class k:

$$NB_k(x) \approx \pi_k\, f(x \mid y = k) \approx \pi_k \prod_{j=1}^{p} f(x_j \mid y = k)$$

   Every individual is assigned to the class with $NB_k(x)$ maximum and in spite of its simplicity it works well in practice.

5. Once we have the classifier, we finally have the model *naive Bayes.* Now, we can predict the default probability of the customer i with explanatory variables x_i with the Bayes theorem:

$$P(y_i = k \mid x_i) = \frac{\pi_k \prod_{j=1}^{p} f(x_{ij} \mid y = k)}{\sum_{m=1}^{K} \{\, \pi_m \prod_{j=1}^{p} f(x_{ij} \mid y = m)\,\}}$$

In the 3D graphic at the following page, you can find we can see all the probability distributions and concepts explained above for a binary response variable (y equals 0 or 1) with prior probabilities of 0.8 and 0.2 respectively, and it displace the NB classifier. You will see in next sections that it is similar to the real case approach.

Note: In the next sections, the abbreviation NB means *naïve Bayes.*

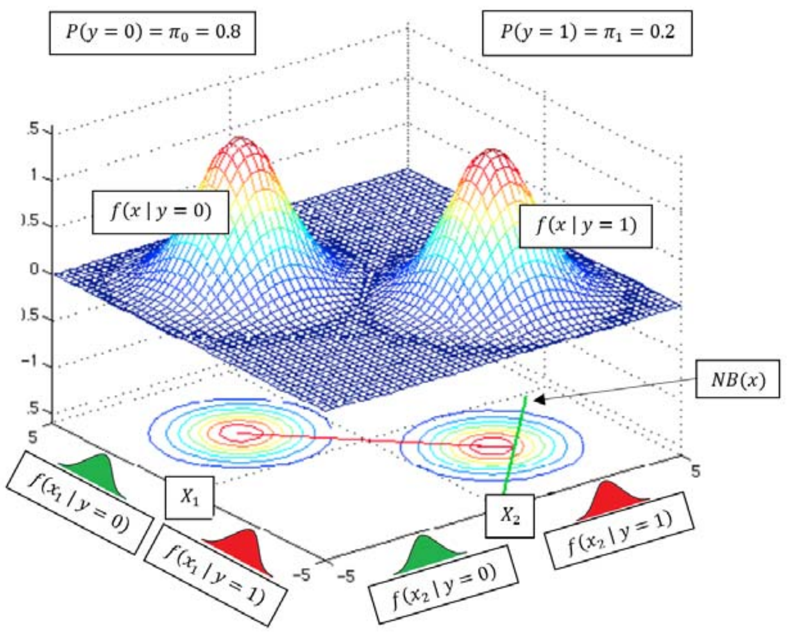

Figure 2: Naive Bayes graphic from https://www.byclb.com/TR/Tutorials/neural_networks/ch4_1.htm.

3.3. Tree-based models

In this section, tree-based methods for classification will be described, although these models can be used also for regression problems with a quantitative target variable. They are non-parametric methods with the objective of predicting the value of a response variable, in the real case approach if a customer defaults or not, based on decision rules derived from the data. In this way, observations are divided into several homogeneous groups with respect to the response variable and the aim is to discriminate against. In short, these models make splits of the data with the explanatory variables (such as *less than 45 years old or not*) creating *internal nodes*, from the *root node* to the *leaf nodes*. The next schema describes this concept:

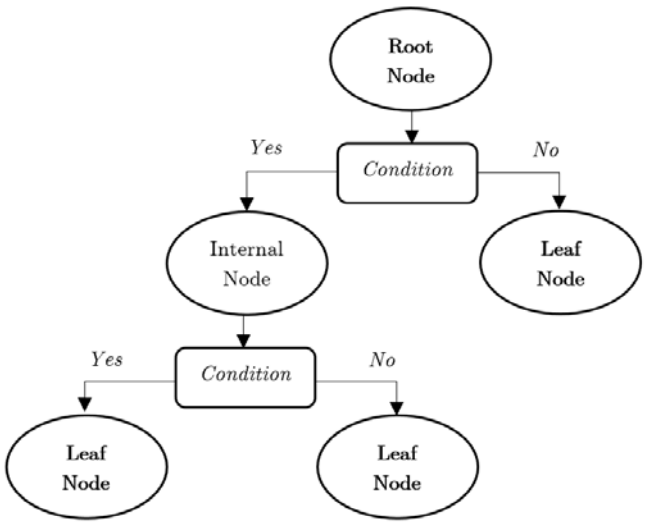

Figure 3: Schema of tree-based models (nodes and splits).

3.3.1. *Decision tree*

As it has been explained, *decision trees* are tree-based models that make splits in the data until they get the *leaf nodes*, in short, prediction via stratification of the feature space.

In more detail, imagine that the predictor space, a set of explanatory variables $x_1, \dots, x_p$, is divided into J distinct and non-overlapping regions $R_1, \dots, R_J$. The goal is to find boxes $R_1, \dots, R_J$ as homogeous as possible with respect to the response variable. Then, every observation that falls into the region Rj have the same prediction, which is simply:

- For a quantitative response: the mean of response values for the training cases in Rj.
- For a categorical response: the % of response values equals to the class k for the training observations in Rj giving a probabilities distribution of the response K classes.

Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes. For this reason, a recursive binary splitting is used beginning at the top of the tree (at which point all observations belong to a single region) and then successively splitting the predictor space via two new branches every node split. It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than picking a split that will lead to a better tree in some future step.

But which criteria is used to split the data? Which is the exact process for getting the model? Now, let see the simple steps for understanding how the *decision tree* works:

1. Establish the split criteria for choosing the optimal data split in every split of the features space. It will be the impurity of the resulting nodes with respect to the response variable and in the classification case, it corresponds to the next *Gini Index* formula. Basically, with the example of the real case approach, the idea is getting as many defaulters as possible ($y = 1$) in a resulting node of the split and as much non-defaulters ($y = 0$) as possible in the other.

$$\mathrm{i}(t) = 1 - \sum_{\mathrm{j}=1}^{\mathrm{K}} P(\mathrm{y} = j \mid t)^2$$

   Notation and terminology:
   - K is the number of classes in the response categorical variable
   - t represents a resulting node of the tree split
   - $P(\mathrm{y} = j \mid t)$ represents the probability of belonging to class j of the response variable in the node t, basically the % of observations with $y = j$ in the node t

2. Perform the first binary splitting of the explanatory features space. It means considering all predictors $X_1, \dots, X_p$ and all possible values of the cutpoint c for each of the predictors as follows:

   - For quantitative explanatory variables: with a feature Xj and the cutpoint c split the predictor space into the regions $\{X \mid Xj < c\}$ and $\{X \mid Xj \geq s\}$.
   - For categorical explanatory variables: with a feature Xj and a category c split the predictor space into the regions $\{X \mid Xj = c\}$ and $\{X \mid Xj \neq c\}$.

Then, choose the predictor and cutpoint that minimize the impurity $i(t)$ between a resulting node and its parent.

3. Next, repeat the process of step 2, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the impurity of the resulting regions. However, this time, instead of splitting the entire predictor space, only split one of the two previously identified regions in step 2.

4. Again, the process continues until a stopping criterion is reached. For example, in the *Spark R* implementation of the model, the tree construction is stopped at a node when one of the following stopping criteria is met for all *leaf nodes* or identified regions:

    - The node depth is equal to the training *maxDepth* parameter.
    - No split candidate leads to an information gain greater than the minimum *minInfoGain* parameter.
    - No split candidate produces 2 *leaf nodes* which each have at least training *minInstancesPerNode* (parameter) instances.

5. Once we have the tree built, we finally have the model *decision tree.* Now, we can predict the response class of the customer i with explanatory variables x_i passing the customer by the tree and choosing the class with highest % of the *leaf node* in which the customer falls.

Among the points in favour of this method, we find the simplicity of interpretation from the splits, the possibility of visualizing the result graphically and working with numerical and categorical variables at the same time. Also, it is an efficient method computationally, however, if there are some minority classes in the answer variable, it is likely that it will make bad predictions for this class.

Note: In the next sections, the abbreviation DT means *decision tree.*

3.3.2. *Random forest*

This model was proposed by Tin Kam Ho in 1995 and deeply developed by Leo Breiman. The purpose of the *random forest* is to obtain a better classifier from the average of B different *decision trees.* However, as a result of applying the average results of a high number of trees, the method loses the interpretation that was obtained with an individual *decision trees.*

In more detail, the *random forest* includes some concepts or adjustments which are:

a. *Bagging.* It means resampling the single training data (known as *bootstrapping*[5]) to generate B different bootstrapped training data sets and then, train the B trees on these sets in order to average the predictions of them. *Bagging* reduces prediction

[5] The concept of *bootstrap* means resampling a sample of the population for having several pairs of training and test sets. Then, a estimator or model can be trained with the training sets and evaluated in several test sets.

variance over a single tree and improves prediction accuracy at the expense of interpretability.

b. *Fix a number of features to consider in each tree.* Apart from building the trees with different bootstrapped sets, a determined number of explanatory variables will be randomly selected to the splits in every tree. It has a clever rationale because if there is one very strong predictor in the data set most of the trees will use this strong predictor in the top split and consequently, all of the bagged trees will be similar and their predictions will be highly correlated. Thus, it adds diversity to the trees and more reliable results are obtained.

Once we have all the trees built, we finally have the model *random forest.* Now, we can predict the response class of the customer i with explanatory variables x_i passing the customer by all the tree and averaging the decision of all the trees generated.

Note: In the next sections, the abbreviation RF means *random forest.*

3.3.3. *Gradient boosted trees*

This model was proposed by Leo Breiman and deeply developed by Jerome H. Friedman. *Gradient boosted trees* is another approach for improving the predictions from a single *decision tree*. What is the idea behind this procedure? Given the first *decision tree* that predicts the original response variable, a second *decision tree* is fitted with the residual errors (the values of the objective variable minus the predictions) from the first one as the response variable. That is, fitting B trees using the residuals of the previous tree rather than the original response. By fitting small trees to the residuals, we slowly improve the outputs in areas where the model does not perform well. In general, statistical learning techniques that learn slowly tend to perform well and in boosting, unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown.

In the history of *Kaggle machine learning* competitions, they usually master two techniques: tree-based models for structured data and neuronal networks when the data includes images or voice. Traditionally *random forest* predominated in structured data competitions, but another algorithm won it: *gradient boosted trees.*

Now, let see the simple steps for understanding how the *gradient boosted trees* works:

1. Set the output prediction function of the model $\hat{f}(x) = 0$ and the response variable for the first tree $r_i = y_i$ for all i observations in the training set.

   Notation and terminology:
   - y is the original response variable
   - x is the matrix of explanatory features with p variables and n observations

2. Fit the B decisions trees. For $b = 1, \dots, B$, repeat:

   a. Fit a tree $\widehat{f_b}(x)$ with the x as features and r as the response variable to predict.

b. Update $\hat{f}(x)$ by adding a shrunken version of the new tree. It means using a weighting factor called the shrinkage factor or the learning rate to slow down the learning in the model.

$$\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}_b(x)$$

c. Update the residuals which will be the response variable in the next iteration.

$$r_i = r_i - \lambda \hat{f}_b(x)$$

3. Output the *gradient boosted trees* model.

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}_b(x)$$

Although it has not been mentioned, the implemented function in *Spark R* for this model includes *bagging*. Remember that, as in the *random forest* model, it means that each tree is built on a bootstrap dataset randomly sampled from the single training set.

Note: In the next sections, the abbreviation GBT means *gradient boosted trees.*

3.4. Neural networks

3.4.1. *Multilayer perceptron classifier*

The first perceptron algorithm was invented by Frank Rosenblatt in 1957. The *multilayer perceptron classifier* (MLPC) is a classifier based on the feedforward artificial neural network, which consists of multiple layers of nodes or neurons fully connected to the subsequent layer in the network. Nodes in the input layer represent the input data and all other nodes map inputs to outputs applying an activation function to a linear combination of the inputs with the node's weights. The next graphic represents a simple schema of MLPC with n input features, one hidden layer with m neurons and k neurons in the output layer (k classes).

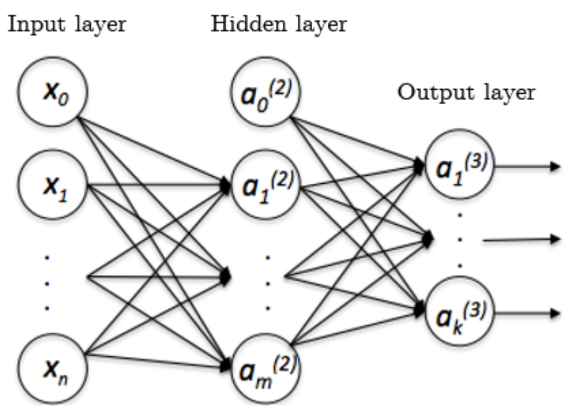

Figure 4: Neural network MLPC example schema.

This model is called feedforward because information flows through the function being evaluated from the input data, through the intermediate nodes, and finally to the output. It means that there are no feedback connections in which outputs of the model are fed back into itself. When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks.

In more detail, the main concepts for understanding the neural network *multilayer perceptron* are:

a. *Activation function.* Except for the input nodes, each node is a neuron that uses a non-linear activation function, it means that in every node, the resulting value of the linear combination of the inputs (outputs of previous nodes) with the node's weights is applied to an activation function. In short, it is the same idea that the *logit link function* in the *logistic regression,* but applied to every node of hidden layers.

   - Nodes in intermediate layers use the *sigmoid or logistic function*:

$$f(z_i) = \frac{e^{z_i}}{1 + e^{z_i}}$$

   - Nodes in the output layer use softmax function:

$$f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

   Notation and terminology:
   - z_i is the resulting value of the linear combination of the inputs (outputs of previous nodes) with the node's weights
   - K is the number of output neurons corresponding to the number of classes in the response variable.

b. *Layers. MLPC* consists of three or more layers (an input and an output layer with one or more hidden layers) with a determined number of neurons or nodes. All they are fully connected with the next layer with a certain weight (parameter to optimize) to every node in the following layer. The number of nodes by layers is:

   - Input layer: corresponds to the numbers of explanatory variables.

   - Hidden layers: the number of hidden layers ranges from one to many and the number of neurons is a tuning parameter with no optimal value for all cases.

   - Output layer: corresponds to the number of classes in the response variable.

c. *Learning or weights optimization.* It consists of optimizing the weights that connect neurons layers, based on the amount of error in the output compared to the expected result. The optimization algorithm repeats a two-phase cycle:

1. *Propagation.* When an input data is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired or true output, using an *error function or loss function* and the resulting error value is calculated for each of the neurons in the output layer.

2. *Backpropagation for updating weights.* Here, the error values are then propagated from the output back through the network for optimizing the weights of each connection in order to reduce the value of the error predictions by some small amount. To learn or adjust weights properly, *backpropagation* is commonly used by the gradient descent optimization algorithm, basically, it calculates the derivative of the error function with respect to the network weights, and changes the weights such that the error decreases.

After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small.

Note: For more information about backpropagation process, visit the following link with an example step by step: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example.

d. *Loss function or error function.* It is a function used in the *learning* that maps values of one or more variables onto a real number intuitively, through the optimization algorithm, representing some *cost* associated with those values. In the case of the implemented *Spark R* function for the binary response, the *loss function* is the *logistic loss function*:

$$L(f(z), y) = -(y \log f(z) + (1 - y) \log\ (1 - f(z))$$

Notation and terminology:

- $f(z)$ is the resulting value of applying the activation function f to the linear combination of the inputs (outputs of previous nodes) with the node's weights.
- y is the vector of true value in the response variable.

Note: In the next sections, the abbreviation NN means neural network *multilayer perceptron classifier.*

4. REAL CASE APPROACH

4.1. Stages of a predictive analytics project

This section attempts to give a brief idea of all the project steps for developing a *machine learning* model:

1. *Variable exploration.* This step decides the features types (nominal, ordinal, numerical, etc.) and explores each variable distribution detecting singular values such as errors or missing values. In this stage, it is typical to use *unsupervised machine learning*[6] for a better data understanding.

2. *Feature selection.* This step performs the variable selection trying to include all possible features that discriminate well the target variable. It is important for the future model application because, for example, it would be important to consider if it is legal to use the gender of clients as an explanatory variable when the model is put into production.

3. *Pre-processing.* This part modifies the dataset for adapting it to the way of reading the data by the models and for optimising the performance. For example:

    - Transforming or imputing missing values for models that cannot manage them.
    - Deciding how many observations within each nominal feature level should be kept in the model or otherwise merged to an *OTHERS* level. Because if a variable has levels with too few observations, the models do not have enough information to learn.
    - Normalization and scaling numerical variables in order to improve model performance.

4. *Parameter tuning.* This step seeks to find the best combination of model parameters, using optimization approaches. This step is critical for most algorithms and interacts with other steps, like pre-processing or feature selection.

5. *Model comparison.* This part accurately compares the predictive performance of several chosen models with performance measures.

6. *Feature importance.* This step computes, for each variable, measures of its contribution to the model. It is important for understanding how the model classifies the target variable and which variables contribute more to discriminate it.

7. *Application of the model.* Analysis of the potential impact of the model and put it into production.

[6] This term usually refers to machine learning techniques where there is not a target variable, and the objective is exploring the data without trying to discriminate the target variable using explanatory features. For example, two well-known techniques are *clustering* and *principal components analysis* (*PCA*).

4.2. Database description and pre-processing

The data used in the real case approach is provided by a *Fintech* which has tracked the risk of its financed customers. The dataset includes many variables that are expected to explain the customer default behaviour. Concretely, the dataset contains 3468 financed customers of the company and 26 variables including the response and the customer identifier. This is the list of variables:

Related to	Variable
Loan application	*Contract ID:* customer identifier unique for everyone *Application Week Day:* weekday of the customer application *Application Hour Grouped:* application hour grouped into 3 ranges *Amount:* loan amount requested in € *Maturity:* duration of the loan in months *Purpose:* objective of the customer loan
Profile	*Age:* age of the customer *Gender:* gender of the customer *Marital Status:* marital status of the customer *People in Household:* number of people in charge of the customer
Professional situation	*Profession Code:* the profession of the customer *Profession Sector:* private or public profession sector *Contract Type:* type of employment contract *Seniority:* number of months since the customer profession start date
Housing situation	*Province:* residence province of the customer *Postal Code ASNEF:* probability of being in ASNEF[7] by postal code *Housing Type:* housing situation of the customer residence *Housing Seniority:* months since the customer housing start date
Expenses	*Rent:* monthly rent cost of the customer in € *Mortgage:* monthly mortgage cost of the customer in € *Amount of Ongoing Credits:* monthly amount of ongoing credits in € *Number of Ongoing Credits:* number of ongoing credits
Revenues	*Income:* monthly salary of the customer in € *Additional Income:* monthly additional income of the customer in € *Partner Income:* monthly partner income of the customer in €
Outcome	*Defaulted:* if the customer has defaulted (1) or not (0)

Table 2: List of dataset variables. In the annexes, you can find the extended metadata.

The dataset is unbalanced and it has about 80% proportion of customers non-defaulters versus 20% defaulters. Having a balanced target variable is an important thing to consider when fitting a model because if there are some minority classes in the outcome variable, in this real case the outcome is binary (defaulted or not), it is likely that it will make poor predictions for these classes because of the scarce information about these categories. For example, in the case of this thesis, if the proportion of classes is too unbalanced (more non-defaulters than

[7] Negative database of defaulters that companies can check to know if a customer is registered there or not.

defaulters), the models would tend to predict lower probabilities of default since it would be more difficult to learn from the characteristics of defaulters.

Regarding the analysis of missing values that can be found in the metadata of the annexes, it has been decided to use the whole dataset with a total number of 3468 customers. Once the data is appropriately pre-processed, it will be divided into two groups (training and test) in a ratio of 75%-25% respectively. It will be explained in the posterior sections 4.4 and 4.5.

Although the dataset is quite clean and complete in terms of data quality, it is necessary some data pre-processing. The treatments include:

Treatment	Variable type	Action
Missing values	Numerical	Modifying missing values by 0 because some models do not support them. This imputation is given by the private procedures of the company that owns the data.
	Categorical	Modifying missing values in categorical variables by *MISSING,* and it means to create a new category for missing values.
Outliers	Numerical	No treatment is needed.
	Categorical	Change infrequent categories (frequency distribution <5%) by *OTHERS.* It needs an analysis of categorical variables distribution for detecting categories that appear in too few customers and it is attached in the annexes section 9.2.
Indexed categories	Categorical	Some algorithms require only numerical input data. For this reason, the non-numerical variables should be mapped with an indexer (0, 1, 2, ...) in both training and test sets, with always the same mapping for avoiding different indexers by categories in different datasets. Although in this thesis, with the implemented *Spark R* functions it is not necessary.

Table 3: List of data pre-processing required treatments.

Note: When predicting a fresh data set, it means a different population sample than the training data, it is necessary to do the same pre-processing than in the training data as well as modify new categories of categorical features (that were not in the training dataset) by *OTHERS.* Thus, the exact final mapping of categories used for training should be saved.

4.3. Target variable and possible applications of the model

The target variable is *Defaulted.* Let $y_i = \{0,1\}$ be an observation of the outcome random variable Y_i which provides information about the client behaviour. If the customer has defaulted, the observed value is equal to 1 and when has not is equal to 0. Then, the target variable *Defaulted* is binary and for this reason, it is a *machine learning* classification problem with the objective of predicting the probability of default.

Going back to the risk sector introduction, it would be useful to explain possible implementations of the algorithms in a credit company. Remember that the key is basically selecting well the clients that will be financed after the documentation check (for avoiding marketing costs in pre-accepting clients that will not be financed and then with no profit) and the ones that will have good risk metrics (for avoiding losing money with miss payments).

Once it is said, the possible applications of the models are:

- Use the model as a credit score for punctuating the customers in terms of risk, and then rejecting those with too much probability of default and offering a better price to those with the lowest probability.
- Use the model as a rejection rule, for example, directly rejecting and not pre-accepting all the applications with a predicted probability higher than a determined threshold.
- Use the model in a marketing campaign to dedicating the budget to those clients with the lowest probability of default.
- Use the model for analysing and discovering hidden characteristics of customers with a low probability of default and then, offering them a better financial product (not only in terms of lower interest rates, it could be a product designed for these profiles).

4.4. Predictive performance measures

This section will explain the predictive performance measures that will be used for choosing and comparing the models. The next section will explain the protocol of model validation, it means how the whole training process works and which performance measures are calculated in every phase for deciding and comparing between models. Thus, it is important to understand all the metrics before mentioning them in the next section.

The project presents 2 performance measures in the second protocol phase and 5 more in the third (phases explained in the next section 4.5). These metrics let us compare and assess the power for predicting the default probability between models.

4.4.1. *Measures with cut-off needed*

As it has been told, the algorithms estimate the probability of default p_i that allow to define the predicted class by comparing p_i with different cut-offs $c \in [0,1]$. Clarifying with an example, if a customer has a default probability of 0.45, he will be classified as a defaulter if the cut-off is lower or equal to 0.45 but as a non-defaulter, if the cut-off is greater to 0.45.

Then, having an entire dataset punctuated by a model and comparing a concrete cut-off with the predicted probabilities, it provides a confusion matrix like this:

		Predicted	
		Not Default	Default
Real	Not Default	True negative (TN)	False positive (FP)
	Default	False negative (FN)	True positive (TP)

Table 4: Confusion matrix structure.

Once we have a confusion matrix, as a result of comparing a vector of predicted probabilities with a defined cut-off, these metrics will be calculated:

- Sensitivity (true positive over real positive): $\frac{TP}{TP+FN}$
- Specificity (true negative over real negative): $\frac{TN}{TN+FP}$
- Accuracy: $\frac{TP+TN}{TN+FN+TP+FP}$

However, looking for the optimal cut-offs of these performance measures, as the dataset is unbalanced (only 20% of defaulters in the response variable), it is easy to find that:

- For sensitivity, the optimal cut-off is approximately a probability of 0 which means predicting every client as a defaulter, then all the defaulters are well classified.
- For specificity, the optimal cut-off is approximately a probability of 1 which means predicting every client as a non-defaulter, then all the non-defaulters are well classified.

For avoiding this situation, the final metrics with cut-off needed that will be used in the protocol of model validation will be the accuracy and the sum of sensitivity and specificity.

4.4.2. *Measures without cut-off needed*

Apart from the mentioned metrics, other metrics are needed for comparing models without establishing any cut-off. It is important because, for example, if we use the explained measures with the cut-off 0.5 it makes no sense comparing a model with a range of predicted probabilities from 0.1 to 0.4 with other from 0.1 to 0.9.

These are the metrics that will be calculated once we have a vector of predicted probabilities by a model:

a. *Percentage of true positive by quartiles of predicted probabilities vectors.* It is a measure without cut-off needed that shows the distribution of real positive in the dataset by groups of 25% from lowest to highest predicted probability of default. In short, the process for calculating it is: ordering a set of observations by default predicted probability, selecting the customers that leave 25%, 50% and 75% of cases below, using their probabilities as the cut-off for every quartile, calculate the % of true defaulters in every group. Logically, the group of 25% observations with the lowest predicted probabilities of default would have to contain fewer defaulters than the group with highest default probabilities.

b. *Area under the curve (AUC) of the receiver operating characteristics curves (ROC).* It is a well-known measure to evaluate the discriminative power of models in binary classification problems. The process of calculating it is simple:

   1. Comparing a vector of predicted probabilities with a concrete cut-off, get the corresponding confusion matrix explained above and obtain the specificity and sensitivity for this concrete cut-off.

2. Repeat the step 1 for all possible cut-offs from 0 to 1.

3. Plot the ROC curve where every point corresponds to a concrete cut-off with its pair of sensitivity and specificity. As you can see in the graphic below, the y-axis is the sensitivity and the x-axis is the complement of specificity equals the false positive over real negative rate.

4. Calculate the area under the curve or AUC metric.

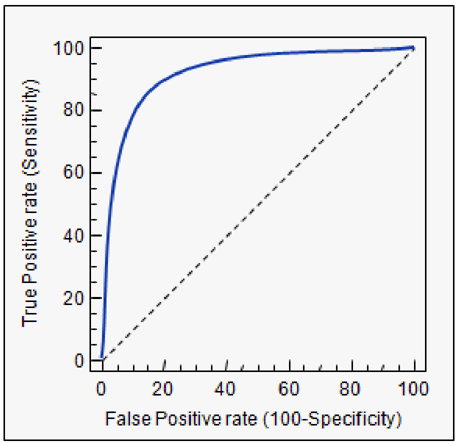

Figure 5: Example of ROC curve and AUC.

The area under the diagonal measures 0.5 and it is associated to a random classifier, so the perfect model is AUC equals to 1. It gives an idea of the predictive robustness of models because, without any determined cut-off, we have a metric that tells us: how robust are the predicted probabilities by the model with a wide range of cut-offs.

4.5. Phases of the protocol of model validation

As it has been told in the point 4.1, the predictive analytics projects need a meticulous methodology. Basically, apart from loading and pre-processing the data, as it has been explained, it is necessary to find the best parametrization of every model and for doing it, the key is using a consistent and impartial protocol of validation when comparing between models.

For example, a typical example of a wrong method is to compare only the performance of models with the predicted probabilities of the training dataset, because here the model with overfitting[8] will win and, in the future, its behaviour predicting fresh datasets could be worst.

For this reason, this section explains the global picture of models training and protocol of validation used in this thesis, step by step. These are the four germane phases:

[8] The concept of good predictive performance in the training dataset and poor in the test set because of over adjusting the complexity of a model. It is a typical situation that cross-validation and other techniques try to solve.

1. *Create training and test sets as well as the training folds for cross-validation.* Once the data is appropriately pre-processed, it will be divided into two groups (training and test) in a ratio of 75%-25% respectively. The training set will be used for finding the best parametrizations of every model (phase 2). The test set will be used for training the models with its optimal parametrization (phase 3) and comparing the results with all the performance measures (phase 4) applied to both training and test datasets. Then, it is important to check that the two partitions have the same possible categories by categorical variables for avoiding problems when predicting the test set with the models.

2. *Find the best parametrization of every model with the training set.* This project will use 5-Fold Cross-Validation (CV) with the performance metrics: out of bag AUC and out of bag % of true positive by quartiles[9]. Out of bag (OOB) means to use the prediction of observations not included in the training dataset. What does all that mean? In short, the training dataset will be separate in 5 sets and for every fold, the other 4 folds will be used as the training dataset for training every model parametrization. Then, with the predictions in the test set or the fold in question, we will have the out of bag predictions. Finally, the 5 OOB vectors of predicted default probabilities by fold will be saved in a global vector (having all the training dataset punctuated with OOB predictions for every model parametrization) and then, with this entire vector, the global AUC and % of true defaulters by quartiles will be calculated.

3. *Train models with optimal parametrizations.* Once the best parametrization of every model is chosen, we re-train every final model with the whole training dataset. Now, with the trained model we will predict the test dataset for having out of bag predictions as well as the training predictions. Note that it is different from the global vector in the section before (5 OOB vectors of predicted default probabilities) because now we have only one model trained with all the training set and before, we were talking about five models trained with a different 80% of the training set every time. Finally, having all the training and test sets predicted for every model with its optimal parametrization, several performance metrics will be calculated (with the optimal cut-off for everyone) for both training and test sets.

4. *Compare the models with performance measures.* Finally, the protocol of validation is concluded and all the explained performance metrics are calculated. Then it is time for comparing the results and finding positive and negative aspects of the models.

   In this step, apart from all the performance measures, the *unsupervised machine learning* technique called *principal components analysis* (*PCA*) will be used for understanding the correlations between the predicted probabilities of default and the explanatory variables. It will complement the thesis by using an *unsupervised* technique.

[9] It means ordering the vector of predicted probabilities and doing groups of 25% from lowest to highest probability as explained in the previous section.

4.6. Results of the protocol of model validation phase 2

This section contains the results table of phase 2 by models and descending order according to the global OOB AUC. Then, the elected optimal parametrization will be the first row except if the other measure, distribution of true positive by quartiles, is worst than other candidate models.

4.6.1. *Logistic regression*

In the case of the model *logistic regression*, the candidate parametrizations of the model are only numerical features as explanatory variables for predicting the target, only categorical features, and both numerical and categorical features. The aim is trying different model combinations to check if removing variables we can achieve similar performance results, however, it is not so important as the germane objective is predicting and not interpreting the results.

These are the results for the candidate parametrizations of the model (ordered by global OOB AUC):

	AUC BY CV	% OF TRUE POSITIVE BY QUARTILES			
PARAMETRIZATION OF MODEL: LR		Q1	Q2	Q3	Q4
All features	0.5733	0.1446	0.1900	0.2181	0.2488
Numerical features	0.5697	0.1446	0.1978	0.2118	0.2473
Categorical features	0.5655	0.1400	0.2103	0.2103	0.2411

Table 5: Results of logistic regression in the protocol of model validation phase 2.

In this case, there are not too many differences between the three parametrizations but the optimal is using all features because it has the highest global AUC and a coherent distribution of true positive by quartiles. It seems that the numerical and categorical features provide the same information in terms of predictability, but once they are used together the model does not improve so much. However, the optimal model will be with all features.

4.6.2. *Decision tree*

In the case of the model *decision tree*, the candidate parametrizations of the model (all with both numerical and categorical features) are all the possible combinations of these parameters:

- Maximum number of bins for discretizing, at least the maximum number of categories for any categorical feature (max_bins): 10, 20, 30.
- Maximum number of nodes separating any leaves from the root of the tree (max_depth): 5, 10, 15.
- Minimum number of instances each child must have after the split (min_instance_node): 5, 9, 13.

These are the results for the best candidate parametrizations of the model (ordered by global OOB AUC):

	AUC BY CV	% OF TRUE POSITIVE BY QUARTILES			
PARAMETRIZATION OF MODEL: DT		Q1	Q2	Q3	Q4
max_bin=30;max_depth=5;min_instance=5	0.5601	0.1562	0.1831	0.2332	0.2321
max_bin=30;max_depth=5;min_instance=13	0.5574	0.1674	0.1750	0.2226	0.2369
max_bin=30;max_depth=5;min_instance=9	0.5545	0.1581	0.1804	0.2411	0.2226
max_bin=10;max_depth=5;min_instance=5	0.5451	0.1572	0.2083	0.2255	0.2120
max_bin=30;max_depth=10;min_instance=5	0.5429	0.1697	0.2013	0.2006	0.2387
max_bin=20;max_depth=5;min_instance=5	0.5415	0.1787	0.1714	0.2319	0.2241
max_bin=20;max_depth=5;min_instance=9	0.5411	0.1774	0.1711	0.2349	0.2224
max_bin=10;max_depth=5;min_instance=13	0.5408	0.1602	0.2061	0.2269	0.2076
max_bin=10;max_depth=5;min_instance=9	0.5402	0.1586	0.2114	0.2237	0.2084
max_bin=30;max_depth=10;min_instance=9	0.5362	0.1607	0.2122	0.2047	0.2259
…	…	…	…	…	…

Table 6: Results of the decision tree in the protocol of model validation phase 2.

Although there is not a clear pattern in the parameters, the parametrization with max_bins equal to 30 and max_depth equal to 5 have the highest global AUC. In this case, various candidates could give similar models in terms of predictive robustness but the first option will be chosen.

4.6.3. *Random forest*

In the case of the model *random forest*, the candidate parametrizations of the model (all with both numerical and categorical features) are all the possible combinations of these parameters:

- Maximum number of bins for discretizing, at least the maximum number of categories for any categorical feature (max_bins): 10, 20, 30.
- Maximum number of nodes separating any leaves from the root of the tree (max_depth): 5, 10, 15.
- Number of trees to train (num.tree): 15, 30, 45.
- Minimum number of instances each child must have after the split (min_instance_per_node): 5, 9, 13.
- The fraction of the training data used for learning each *decision tree* (subsampling_rate): 1.

These are the results for the best candidate parametrizations of the model (ordered by global OOB AUC):

	AUC BY CV	% OF TRUE POSITIVE BY QUARTILES			
PARAMETRIZATION OF MODEL: RF		Q1	Q2	Q3	Q4
max_bins=30;max_depth=5;num.trees=30;	0.5904	0.1415	0.1760	0.2134	0.2706

min_instances_per_node=9					
max_bins=30;max_depth=15;num.trees=30; min_instances_per_node=9	0.5887	0.1322	0.1776	0.2352	0.2566
max_bins=30;max_depth=5;num.trees=15; min_instances_per_node=13	0.5886	0.1353	0.1822	0.2181	0.2659
max_bins=20;max_depth=5;num.trees=30; min_instances_per_node=5	0.5868	0.1369	0.1838	0.2243	0.2566
...	...	...	...	...	...

Table 7: Results of random forest in the protocol of model validation phase 2.

It seems that the models with more number of trees and higher maximum number of bins are the best in terms of predictive power. The maximum depth and number of instances in each child do not seem to be as important in this case. Then, the parametrization chosen as optimal is the first one with the highest global AUC and the best trend of true defaulters by probabilities quartiles.

4.6.4. *Gradient boosted trees*

In the case of the model *gradient boosted trees*, the candidate parametrizations of the model (all with both numerical and categorical features) are all the possible combinations of these parameters:

- Maximum number of bins for discretizing, at least the maximum number of categories for any categorical feature (*max_bins*): 10, 20, 30.
- Maximum number of nodes separating any leaves from the root of the tree (*max_depth*): 5, 10, 15.
- Maximum number of iterations (*max_iter*): 15, 30, 45.
- Step size to be used for each iteration of optimization (*step_size*): 0.05, 0.1, 0.15.
- Minimum number of instances each child must have after the split (*min_instance_node*): 1.
- The fraction of the training data used for learning each *decision tree* (*subsampling_rate*): 1.

These are the results for the best candidate parametrizations of the model (ordered by global OOB AUC):

	AUC BY CV	% OF TRUE POSITIVE BY QUARTILES			
PARAMETRIZATION OF MODEL: GBT		Q1	Q2	Q3	Q4
max_depth=10;max_iter=15;step_size=0.1	0.5714	0.1462	0.1822	0.2150	0.2582
max_depth=10;max_iter=30;step_size=0.1	0.5650	0.1602	0.1776	0.2134	0.2504
max_depth=10;max_iter=30;step_size=0.05	0.5618	0.1477	0.1900	0.2290	0.2348
max_depth=15;max_iter=45;step_size=0.05	0.5614	0.1462	0.1869	0.2430	0.2255
max_depth=15;max_iter=30;step_size=0.1	0.5611	0.1524	0.1604	0.2539	0.2348
max_depth=15;max_iter=45;step_size=0.1	0.5606	0.1462	0.1791	0.2414	0.2348
max_depth=10;max_iter=15;step_size=0.05	0.5604	0.1446	0.2040	0.2212	0.2317

max_depth=10;max_iter=45;step_size=0.1	0.5598	0.1571	0.1838	0.2165	0.2442
...	...	...	...	...	...

Table 8: Results of gradient boosted trees in the protocol of model validation phase 2.

The best combination of parameters is an intermediate maximal trees depth and a lower step size with more iterations or a higher step size with fewer iterations. The results are quite similar but the chosen parametrization is the first with the highest AUC and best true positive distribution.

4.6.5. *Naive Bayes*

In the case of the model naive Bayes, the candidate parametrizations of the model are:

- Only numerical features as explanatory variables for predicting the target.
- Only categorical features as explanatory variables for predicting the target.
- Both numerical and categorical features as explanatory variables for predicting the target.

These are the results for the candidate parametrizations of the model (ordered by global OOB AUC):

	AUC BY CV	% OF TRUE POSITIVE BY QUARTILES			
PARAMETRIZATION OF MODEL: NB		Q1	Q2	Q3	Q4
Categorical features	0.5621	0.1711	0.1838	0.1931	0.2535
All features	0.5305	0.1649	0.2134	0.2117	0
Numerical features	0.5304	0.1649	0.2134	0.2117	0

Table 9: Results of naive Bayes in the protocol of model validation phase 2.

In this case, the differences are quite relevant between the three parametrizations. The optimal is using only categorical features and although the others do not have a too low global AUC, they have 0 true defaulters in the fourth quartile which makes no sense.

4.6.6. *Multilayer perceptron classifier*

In the case of the model *multilayer perceptron classifier*, the candidate parametrizations of the model (all with both numerical and categorical features) are all combinations of these parameters:

- Number of nodes in input layer: number of numerical features plus the number of unique categories less one per categorical feature.
- Number of nodes in output layer: 2 equals to the number of classes in the target variable.

- Number of nodes in the first hidden layer: 6, 8, 10, 12.
- Number of nodes in the second hidden layer: 4, 6, 8, 10.
- The rest of parameters (initial weights for weights initialization, step size, maximum iteration number, convergence tolerance of iterations) will be set by default in the implemented function.

These are the results for the ten best candidate parametrizations of the model (ordered by global OOB AUC):

	AUC BY CV	% OF TRUE POSITIVE BY QUARTILES			
PARAMETRIZATION OF MODEL: NN		Q1	Q2	Q3	Q4
hidden_layer_1=12;hidden_layer_2=4	0.5359	0.1757	0.1905	0.1982	0.2379
hidden_layer_1=8;hidden_layer_2=10	0.5284	0.1891	0.1720	0.2434	0.2093
hidden_layer_1=12;hidden_layer_2=10	0.5281	0.1890	0.1813	0.1930	0.2358
hidden_layer_1=10;hidden_layer_2=10	0.5279	0.1731	0.1975	0.2339	0.2162
hidden_layer_1=6;hidden_layer_2=10	0.5168	0.1815	0.2052	0.2200	0.1941
hidden_layer_1=8;hidden_layer_2=6	0.5156	0.1920	0.1953	0.2039	0.2123
hidden_layer_1=6;hidden_layer_2=8	0.5061	0.1968	0.2008	0.1994	0.2059
hidden_layer_1=8;hidden_layer_2=4	0.5029	0.1949	0.1977	0.2220	0.1895
hidden_layer_1=6;hidden_layer_2=4	0.4997	0.2049	0.1956	0.2003	0.2009
hidden_layer_1=10;hidden_layer_2=8	0.4973	0.2059	0.1944	0.2118	0.1821
...	...	...	...	...	...

Table 10: Results of multilayer perceptron in the protocol of model validation phase 2.

None of these neural network models is better than other model candidates, but the combination of 12 nodes in the first hidden layer and 4 in the second corresponds to the *multilayer perceptron classifier* model with the better true positive distribution and global AUC.

4.7. Results of the protocol of model validation phase 3

This section works with the optimal parametrization of every model trained with the whole training set. Then, punctuating both training and test sets with these final models, the tables below show all the explained performance measures for both datasets. In the next section, the results will be commented and the models compared for deciding if one has performed better.

4.7.1. *Training set*

		LR	DT	RF	GBT	NB	NN
CRITERIA	Sensitivity + Specificity	1.2091	1.2496	1.3796	1.9791	1.1574	1.1062
	Accuracy	0.8004	0.8058	0.8183	0.9938	0.8000	0.5514
	AUC	0.6500	0.6807	0.7639	0.9988	0.6096	0.5494
	% True positive in Q1	0.0886	0.0824	0.0404	0.0000	0.1322	0.1650
	% True positive in Q2	0.1807	0.2021	0.1168	0.0000	0.1838	0.1793
	% True positive in Q3	0.2321	0.2348	0.2259	0.0062	0.2056	0.2348
	% True positive in Q4	0.3002	0.3425	0.4184	0.7947	0.2799	0
CUT-OFF	Sensitivity + Specificity	0.2000	0.1900	0.2000	0.3100	0.2200	0.2300
	Accuracy	0.4600	0.4200	0.2800	0.3100	0.4100	0.2500
	% True positive in Q1	0.1385	0.1288	0.1716	0.0631	0.1558	0.1608
	% True positive in Q2	0.1957	0.2281	0.1978	0.0936	0.1990	0.2058
	% True positive in Q3	0.2540	0.2349	0.2292	0.1983	0.2503	0.2509
	% True positive in Q4	0.5355	0.8889	0.4472	0.9679	0.4452	0.2509

Table 11: Results of the training set in the protocol of model validation phase 3.

4.7.2. *Test set*

		LR	DT	RF	GBT	NB	NN
CRITERIA	Sensitivity + Specificity	1.1329	1.1353	1.1302	1.1051	1.0871	1.0196
	Accuracy	0.7895	0.7895	0.7895	0.7884	0.7895	0.5323
	AUC	0.5838	0.5731	0.5905	0.5612	0.5552	0.5175
	% True positive in Q1	0.1511	0.1471	0.1556	0.1511	0.1733	0.1928
	% True positive in Q2	0.1964	0.2135	0.1786	0.2009	0.2098	0.2813
	% True positive in Q3	0.2321	0.2531	0.2277	0.2500	0.2009	0.2180
	% True positive in Q4	0.2667	0.2463	0.2844	0.2444	0.2622	0
CUT-OFF	Sensitivity + Specificity	0.1900	0.1900	0.2200	0.1500	0.2200	0.2000
	Accuracy	0.4500	0.7600	0.3100	0.9500	0.4300	0.2500
	% True positive in Q1	0.1446	0.1288	0.1713	0.0803	0.1595	0.1608
	% True positive in Q2	0.1991	0.2281	0.1969	0.1413	0.2039	0.2058
	% True positive in Q3	0.2561	0.2349	0.2279	0.2670	0.2523	0.2509
	% True positive in Q4	0.5398	0.8889	0.4270	0.9778	0.4742	0.2509

Table 12: Results of the test set in the protocol of model validation phase 3.

For example in table 12, for the *logistic regression,* the value 0.2561 is the default probability of the customer that leaves a 75% of customers below in the test set with respect to their predicted probabilities, and the value 0.1991 is the probability of the customer that leaves a 50%. Then, 23.21% is the percentage of defaulters with a predicted probability between 0.1991 and 0.2561, statistically the third quartile. Also, with the same example of table 12, the optimal accuracy of the *logistic regression* is 0.7895 and it is obtained with the cut-off of 0.45.

4.8. Results of the protocol of model validation phase 4

According to the results table of the training dataset, there is a clear winner: *gradient boosted trees*. It has the higher values for the metrics AUC, accuracy and the sum sensitivity and specificity. Furthermore, it performs the best distribution of true defaulters having 0 defaulters in the quartile with the lowest probabilities and an 80% of defaulters in the highest. Additionally, looking at the cut-offs of the true defaulters' distribution, we can see that the range of probabilities is wider in this model, from 0.08 in the first quartile to 0.96 in the fourth.

That said, this model could be a clear case of over-fitting if in the test set it gets worst results. For this reason, it is important to comment that the other models, except the *multilayer perceptron classifier* and concretely the *random forest*, have very reasonable performance measures with maybe no over-fitting.

According to the results table of the test set, now the model *gradient boosted trees* is not the best one. The *logistic regression* and the *random forest* have the best AUC and very similar values of accuracy and sum of sensitivity and specificity compared with the *logistic regression* and the *decision tree*. Also, they have the best trends of true defaulters by quartiles.

Looking at the cut-off, all models have similar optimal cut-offs between the training and test sets performance measures, but the accuracy cut-offs for the *decision tree* and the *gradient boosted trees* are quite different and it suggests that they are not as solid models as others for predicting fresh datasets. However, the *logistic regression* and *random forest* seem to be really robust comparing the figures of the two tables and although there is not a clear winner, these models appear to be the more appropriate between all candidates.

Apart from that, it is interesting that the defaulters' correlation with the *gradient boosted trees* model in the training dataset is now deprecated in the test. Furthermore, it is curious that the neural network *multilayer perceptron classifier* is the worst model in this real case approach. Although neural networks are considered the best models in terms of predictive power for many situations, in this case, it seems that the low amount of data is not appropriate for this kind of complex algorithms which require large amounts of data to beat the others.

4.8.1. *Analysis of the correlation between features and outcome predictions*

This section will try to study a point that has not been mentioned until now with an *unsupervised machine learning* technique called *principal components analysis*. The aim is to study the correlations between numerical features, it means in which directions they move according to each other, and also research which profiles of customers have a higher predicted probability of default by models. This *machine learning* technique can simplify this task because these models are quite complex and the high number of interactions make really difficult to understand why a customer is considered as a defaulter or not.

Only numerical features will be considered, but as we have seen in the *logistic regression* model, the categorical features do not seem to add too much information to the numerical ones (or vice versa). For this reason, this section will simply make the analysis with the numerical. First of all, let see the results of the *PCA* (explained in the theory section) by observing the first two principal components. In essence, it means studying how every variable affects the new space of artificial variables created by the *PCA* because every new artificial variable or principal component is a linear combination of the original features. This is the graphic:

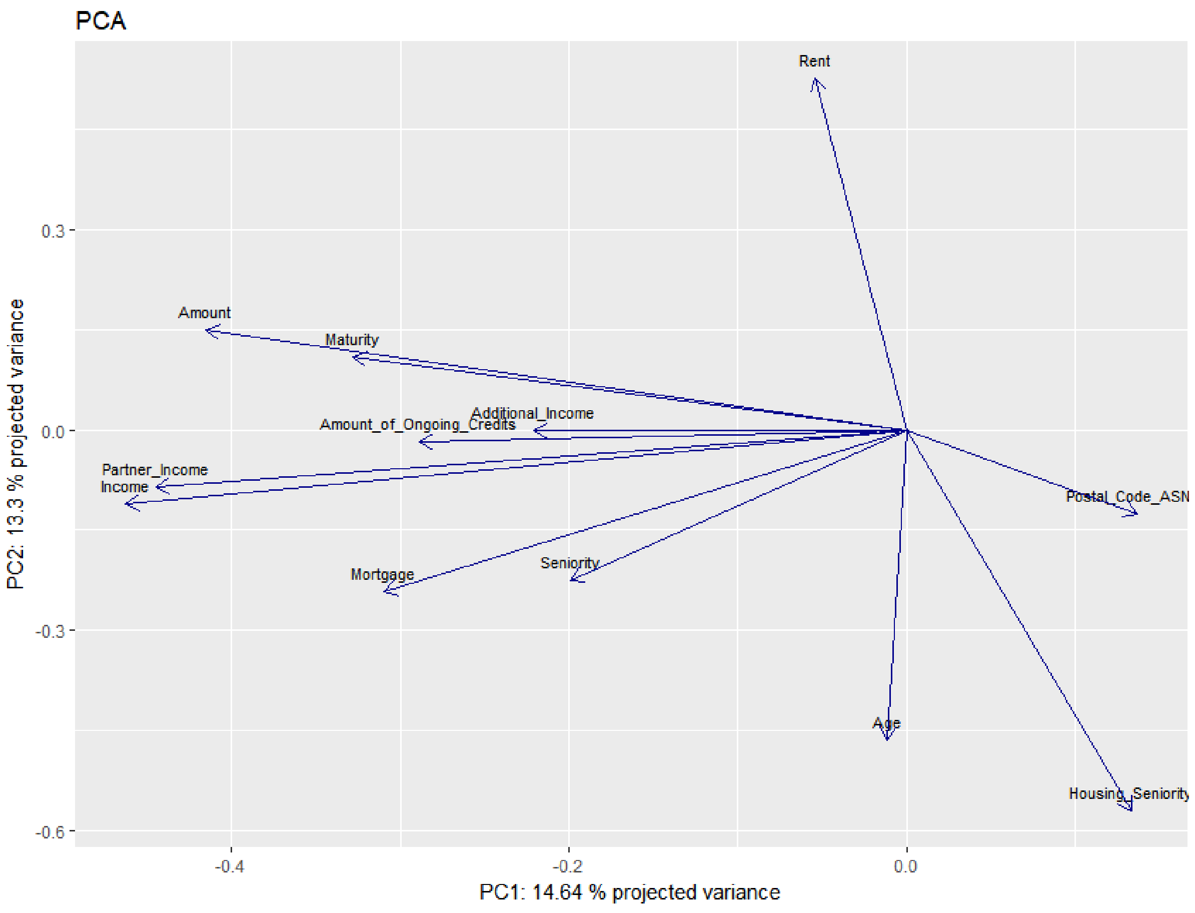

Figure 6: Plot of the two first principal components of the PCA.

From this graphic of the first two principal components, it is possible to extract logical customer patterns that will help to understand why the models predict lower or higher probability of default. These are the main patterns:

a. The more income and partner income that customers have, the more loan amount requested, maturity, amount of ongoing credits, mortgage, seniority, and the less probability of being in ASNEF by postal code which means that customers live in zones with fewer people in negative databases of defaulters.

b. Other characteristics, not too much correlated with the explained above, are that the more cost of rent that customers have, the less age and housing seniority. It completely makes sense in terms of social behaviour.

That said, we will project all the customers in this new artificial two principal components with optimal models and the points will be coloured according to the predicted probability of default. These are the resulting graphics by models:

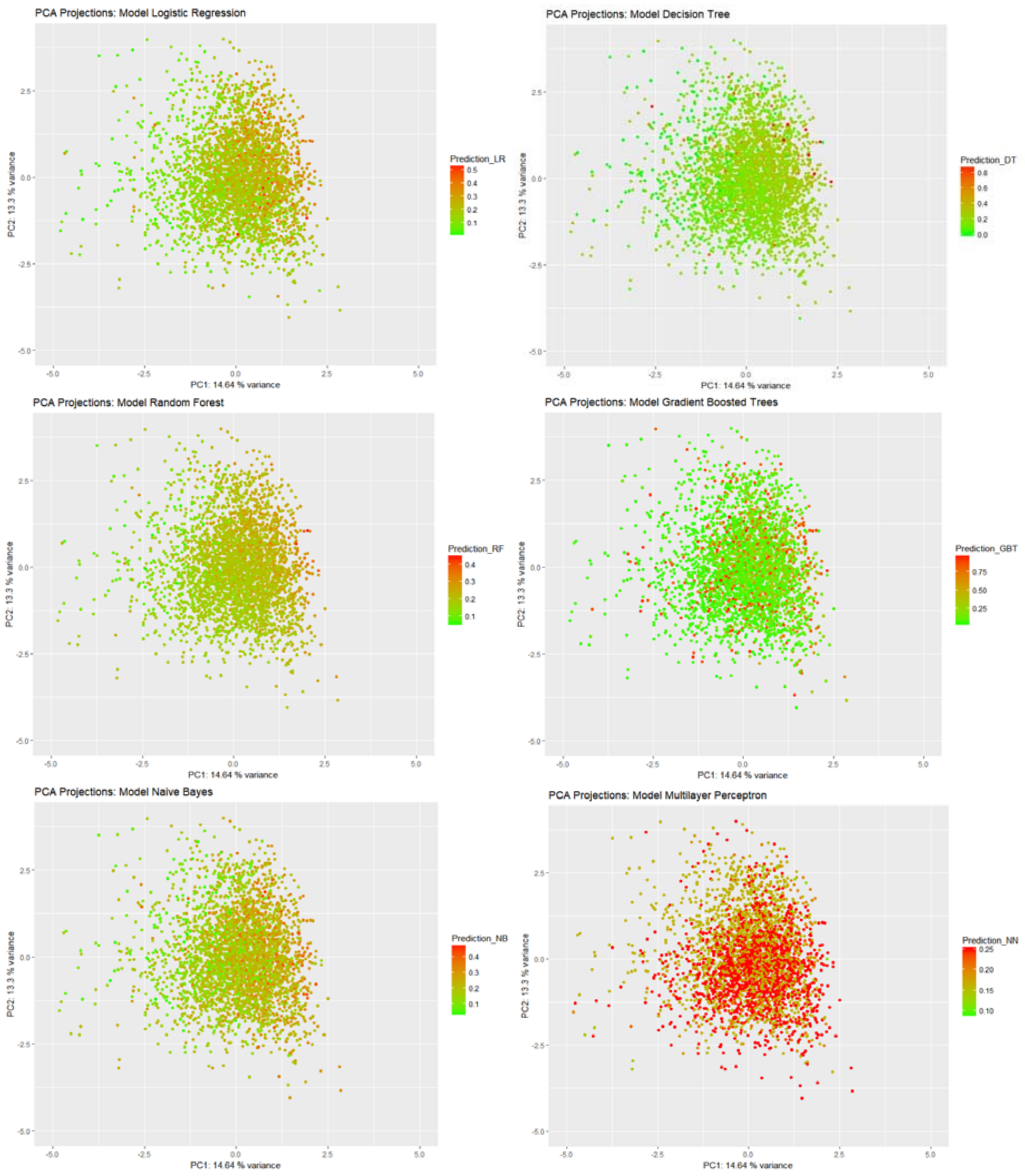

Figure 7: Projections of customers in the two first principal components of the PCA.

Ignoring the *multilayer perceptron classifier* neural network, because of we cannot see any clear pattern, the projections of the customers into the two first principal components show us that in general, the profile of a defaulter is a customer that:

- Has a residence in a postal code with a high probability of being in ASNEF.
- Has low income, partner income, mortgage, seniority and amount of ongoing credits.
- Does not have a concrete age, monthly rent cost and housing seniority.

Nevertheless, as we can see in the graphics, these principal components have projected only a 30% of the global variance, and it means that the described profile of defaulters does not fit for every customer.

Apart from this analysis of the correlation between features and outcome predictions by the models, in the annexes, you can find the models features importance for the models: *logistic regression, decision tree* and *random forest.* There, you will find the estimated coefficients by the *logistic regression* ordered by absolute coefficient value, and the variable importance for splitting the trees in the *decision tree* and *random forest.* In general, a higher value means that this variable is more relevant in the model in terms of discriminating the response.

5. CONCLUSIONS

This is the last section of the thesis and it contains the conclusions after developing the whole project.

5.1. Hypothesis results

According to the hypothesis of the theoretical section, it is confirmed that although black-box models such as neural networks or tree-based models are more complex than, for example, logistic regression, they follow algorithmic theories that are not as different or advanced in terms of theory comprehension. The logic behind the building process of these models is not so complicated but a large number of iterations when optimizing the models makes them a black-box in terms of interpretability.

On the other hand, the hypothesis of the real case approach has been partially accomplished. It was to be expected that more complex models, such as the neural network, would perform better in terms of predictive power. However, what we have seen is that the simplest model, the *logistic regression,* has performed as the best in the test set. That said, it is important to note that the hypothesis considered the possibility that the dataset dimension influences the results. It means that as the data had only 3468 observations, it has not been enough data for the more complex algorithm which are the best in a competition such as *Kaggle*.

According to the hypothesis that the coding language *Spark* would be so time-consuming, after the development of the entire project, it is possible to confirm that the *Big Data* engine *Spark* has a really user-friendly framework and integration with well-known statistical coding languages such as *R* or *Python*. However, for the author, it has been the first time in doing data science and *machine learning* with *Spark* and a lot of time has been spent on learning it.

5.2. Aims achieved

The principal aim of this thesis was to create a guide (with theory and practice) to develop a predictive analytics project from scratch with the *Big Data* tool *Spark*. Apart from that, a key point was that all kind of users, experts or not, could understand the thesis even the more theoretical concepts. Once the thesis is done, it can be said that from lots of resources and references that can be found in books or on the internet, this thesis provides a hybrid guide for *machine learning* with succinct theory and practice aimed to different audiences.

Furthermore, it was mentioned at the beginning that the Spanish banking regulator only allows using the *logistic regression* model and that laws have to progress and be adapted to all these new available and widely used algorithms. After seeing the results, the wrong way of interpreting the results is that *logistic regression* performs well enough and for this reason, it would be not necessary that banking regulators attempt to be more flexible. Why? In many fields, neural networks and advanced tree-based are states of the art and the financial sector need be no different, provided that enough data is available.

5.3. Possible thesis extension

In *machine learning* and *artificial intelligence*, finding the optimal parameters for a model is the key point. Thus, the first important extension of the thesis would be looking for new models' parametrizations that improve the predictive performance as well as the creation of new artificial variable from the original data for discriminating better the defaults.

Apart from that, it would be really interesting to analyse the business impact of the models. In other words, researching how applying these *machine learning* will impact a credit company in economic terms. After all, the new era of artificial intelligence is beginning and it is clear that it will create a strong impact in the society.

6. REFERENCES

[1] Hastie, Trevor; Tibshirani, Robert; James, Gareth; Witten, Daniela. 2013. *An introduction to statistical learning.* New York, USA. Springer.

[2] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome. 2008. *The elements of statistical learning.* California, USA. Springer.

[3] Hotelling, H. 1993. *Analysis of a complex of statistical variables into principal components.* Journal of Educational Psychology, 24, 417-441, and 498-520.

[4] Cox, D.R. 1958. *The regression analysis of binary sequences.* Journal of the Royal Statistical Society: series B, 20, 215-242.

[5] Hand, David. 1997. *Construction and assessment of classification* rules. Wiley, USA. Chichester.

[6] H. Witten, Ian. 2011. *Data mining: practical machine learning tools and techniques with Java implementations.* Third edition. San Francisco, USA. Morgan Kaufmann.

[7] J. Mester, Loretta. 2015. *What's the point of credit scoring?* [pdf] Federal Reserve Bank of Philadelphia. Available at: http://r-es.org/9jornadasR/pdf/9JUR_paper_2.pdf.

[8] Philadelphia Media Network (Digital), 2008. *History of credit scores.* Philly, [online] 8 May. Available at: http://www.philly.com/philly/business/cars/research/general_cars/General_History_of_Credit_Scores.html.

[9] Pell, Nicholas. 2015. *A secret history of credit scores.* TheStreet, [online] 13 April. Available at: https://www.thestreet.com/story/13097739/1/a-secret-history-of-credit-scores-who-determined-what-matters-and-why.html.

[10] J. Parra, Manuel. 2017. *Taller procesamiento en Big Data con Spark R.* [pdf] University of Granda. Available at: http://r-es.org/9jornadasR/pdf/9JUR_paper_2.pdf.

[11] Bradley, Joseph; Meng, Xiangrui; Lee, Denny. 2016. *Why you should use Spark for machine learning.* InfoWorld, [online] 11 February. Available at: https://www.infoworld.com/article/3031690/analytics/why-you-should-use-spark-for-machine-learning.html.

[12] R Studio. *Sparklyr from R Studio.* [online] Available at: https://spark.rstudio.com.

[13] The Apache Spark Software Foundation. *Apache Spark, lightning-fast unified analytics engine.* [online] Available at: https://spark.apache.org.

[14] DataCamp. *RDocumentation*. [online] Available at: https://www.rdocumentation.org.

[15] GitHub. *GitHub platform*. [online] Available at: https://github.com.

[16] R-bloggers. *R news and tutorials by R bloggers*. [online] Available at: https://www.r-bloggers.com.

[17] Stack Overflow. *Stack Overflow online community*. [online] Available at: https://stackoverflow.com.

[18] Wikipedia. *Wikipedia, the free encyclopedia*. [online] Available at: https://en.wikipedia.org.

[19] The R Foundation, 2018. *R 3.5.0 version (Joy in Playing)*. [computer program] Available at: https://www.r-project.org.

[20] RStudio, 2018. *RStudio 1.1.453 version*. [computer program] Available at: https://www.rstudio.com.

7. INDEX OF FIGURES AND TABLES

7.1. Figures

7.2. Tables

8. ANNEXES

8.1. Metadata

Variable	Type for models	Number of categories	Number if missing
Contract_ID	Categorical	-	0
Defaulted	Categorical	2	0
Application_Hour_Group	Categorical	3	0
Application_Week_Day	Categorical	7	0
Amount	Numerical	-	0
Maturity	Numerical	-	0
Purpose	Categorical	13	0
Province	Categorical	50	0
Postal_Code_ASNEF	Numerical	-	12
Age	Numerical	-	0
Gender	Categorical	2	0
Profession_Code	Categorical	32	53
Profession_Sector	Categorical	2	0
Contract_Type	Categorical	3	0
Seniority	Numerical	-	0
Housing_Type	Categorical	5	0
Housing_Seniority	Numerical	-	0
Marital_Status	Categorical	6	0
People_in_Household	Categorical	7	0
Income	Numerical	-	0
Additional_Income	Numerical	-	2290
Partner_Income	Numerical	-	1828
Rent	Numerical	-	2566
Mortgage	Numerical	-	1984
Amount_of_Ongoing_Credits	Numerical	-	0
Num_Ongoing_Credits	Categorical	6	0

8.2. Analysis of categorical variables distribution

Variable	Category	Distribution
Defaulted	0	79.67%
	1	20.33%
Application_Hour_Group	[7H, 20H)	85.55%
	[20H, 23H)	8.42%
	[23H, 7H)	6.03%
Application_Week_Day	1	19.20%
	3	19.00%
	2	18.66%
	4	17.16%
	5	13.93%
	6	6.31%
	7	5.74%
Purpose	HOMEIMPROVEMENT	28.98%
	LIQUIDITY	21.80%
	USEDCAR	8.77%
	DEBTS	7.93%
	MEDICALCARE	6.46%
	VACATION	6.26%
	FURNITURE_AND_APPLIANCES	5.82%
	NEWCAR	4.33%
	TRAINING	3.86%
	WEDDINGS	2.62%
	HITECH	1.27%
	MOTO	0.95%
	RELOCATION	0.95%
Gender	MALE	65.43%
	FEMALE	34.57%
Profession_Sector	PRIVATE_SECTOR	79.67%
	PUBLIC_SECTOR	20.33%
Contract_Type	PERMANENT	81.72%
	PENSION	16.75%
	INDEPENDENT	1.53%
Housing_Type	HOME_OWNERSHIP_WITH_MORTGAGE	36.85%
	HOME_OWNERSHIP_WITHOUT_MORTGAGE	22.49%
	THIRD_PARTY_PROVIDED_LODGING	21.66%
	TENANT	18.22%
	EMPLOYER_PROVIDED_LODGING	0.78%
Marital_Status	MARRIED	48.50%
	SINGLE	27.19%
	DIVORCED	10.03%
	COHABITING	7.53%
	WIDOWED	4.09%
	SEPARATED	2.65%
People_in_Household	0	55.42%
	1	22.84%
	2	16.78%

	3	4.15%
	4	0.69%
	5	0.09%
	6	0.03%
Num_Ongoing_Credits	1	41.98%
	2	23.36%
	0	22.38%
	3	8.82%
	4	2.68%
	5	0.78%
Profession_Code	OPERATOR	16.75%
	ADMINISTRATIVE	11.71%
	TECHNICIAN	8.42%
	MIDDLEGRADEMANAGER	7.09%
	RETIREMENT	6.52%
	MEDICAL_PROFESSION	4.93%
	POLICEMAN_FIREMAN_MILITARY	4.64%
	EDUCATION	4.41%
	INVALIDITY	4.15%
	STAFFMANAGER	3.32%
	COMMERCIAL	3.20%
	ABSOLUTE_INVALIDITY	2.94%
	SALESMAN	2.60%
	HOTELIER	2.48%
	COMPUTERSCIENCE_MATH	2.36%
	EXECUTIVE	2.28%
	DRIVER	1.99%
	GUARD	1.93%
	NA	1.53%
	OTHER	1.44%
	ENGINEER	1.15%
	CONSULTANT	0.95%
	ANALYST_FINANCE_MARKETING	0.84%
	ENTREPRENEUR	0.66%
	LIBERAL_PROFESSION	0.46%
	LAWYER_NOTARY	0.26%
	RESIDENCEEMPLOYEE	0.20%
	MAJOR_INVALIDITY	0.17%
	ARCHITECT	0.17%
	CRAFTMAN_SALEMAN	0.14%
	LAWYER_JUDGE	0.12%
	ECONOMIST_ACCOUNTANT	0.12%
	DOCTOR	0.06%
Province	Barcelona	16.29%
	Madrid	13.99%
	Asturias (Oviedo)	5.71%
	Coruna	4.84%
	Valencia	4.35%
	Alicante	3.46%
	Vizcaya (Bilbao)	3.29%

	Pontevedra	3.29%
	Zaragoza	2.51%
	Guipuzcoa (Donostia-San Sebastian)	2.48%
	Navarra (Pamplona)	2.39%
	Leon	2.34%
	Cantabria (Santander)	2.25%
	Valladolid	2.22%
	Sevilla	1.70%
	Tarragona	1.70%
	Salamanca	1.59%
	Burgos	1.53%
	Gerona	1.50%
	Toledo	1.50%
	Alava (Vitoria-Gasteiz)	1.38%
	Lugo	1.36%
	Las Palmas	1.36%
	Murcia	1.27%
	Santa Cruz de Tenerife	1.21%
	Orense	1.21%
	La Rioja (Logrono)	0.98%
	Baleares (Palma de Mallorca)	0.92%
	Ciudad Real	0.89%
	Castellon	0.87%
	Malaga	0.84%
	Badajoz	0.78%
	Lerida	0.72%
	Albacete	0.72%
	Cadiz	0.69%
	Granada	0.63%
	Huelva	0.61%
	Avila	0.61%
	Caceres	0.61%
	Palencia	0.58%
	Guadalajara	0.46%
	Segovia	0.40%
	Cordoba	0.40%
	Almeria	0.35%
	Jaen	0.32%
	Huesca	0.29%
	Cuenca	0.26%
	Teruel	0.17%
	Soria	0.12%
	Zamora	0.09%

8.3. Code

This section contains the code of the whole thesis and it can be also found in the author's *GitHub* account https://github.com/alvarorgaz/Guide-to-Spark-Machine-Learning-for-credit-scoring with a more practical and attractive format.

```
### TITLE: Guide to Spark Machine Learning for credit scoring
### AUTHOR: Álvaro Orgaz Expósito
### ADVISORS: Ana María Pérez Marín Catalina Bolancé Losilla
### DEPARTMENT: Econometrics, Statistics and Applied Economics
### ACADEMIC YEAR: 2017-2018

### LEGEND OF CODE COMMENTS (#)
### 1#: Code actions
### 2#: Optional or additional functions
### 3#: Notes and extra comments about the code

#########################################################################################################

# 1. INSTALLATION AND CONNECTION TO SPARK

### Notes:
### N1. As the package "SparkR" is removed from R CRAN, download the package file "SparkR_2.3.0.tar.gz"
###     at the link: https://cran.r-project.org/src/contrib/Archive/SparkR/
### N2. You will need to have installed the software Java and the firewall unblocked.
### N3. The function "sparkR.init" of the package "SparkR" will install the latest version of Spark in
###     your computer (if not installed), then it is not necessary that you do it manually. It will be
###     used too by the function "spark_connect" of the package "sparklyr".

# Mute warnings
options(warn=-1)

# Install the necessary R packages from R CRAN (if not installed): "dplyr", "sparklyr", "pROC", "DBI",
# “ggplot2”
## install.packages("dplyr")
## install.packages("sparklyr")
## install.packages("pROC")
## install.packages("DBI")
## install.packages("ggplot2")

# Install the necessary R package removed from R CRAN (if not installed) : "SparkR"
## install.packages("SparkR_2.3.0.tar.gz",repos=NULL,type="source")

# Load the necessary installed packages
library(SparkR)
library(dplyr)
library(sparklyr)
library(pROC)
library(DBI)
library(ggplot2)

# Connect to Spark cluster in local mode (package "SparkR")
sc_SparkR <- sparkR.init(master="local")
sc_SparkR_sql <- sparkRSQL.init(sc_SparkR)

# Connect to Spark cluster in local mode (package "sparklyr")
sc_sparklyr <- spark_connect(master="local")

### Notes:
### N4. You can connect to both local instances of Spark as well as remote Spark clusters but we will
###     connect to a local. The returned Spark connection (sc) provides a remote data source to the
###     Spark cluster. Once you have connected to Spark, you will be able to browse the tables contained
###     in the Spark cluster and also, in the case of the package "sparklyr", preview Spark data frames
###     using the RStudio data viewer.

#########################################################################################################

# 2. DATA LOADING

### Notes:
### N5. You can read and write data in CSV, JSON, and Parquet formats. Data can be stored in remote
###     clusters or on the local cluster, and it returns a reference to a Spark data frame.

# Load the data specifying the type of variables: numerical ("double") or categorical ("character")
data <- spark_read_csv(sc=sc_sparklyr,
                       name="data",
                       path="Data Financed Defaults.csv",
                       header=TRUE,
                       delimiter=";",
                       infer_schema=FALSE,
                       columns=list(Contract_ID="character",
                                    Defaulted="character",
                                    Application_Hour_Group="character",
                                    Application_Week_Day="character",
                                    Amount="double",
                                    Maturity="double",
                                    Purpose="character",
                                    Province="character",
                                    Postal_Code_ASNEF="double",
                                    Age="double",
                                    Gender="character",
                                    Profession_Code="character",
                                    Profession_Sector="character",
                                    Contract_Type="character",
                                    Seniority="double",
                                    Housing_Type="character",
                                    Housing_Seniority="double",
                                    Marital_Status="character",
                                    People_in_Household="character",
                                    Income="double",
                                    Additional_Income="double",
                                    Partner_Income="double",
                                    Rent="double",
                                    Mortgage="double",
                                    Amount_of_Ongoing_Credits="double",
                                    Num_Ongoing_Credits="character"))

# Number of customers and variables
count(data)  ### 3468 customers
ncol(data)   ### 26 variables (including the identifier of customers)

#############################################################################################################

# 3. ANALYSIS OF THE DATA VARIABLES BEFORE PRE-PROCESSING

# Balanced or unbalanced dataset?
count(filter(data,Defaulted==1))  ### 705 = 20.33%
count(filter(data,Defaulted==0))  ### 2763 = 79.67%

# Variables by type
variables_type <- sdf_schema(data)
variables_type <- data.frame(Variable=names(variables_type),
                             Type=as.vector(unlist(sapply(names(variables_type),
                                                          function(i){variables_type[[i]][2]}))))
categorical <- variables_type[variables_type$Type=="StringType","Variable"]
numerical <- variables_type[variables_type$Type=="DoubleType","Variable"]
variables_type

# Missing values by variables
missings <- collect(data %>% mutate_all(is.na) %>% mutate_all(as.numeric) %>% summarize_all(sum))
missings <- data.frame(Variable=names(missings),Number_of_missings=as.vector(t(missings)))
missings[missings$Number_of_missings>0,]

# Categories with frequency distribution <5% in its variable
for(i in categorical[-which(categorical=="Contract_ID")]){
  show(dbGetQuery(sc_sparklyr,paste0("SELECT ",i,",COUNT(*)/3468 AS Distribution FROM data GROUP BY ",
                                     i," ORDER BY Distribution DESC")))
}

#############################################################################################################

# 4. DATA PRE-PROCESSING

# Modify missing values found in the analysis:
# - in numerical variables by 0
# - in categorical variables by "Missing"
data <- data %>% mutate(
  Postal_Code_ASNEF=ifelse(is.na(Postal_Code_ASNEF),0,Postal_Code_ASNEF),
  Additional_Income=ifelse(is.na(Additional_Income),0,Additional_Income),
  Partner_Income=ifelse(is.na(Partner_Income),0,Partner_Income),
  Rent=ifelse(is.na(Rent),0,Rent),
  Mortgage=ifelse(is.na(Mortgage),0,Mortgage),
  Profession_Code=ifelse(is.na(Profession_Code),"Missing",Profession_Code)
  )

# Create the list with valid levels for categorical variables (excluding categories with <5%)
valid_levels <- list(
  Levels_Application_Hour_Group=
    c("[23H, 7H)","[7H, 20H)","[20H, 23H)","OTHERS"),
  Levels_Application_Week_Day=
    c("1","2","3","4","5","6","7","OTHERS"),
  Levels_Gender=
    c("MALE","FEMALE","OTHERS"),
  Levels_Profession_Sector=
    c("PRIVATE_SECTOR","PUBLIC_SECTOR","OTHERS"),
  Levels_Contract_Type=
    c("PERMANENT","PENSION","OTHERS"),
  Levels_People_in_Household=
    c("0","1","2","OTHERS"),
  Levels_Num_Ongoing_Credits=
    c("0","1","2","3","OTHERS"),
  Levels_Marital_Status=
    c("DIVORCED","SINGLE","COHABITING","MARRIED","OTHERS"),
  Levels_Province=
    c("Madrid","Barcelona","Asturias (Oviedo)","OTHERS"),
  Levels_Profession_Code=
    c("OPERATOR", "ADMINISTRATIVE","TECHNICIAN","MIDDLEGRADEMANAGER","RETIREMENT","OTHERS"),

Levels_Purpose=
    c("LIQUIDITY","HOMEIMPROVEMENT","DEBTS","FURNITURE_AND_APPLIANCES","USEDCAR","MEDICALCARE",
      "VACATION","OTHERS"),
  Levels_Housing_Type=
    c("THIRD_PARTY_PROVIDED_LODGING","HOME_OWNERSHIP_WITHOUT_MORTGAGE","TENANT",
      "HOME_OWNERSHIP_WITH_MORTGAGE","OTHERS")
  )

# Change outliers categories by OTHERS
data <- data %>% mutate(
  Purpose=
    ifelse(Purpose %in% valid_levels[["Levels_Purpose"]],Purpose,"OTHERS"),
  Gender=
    ifelse(Gender %in% valid_levels[["Levels_Gender"]],Gender,"OTHERS"),
  Housing_Type=
    ifelse(Housing_Type %in% valid_levels[["Levels_Housing_Type"]],Housing_Type,"OTHERS"),
  Province=
    ifelse(Province %in% valid_levels[["Levels_Province"]],Province,"OTHERS"),
  Marital_Status=
    ifelse(Marital_Status %in% valid_levels[["Levels_Marital_Status"]],Marital_Status,"OTHERS"),
  Profession_Code=
    ifelse(Profession_Code %in% valid_levels[["Levels_Profession_Code"]],Profession_Code,"OTHERS"),
  Contract_Type=
    ifelse(Contract_Type %in% valid_levels[["Levels_Contract_Type"]],Contract_Type,"OTHERS"),
  Profession_Sector=
    ifelse(Profession_Sector %in% valid_levels[["Levels_Profession_Sector"]],Profession_Sector,
           "OTHERS"),
  Application_Week_Day=
    ifelse(Application_Week_Day %in% valid_levels[["Levels_Application_Week_Day"]],
           Application_Week_Day,"OTHERS"),
  People_in_Household=
    ifelse(People_in_Household %in% valid_levels[["Levels_People_in_Household"]],People_in_Household,
           "OTHERS"),
  Num_Ongoing_Credits=
    ifelse(Num_Ongoing_Credits %in% valid_levels[["Levels_Num_Ongoing_Credits"]],Num_Ongoing_Credits,
           "OTHERS"),
  Application_Hour_Group=
    ifelse(Application_Hour_Group %in% valid_levels[["Levels_Application_Hour_Group"]],
           Application_Hour_Group,"OTHERS")
  )

### Notes:
### N6. In the case that you need to convert all categorical variables to numerical with an integer
###     index, the necessary Spark R code could be:

## for(i in categorical[-which(categorical=="Contract_ID" | categorical=="Defaulted")]){
##   label_i  <- as.vector(valid_levels[[paste0("Levels_",i)]])
##   data <- data %>% ft_string_indexer_model(input_col=i,output_col=paste0(i,"_INDEXED"),
##           labels=label_i)
## }

# Save the Spark data frame with the pre-processed data in the Spark cluster
data <- copy_to(sc_sparklyr,data,overwrite=T)

# Save the Spark data frame with the pre-processed data in a local file with format Parquet
## spark_write_parquet(data,"data.parquet")

#####################################################################################################

# 5. PROTOCOL OF MODEL VALIDATION PHASE 1: Create training and test sets as well as training folds for CV

# Read the Spark data frame with the pre-processed data in a local file with format Parquet
## data <- spark_read_parquet(sc_sparklyr,"data","data.parquet")

# Create training and test datasets (75%-25%)
data_partitions <- data %>% compute("data_partitions") %>% sdf_partition(train=0.75,test=0.25,seed=1)

# Create K folds or partitions from training data for cross-validation
K <- 5
weights <- rep(1/K,times = K)
names(weights) <- paste0("Fold ",as.character(1:K))
train_partitions <- data_partitions$train %>% compute("train_partitions") %>%
  sdf_partition(weights=weights,seed=1)

# Check that all categories in the test set are included in the training set
for(i in categorical[-which(categorical=="Contract_ID")]){
  cat("Does the variable ",i," have the same categories in training and test sets?","\n")
  cat(sum(!unique(as.data.frame(collect(data_partitions$test))[,i]) %in%
           unique(as.data.frame(collect(data_partitions$train))[,i]))==0,"\n")
}
```

```r
### Notes:
### N7. The function "sdf_partition" returns a list with as much Spark datasets as you define. The
###     fold weights are the probabilities of being in every fold for the observations, and they do not
###     mean the fold size.

################################################################################################################

# 6. PROTOCOL OF MODEL VALIDATION PHASE 2: Find the best parametrization of every model with training set

# Create the vectors with the names of variables by types
out <- c("Contract_ID","Postal_Code_ASNEF","Additional_Income","Partner_Income","Rent","Mortgage")
response <- c("Defaulted")
features_num <- c("Amount","Maturity","Postal_Code_ASNEF","Age","Seniority","Housing_Seniority",
                  "Income","Additional_Income","Rent","Partner_Income","Mortgage",
                  "Amount_of_Ongoing_Credits")
features_cat <- c("Application_Hour_Group","Application_Week_Day","Purpose","Province","Gender",
                  "Profession_Code","Profession_Sector","Contract_Type","Housing_Type",
                  "Marital_Status","People_in_Household","Num_Ongoing_Credits")

# 6.1 LOGISTIC REGRESSION   ##############################################################################

# Starting time
Sys.time()

# Create the list of candidate parametrizations
params <- list(features_num,features_cat,c(features_num,features_cat))

# Iterate all candidate parametrizations
results_cv_LR <- data.frame()

for(p in 1:length(params)){

  response_LR_global <- c()
  pred_LR_global <- c()

  # Iterate all training folds
  for(i in 1:K){

    # Create the training and test sets for the ith fold
    test_i <- train_partitions[[i]]
    train_i <- rbind(train_partitions[[c(1:K)[-i][1]]],train_partitions[[c(1:K)[-i][2]]],
                     train_partitions[[c(1:K)[-i][3]]],train_partitions[[c(1:K)[-i][4]]])

    # Train the model without ith fold and the pth parametrization
    model_LR_i <- ml_logistic_regression(train_i,
                                         response=response,
                                         features=params[[p]])

    # Predict ith fold with the pth parametrization
    pred_LR_i <- sdf_predict(test_i,model_LR_i)
        pred_LR_i <- data.frame(collect(pred_LR_i %>% select(probability_1)))[,"probability_1"]

    # Calculate the OOB AUC for the ith fold with the pth parametrization
    response_i <- data.frame(collect(test_i %>% select(Defaulted)))[,"Defaulted"]
    results_cv_LR[p,i] <- auc(roc(response_i,pred_LR_i))

    # Save the ith fold response and predictions with the pth parametrization
    response_LR_global <- c(response_LR_global,response_i)
    pred_LR_global <- c(pred_LR_global,pred_LR_i)
  }

  # Calculate the rest of performance measures with the global training set predicted as OOB
  quartile_cutoff <- quantile(pred_LR_global,seq(0.25,0.75,0.25))
  quartile <- ifelse(pred_LR_global<=quartile_cutoff[1],"Q1",
                     ifelse(pred_LR_global<=quartile_cutoff[2],"Q2",
                            ifelse(pred_LR_global<=quartile_cutoff[3],"Q3","Q4")))
  results_cv_LR[p,6] <- auc(roc(response_LR_global,pred_LR_global))
  results_cv_LR[p,7] <- mean(response_LR_global[quartile=="Q1"]==1)
  results_cv_LR[p,8] <- mean(response_LR_global[quartile=="Q2"]==1)
  results_cv_LR[p,9] <- mean(response_LR_global[quartile=="Q3"]==1)
  results_cv_LR[p,10] <- mean(response_LR_global[quartile=="Q4"]==1)
}

# Print the results table
row.names(results_cv_LR) <- c("Parametrization 1: numerical features",
                              "Parametrization 2: categorical features",
                              "Parametrization 3: all features")
names(results_cv_LR) <- c("AUC Fold 1","AUC Fold 2","AUC Fold 3","AUC Fold 4","AUC Fold 5",
                          "AUC Global","% True + Q1","% True + Q2","% True + Q3","% True + Q4")
save(results_cv_LR,file="results_cv_LR.RData")
results_cv_LR

# Finishing time
Sys.time()

# 6.2 DECISION TREE   ####################################################################################

# Starting time
```

```r
Sys.time()

# Create the list of candidate parametrizations
max_bins <- c(10,20,30)
max_depth <- c(5,10,15)
min_instances_per_node <- c(5,9,13)
params <- expand.grid(max_bins,max_depth,min_instances_per_node)
params <- sapply(1:nrow(params),function(i){list(params[i,])})

# Iterate all candidate parametrizations
results_cv_DT <- data.frame()

for(p in 1:length(params)){

  response_DT_global <- c()
  pred_DT_global <- c()

  # Iterate all training folds
  for(i in 1:K){

    # Create the training and test sets for the ith fold
    test_i <- train_partitions[[i]]
    train_i <- rbind(train_partitions[[c(1:K)[-i][1]]],train_partitions[[c(1:K)[-i][2]]],
                     train_partitions[[c(1:K)[-i][3]]],train_partitions[[c(1:K)[-i][4]]])

    # Train the model without ith fold and the pth parametrization
    model_DT_i <- ml_decision_tree(train_i,
                                   type="classification",
                                   response=response,
                                   features=c(features_num,features_cat),
                                   max_bins=as.numeric(params[[p]][1]),
                                   max_depth=as.numeric(params[[p]][2]),
                                   min_instances_per_node=as.numeric(params[[p]][3]),
                                   seed=1)

    # Predict ith fold with the pth parametrization
    pred_DT_i <- sdf_predict(test_i,model_DT_i)
    pred_DT_i <- data.frame(collect(pred_DT_i %>% select(probability_1)))[,"probability_1"]

    # Calculate the OOB AUC for the ith fold with the pth parametrization
    response_i <- data.frame(collect(test_i %>% select(Defaulted)))[,"Defaulted"]
    results_cv_DT[p,i] <- auc(roc(response_i,pred_DT_i))

    # Save the ith fold response and predictions with the pth parametrization
    response_DT_global <- c(response_DT_global,response_i)
    pred_DT_global <- c(pred_DT_global,pred_DT_i)
  }

  # Calculate the rest of performance measures with the global training set predicted as OOB
  quartile_cutoff <- quantile(pred_DT_global,seq(0.25,0.75,0.25))
  quartile <- ifelse(pred_DT_global<=quartile_cutoff[1],"Q1",
                     ifelse(pred_DT_global<=quartile_cutoff[2],"Q2",
                            ifelse(pred_DT_global<=quartile_cutoff[3],"Q3","Q4")))
  results_cv_DT[p,6] <- auc(roc(response_DT_global,pred_DT_global))
  results_cv_DT[p,7] <- mean(response_DT_global[quartile=="Q1"]==1)
  results_cv_DT[p,8] <- mean(response_DT_global[quartile=="Q2"]==1)
  results_cv_DT[p,9] <- mean(response_DT_global[quartile=="Q3"]==1)
  results_cv_DT[p,10] <- mean(response_DT_global[quartile=="Q4"]==1)
}

# Print the results table
row.names(results_cv_DT) <- sapply(1:length(params),function(i){
  paste0("max_bins=",params[[i]][1],";max_depth=",params[[i]][2],";min_instances_node=",
         params[[i]][3])})
names(results_cv_DT) <- c("AUC Fold 1","AUC Fold 2","AUC Fold 3","AUC Fold 4","AUC Fold 5",
                          "AUC Global","% True + Q1","% True + Q2","% True + Q3","% True + Q4")
save(results_cv_DT,file="results_cv_DT.RData")
results_cv_DT

# Finishing time
Sys.time()

# 6.3 RANDOM FOREST   #####################################################################

# Starting time
Sys.time()

# Create the list of candidate parametrizations
max_bins <- c(10,20,30)
max_depth <- c(5,10,15)
num.trees <- c(15,30,45)
min_instances_per_node <- c(5,9,13)
params <- expand.grid(max_bins,max_depth,num.trees,min_instances_per_node)
params <- sapply(1:nrow(params),function(i){list(params[i,])})

# Iterate all candidate parametrizations
results_cv_RF <- data.frame()
```

```r
for(p in 1:length(params)){

  response_RF_global <- c()
  pred_RF_global <- c()

  # Iterate all training folds
  for(i in 1:K){

    # Create the training and test sets for the ith fold
    test_i <- train_partitions[[i]]
    train_i <- rbind(train_partitions[[c(1:K)[-i][1]]],train_partitions[[c(1:K)[-i][2]]],
                     train_partitions[[c(1:K)[-i][3]]],train_partitions[[c(1:K)[-i][4]]])

    # Train the model without ith fold and the pth parametrization
    model_RF_i <- ml_random_forest(train_i,
                                   type="classification",
                                   response=response,
                                   features=c(features_num,features_cat),
                                   max_bins=as.numeric(params[[p]][1]),
                                   max_depth=as.numeric(params[[p]][2]),
                                   num_trees=as.numeric(params[[p]][3]),
                                   min_instances_per_node=as.numeric(params[[p]][4]),
                                   subsampling_rate=1,
                                   seed=1)

    # Predict ith fold with the pth parametrization
    pred_RF_i <- sdf_predict(test_i,model_RF_i)
    pred_RF_i <- data.frame(collect(pred_RF_i %>% select(probability_1)))[,"probability_1"]

    # Calculate the OOB AUC for the ith fold with the pth parametrization
    response_i <- data.frame(collect(test_i %>% select(Defaulted)))[,"Defaulted"]
    results_cv_RF[p,i] <- auc(roc(response_i,pred_RF_i))

    # Save the ith fold response and predictions with the pth parametrization
    response_RF_global <- c(response_RF_global,response_i)
    pred_RF_global <- c(pred_RF_global,pred_RF_i)
  }

  # Calculate the rest of performance measures with the global training set predicted as OOB
  quartile_cutoff <- quantile(pred_RF_global,seq(0.25,0.75,0.25))
  quartile <- ifelse(pred_RF_global<=quartile_cutoff[1],"Q1",
                     ifelse(pred_RF_global<=quartile_cutoff[2],"Q2",
                            ifelse(pred_RF_global<=quartile_cutoff[3],"Q3","Q4")))
  results_cv_RF[p,6] <- auc(roc(response_RF_global,pred_RF_global))
  results_cv_RF[p,7] <- mean(response_RF_global[quartile=="Q1"]==1)
  results_cv_RF[p,8] <- mean(response_RF_global[quartile=="Q2"]==1)
  results_cv_RF[p,9] <- mean(response_RF_global[quartile=="Q3"]==1)
  results_cv_RF[p,10] <- mean(response_RF_global[quartile=="Q4"]==1)
}

# Print the results table
row.names(results_cv_RF) <- sapply(1:length(params),function(i){
  paste0("max_bins=",params[[i]][1],";max_depth=",params[[i]][2],";num_trees=",params[[i]][3],
         ";min_instances_node=",params[[i]][4])})
names(results_cv_RF) <- c("AUC Fold 1","AUC Fold 2","AUC Fold 3","AUC Fold 4","AUC Fold 5",
                          "AUC Global","% True + Q1","% True + Q2","% True + Q3","% True + Q4")
save(results_cv_RF,file="results_cv_RF.RData")
results_cv_RF

# Finishing time
Sys.time()

# 6.4 GRADIENT BOOSTED TREES    ############################################################################

# Starting time
Sys.time()

# Create the list of candidate parametrizations
max_depth <- c(5,10,15)
max_iter <- c(15,30,45)
step_size <- c(0.05,0.1,0.15)
params <- expand.grid(max_depth,max_iter,step_size)
params <- sapply(1:nrow(params),function(i){list(params[i,])})

# Iterate all candidate parametrizations
results_cv_GBT <- data.frame()

for(p in 1:length(params)){

  response_GBT_global <- c()
  pred_GBT_global <- c()

  # Iterate all training folds
  for(i in 1:K){

    # Create the training and test sets for the ith fold
    test_i <- train_partitions[[i]]
    train_i <- rbind(train_partitions[[c(1:K)[-i][1]]],train_partitions[[c(1:K)[-i][2]]],
                   train_partitions[[c(1:K)[-i][3]]],train_partitions[[c(1:K)[-i][4]]])

    # Create the training and test sets for the ith fold in package "SparkR" format
    test_i_SparkR <- createDataFrame(sc_SparkR_sql,as.data.frame(collect(test_i)))
    train_i_SparkR <- createDataFrame(sc_SparkR_sql,as.data.frame(collect(train_i)))

    # Train the model without ith fold and the pth parametrization
    formula <- as.formula(paste(response,"~",paste(c(features_num,features_cat),collapse="+")))
    model_GBT_i <- spark.gbt(train_i_SparkR,
                             formula=formula,
                             type="classification",
                             maxDepth=as.numeric(params[[p]][1]),
                             maxIter=as.numeric(params[[p]][2]),
                             stepSize=as.numeric(params[[p]][3]),
                             subsamplingRate=1,
                             seed=1)

    # Predict ith fold with the pth parametrization
    pred_GBT_i <- predict(model_GBT_i,test_i_SparkR)
    pred_GBT_i <- unlist(lapply(as.data.frame(pred_GBT_i)[,"probability"],
                                function(x)SparkR:::callJMethod(x,"toArray")[[2]]))

    # Calculate the OOB AUC for the ith fold with the pth parametrization
    response_i <- data.frame(collect(test_i %>% select(Defaulted)))[,"Defaulted"]
    results_cv_GBT[p,i] <- auc(roc(response_i,pred_GBT_i))

    # Save the ith fold response and predictions with the pth parametrization
    response_GBT_global <- c(response_GBT_global,response_i)
    pred_GBT_global <- c(pred_GBT_global,pred_GBT_i)
  }

  # Calculate the rest of performance measures with the global training set predicted as OOB
  quartile_cutoff <- quantile(pred_GBT_global,seq(0.25,0.75,0.25))
  quartile <- ifelse(pred_GBT_global<=quartile_cutoff[1],"Q1",
                   ifelse(pred_GBT_global<=quartile_cutoff[2],"Q2",
                          ifelse(pred_GBT_global<=quartile_cutoff[3],"Q3","Q4")))
  results_cv_GBT[p,6] <- auc(roc(response_GBT_global,pred_GBT_global))
  results_cv_GBT[p,7] <- mean(response_GBT_global[quartile=="Q1"]==1)
  results_cv_GBT[p,8] <- mean(response_GBT_global[quartile=="Q2"]==1)
  results_cv_GBT[p,9] <- mean(response_GBT_global[quartile=="Q3"]==1)
  results_cv_GBT[p,10] <- mean(response_GBT_global[quartile=="Q4"]==1)
}

# Print the results table
row.names(results_cv_GBT) <- sapply(1:length(params),function(i){
  paste0("max_depth=",params[[i]][1],";max_iter=",params[[i]][2],
         ";step_size=",params[[i]][3])})
names(results_cv_GBT) <- c("AUC Fold 1","AUC Fold 2","AUC Fold 3","AUC Fold 4","AUC Fold 5",
                           "AUC Global","% True + Q1","% True + Q2","% True + Q3","% True + Q4")
save(results_cv_GBT,file="results_cv_GBT.RData")
results_cv_GBT

# Finishing time
Sys.time()

# 6.5 NAIVE BAYES   #########################################################################

# Starting time
Sys.time()

# Create the list of candidate parametrizations
params <- list(features_num,features_cat,c(features_num,features_cat))

# Iterate all candidate parametrizations
results_cv_NB <- data.frame()

for(p in 1:length(params)){

  response_NB_global <- c()
  pred_NB_global <- c()

  # Iterate all training folds
  for(i in 1:K){

    # Create the training and test sets for the ith fold
    test_i <- train_partitions[[i]]
    train_i <- rbind(train_partitions[[c(1:K)[-i][1]]],train_partitions[[c(1:K)[-i][2]]],
                   train_partitions[[c(1:K)[-i][3]]],train_partitions[[c(1:K)[-i][4]]])

    # Train the model without ith fold and the pth parametrization
    model_NB_i <- ml_naive_bayes(train_i,
                                 response=response,
                                 features=params[[p]])

    # Predict ith fold with the pth parametrization
    pred_NB_i <- sdf_predict(test_i,model_NB_i)
    pred_NB_i <- data.frame(collect(pred_NB_i %>% select(probability_1)))[,"probability_1"]
    # Calculate the OOB AUC for the ith fold with the pth parametrization
    response_i <- data.frame(collect(test_i %>% select(Defaulted)))[,"Defaulted"]
    results_cv_NB[p,i] <- auc(roc(response_i,pred_NB_i))

    # Save the ith fold response and predictions with the pth parametrization
    response_NB_global <- c(response_NB_global,response_i)
    pred_NB_global <- c(pred_NB_global,pred_NB_i)
  }
  # Calculate the rest of performance measures with the global training set predicted as OOB
  quartile_cutoff <- quantile(pred_NB_global,seq(0.25,0.75,0.25))
  quartile <- ifelse(pred_NB_global<=quartile_cutoff[1],"Q1",
                   ifelse(pred_NB_global<=quartile_cutoff[2],"Q2",
                          ifelse(pred_NB_global<=quartile_cutoff[3],"Q3","Q4")))
  results_cv_NB[p,6] <- auc(roc(response_NB_global,pred_NB_global))
  results_cv_NB[p,7] <- mean(response_NB_global[quartile=="Q1"]==1)
  results_cv_NB[p,8] <- mean(response_NB_global[quartile=="Q2"]==1)
  results_cv_NB[p,9] <- mean(response_NB_global[quartile=="Q3"]==1)
  results_cv_NB[p,10] <- mean(response_NB_global[quartile=="Q4"]==1)
}

# Print the results table
row.names(results_cv_NB) <- c("Parametrization 1: numerical features",
                              "Parametrization 2: categorical features",
                              "Parametrization 3: all features")
names(results_cv_NB) <- c("AUC Fold 1","AUC Fold 2","AUC Fold 3","AUC Fold 4","AUC Fold 5",
                          "AUC Global","% True + Q1","% True + Q2","% True + Q3","% True + Q4")
save(results_cv_NB,file="results_cv_NB.RData")
results_cv_NB

# Finishing time
Sys.time()

# 6.6 NEURAL NETWORK    ####################################################################################

# Starting time
Sys.time()

# Create the list of candidate parametrizations
layer_hidden_1 <- c(6,8,10,12)
layer_hidden_2 <- c(4,6,8,10)
params <- expand.grid(layer_hidden_1,layer_hidden_2)
params <- sapply(1:nrow(params),function(i){list(params[i,])})

# Iterate all candidate parametrizations
results_cv_NN <- data.frame()

for(p in 1:length(params)){

  response_NN_global <- c()
  pred_NN_global <- c()

  # Iterate all training folds
  for(i in 1:K){

    # Create the training and test sets for the ith fold
    test_i <- train_partitions[[i]]
    train_i <- rbind(train_partitions[[c(1:K)[-i][1]]],train_partitions[[c(1:K)[-i][2]]],
                     train_partitions[[c(1:K)[-i][3]]],train_partitions[[c(1:K)[-i][4]]])

    # Create the training and test sets for the ith fold in package "SparkR" format
    test_i_SparkR <- createDataFrame(sc_SparkR_sql,as.data.frame(collect(test_i)))
    train_i_SparkR <- createDataFrame(sc_SparkR_sql,as.data.frame(collect(train_i)))

    # Establish the number of neurons:
    layer_output <- 2
    layer_hidden_1 <- as.numeric(params[[p]][1])
    layer_hidden_2 <- as.numeric(params[[p]][2])
    layer_input <- sum(sapply(as.data.frame(train_i_SparkR[,features_cat]),
                          function(x)length(unique(x))-1),length(features_num))

    # Train the model without ith fold and the pth parametrization
    formula <- as.formula(paste(response,"~",paste(c(features_num,features_cat),collapse="+")))
    model_NN_i <- spark.mlp(train_i_SparkR,
                        formula=formula,
                        layers=c(layer_input,layer_hidden_1,layer_hidden_2,layer_output),
                        seed=1)

    # Predict ith fold with the pth parametrization
    pred_NN_i <- predict(model_NN_i,test_i_SparkR)
    pred_NN_i <- unlist(lapply(as.data.frame(pred_NN_i)[,"probability"],
                          function(x)SparkR:::callJMethod(x,"toArray")[[2]]))

    # Calculate the OOB AUC for the ith fold with the pth parametrization
    response_i <- data.frame(collect(test_i %>% select(Defaulted)))[,"Defaulted"]
    results_cv_NN[p,i] <- auc(roc(response_i,pred_NN_i))

    # Save the ith fold response and predictions with the pth parametrization
    response_NN_global <- c(response_NN_global,response_i)
    pred_NN_global <- c(pred_NN_global,pred_NN_i)
  }

  # Calculate the rest of performance measures with the global training set predicted as OOB
  quartile_cutoff <- quantile(pred_NN_global,seq(0.25,0.75,0.25))
  quartile <- ifelse(pred_NN_global<=quartile_cutoff[1],"Q1",
                     ifelse(pred_NN_global<=quartile_cutoff[2],"Q2",
                            ifelse(pred_NN_global<=quartile_cutoff[3],"Q3","Q4")))
  results_cv_NN[p,6] <- auc(roc(response_NN_global,pred_NN_global))
  results_cv_NN[p,7] <- mean(response_NN_global[quartile=="Q1"]==1)
  results_cv_NN[p,8] <- mean(response_NN_global[quartile=="Q2"]==1)
  results_cv_NN[p,9] <- mean(response_NN_global[quartile=="Q3"]==1)
  results_cv_NN[p,10] <- mean(response_NN_global[quartile=="Q4"]==1)
}

# Print the results table
row.names(results_cv_NN) <- sapply(1:length(params),function(i){
  paste0("hidden_layer_1=",params[[i]][1],";hidden_layer_2=",params[[i]][2])})
names(results_cv_NN) <- c("AUC Fold 1","AUC Fold 2","AUC Fold 3","AUC Fold 4","AUC Fold 5",
                          "AUC Global","% True + Q1","% True + Q2","% True + Q3","% True + Q4")
save(results_cv_NN,file="results_cv_NN.RData")
results_cv_NN

# Finishing time
Sys.time()

### Notes:
### N8. In the Multilayer Perceptron Neural Network, the number of neurons by layers are:
###     - output layer: the # of classes in the response.
###     - hidden layers: tunning parameters.
###     - input layer: as much nodes as input variables. It means the # of numerical variables, plus
###       the # of unique categories in all categorical variables, less the # of categorical features
###       (because the model creates, for every categorical feature, dummies for every category except
###       one for avoiding linear dependence).

############################################################################################################

# 7. PROTOCOL OF MODEL VALIDATION PHASE 3: Train models with optimal parametrizations

# Create target variable of training and test sets
response_train <- data.frame(collect(data_partitions$train %>% select(Defaulted)))[,"Defaulted"]
response_test <- data.frame(collect(data_partitions$test %>% select(Defaulted)))[,"Defaulted"]

# Create the training and test sets in "SparkR" package format
train_SparkR <- createDataFrame(sc_SparkR_sql,as.data.frame(collect(data_partitions$train)))
test_SparkR <- createDataFrame(sc_SparkR_sql,as.data.frame(collect(data_partitions$test)))

# Create the results table, one for training and one for test
metrics_names <- c("Sensitivity + Specificity","Accuracy","AUC","% True + Q1","% True + Q2",
                   "% True + Q3","% True + Q4")
row.names <- c(paste0("Criteria: ",metrics_names),paste0("Cut-off: ",metrics_names[-3]))
results_optimals_train <- data.frame(row.names=row.names)
results_optimals_test <- data.frame(row.names=row.names)

# 7.1 LOGISTIC REGRESSION     #############################################################################

# Train the optimal model with all training set
model_LR <- ml_logistic_regression(data_partitions$train,
                                   response=response,
                                   features=c(features_num,features_cat))

# Summary of the model
summary(model_LR)

# Predict training data
pred_LR_train <- sdf_predict(data_partitions$train,model_LR)
pred_LR_train <- data.frame(collect(pred_LR_train %>% select(probability_1)))[,"probability_1"]

# Predict test data
pred_LR_test <- sdf_predict(data_partitions$test,model_LR)
pred_LR_test <- data.frame(collect(pred_LR_test %>% select(probability_1)))[,"probability_1"]

# Iterate all possible cut-offs and calculate performance measures in training set
m1_LR_train <- c()
m2_LR_train <- c()
lowest <- trunc(min(pred_LR_train)*100)/100+0.01
highest <- trunc(max(pred_LR_train)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_LR_train_c <- table(Real=response_train,Predicted=ifelse(pred_LR_train>=c,"1","0"))
  m1_LR_train <- c(m1_LR_train,table_LR_train_c[2,2]/sum(table_LR_train_c[2,]) +
                             table_LR_train_c[1,1]/sum(table_LR_train_c[1,]))
  m2_LR_train <- c(m2_LR_train,sum(diag(table_LR_train_c))/sum(table_LR_train_c))
}

cutoffs_LR_train <- quantile(pred_LR_train,seq(0.25,1,0.25))
quartiles_LR_train <- ifelse(pred_LR_train<=cutoffs_LR_train[1],"Q1",
                             ifelse(pred_LR_train<=cutoffs_LR_train[2],"Q2",
                                    ifelse(pred_LR_train<=cutoffs_LR_train[3],"Q3","Q4")))

results_optimals_train[1,"LR"] <- max(m1_LR_train)
results_optimals_train[2,"LR"] <- max(m2_LR_train)
results_optimals_train[3,"LR"] <- auc(roc(response_train,pred_LR_train))
results_optimals_train[4,"LR"] <- mean(response_train[quartiles_LR_train=="Q1"]==1)
results_optimals_train[5,"LR"] <- mean(response_train[quartiles_LR_train=="Q2"]==1)
results_optimals_train[6,"LR"] <- mean(response_train[quartiles_LR_train=="Q3"]==1)
results_optimals_train[7,"LR"] <- mean(response_train[quartiles_LR_train=="Q4"]==1)
results_optimals_train[8,"LR"] <- seq(lowest,highest,0.01)[which(m1_LR_train==max(m1_LR_train))[1]]
results_optimals_train[9,"LR"] <- seq(lowest,highest,0.01)[which(m2_LR_train==max(m2_LR_train))[1]]
results_optimals_train[10,"LR"] <- cutoffs_LR_train[1]
results_optimals_train[11,"LR"] <- cutoffs_LR_train[2]
results_optimals_train[12,"LR"] <- cutoffs_LR_train[3]
results_optimals_train[13,"LR"] <- cutoffs_LR_train[4]

# Iterate all possible cut-offs and calculate performance measures in test set
m1_LR_test <- c()
m2_LR_test <- c()
lowest <- trunc(min(pred_LR_test)*100)/100+0.01
highest <- trunc(max(pred_LR_test)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_LR_test_c <- table(Real=response_test,Predicted=ifelse(pred_LR_test>=c,"1","0"))

  m1_LR_test <- c(m1_LR_test,table_LR_test_c[2,2]/sum(table_LR_test_c[2,]) +
                     table_LR_test_c[1,1]/sum(table_LR_test_c[1,]))
  m2_LR_test <- c(m2_LR_test,sum(diag(table_LR_test_c))/sum(table_LR_test_c))
}

cutoffs_LR_test <- quantile(pred_LR_test,seq(0.25,1,0.25))
quartiles_LR_test <- ifelse(pred_LR_test<=cutoffs_LR_test[1],"Q1",
                             ifelse(pred_LR_test<=cutoffs_LR_test[2],"Q2",
                                    ifelse(pred_LR_test<=cutoffs_LR_test[3],"Q3","Q4")))

results_optimals_test[1,"LR"] <- max(m1_LR_test)
results_optimals_test[2,"LR"] <- max(m2_LR_test)
results_optimals_test[3,"LR"] <- auc(roc(response_test,pred_LR_test))
results_optimals_test[4,"LR"] <- mean(response_test[quartiles_LR_test=="Q1"]==1)
results_optimals_test[5,"LR"] <- mean(response_test[quartiles_LR_test=="Q2"]==1)
results_optimals_test[6,"LR"] <- mean(response_test[quartiles_LR_test=="Q3"]==1)
results_optimals_test[7,"LR"] <- mean(response_test[quartiles_LR_test=="Q4"]==1)
results_optimals_test[8,"LR"] <- seq(lowest,highest,0.01)[which(m1_LR_test==max(m1_LR_test))[1]]
results_optimals_test[9,"LR"] <- seq(lowest,highest,0.01)[which(m2_LR_test==max(m2_LR_test))[1]]
results_optimals_test[10,"LR"] <- cutoffs_LR_test[1]
results_optimals_test[11,"LR"] <- cutoffs_LR_test[2]
results_optimals_test[12,"LR"] <- cutoffs_LR_test[3]
results_optimals_test[13,"LR"] <- cutoffs_LR_test[4]

# 7.2 DECISION TREE     ###################################################################################

# Train the optimal model with all training set
model_DT <- ml_decision_tree(data_partitions$train,
                             type="classification",
                             response=response,
                             features=c(features_num,features_cat),
                             max_bins=30,
                             max_depth=5,
                             min_instances_per_node=5,
                             seed=1)

# Features importance
ml_feature_importances(model_DT)

# Predict training data
pred_DT_train <- sdf_predict(data_partitions$train,model_DT)
pred_DT_train <- data.frame(collect(pred_DT_train %>% select(probability_1)))[,"probability_1"]

# Predict test data
pred_DT_test <- sdf_predict(data_partitions$test,model_DT)
pred_DT_test <- data.frame(collect(pred_DT_test %>% select(probability_1)))[,"probability_1"]

# Iterate all possible cut-offs and calculate performance measures in training set
m1_DT_train <- c()
m2_DT_train <- c()
lowest <- trunc(min(pred_DT_train)*100)/100+0.01
highest <- trunc(max(pred_DT_train)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_DT_train_c <- table(Real=response_train,Predicted=ifelse(pred_DT_train>=c,"1","0"))
  m1_DT_train <- c(m1_DT_train,table_DT_train_c[2,2]/sum(table_DT_train_c[2,]) +
                     table_DT_train_c[1,1]/sum(table_DT_train_c[1,]))
  m2_DT_train <- c(m2_DT_train,sum(diag(table_DT_train_c))/sum(table_DT_train_c))
}

cutoffs_DT_train <- quantile(pred_DT_train,seq(0.25,1,0.25))
quartiles_DT_train <- ifelse(pred_DT_train<=cutoffs_DT_train[1],"Q1",
                             ifelse(pred_DT_train<=cutoffs_DT_train[2],"Q2",
                                    ifelse(pred_DT_train<=cutoffs_DT_train[3],"Q3","Q4")))

results_optimals_train[1,"DT"] <- max(m1_DT_train)
results_optimals_train[2,"DT"] <- max(m2_DT_train)
results_optimals_train[3,"DT"] <- auc(roc(response_train,pred_DT_train))
results_optimals_train[4,"DT"] <- mean(response_train[quartiles_DT_train=="Q1"]==1)
results_optimals_train[5,"DT"] <- mean(response_train[quartiles_DT_train=="Q2"]==1)
results_optimals_train[6,"DT"] <- mean(response_train[quartiles_DT_train=="Q3"]==1)
results_optimals_train[7,"DT"] <- mean(response_train[quartiles_DT_train=="Q4"]==1)
results_optimals_train[8,"DT"] <- seq(lowest,highest,0.01)[which(m1_DT_train==max(m1_DT_train))[1]]
results_optimals_train[9,"DT"] <- seq(lowest,highest,0.01)[which(m2_DT_train==max(m2_DT_train))[1]]
results_optimals_train[10,"DT"] <- cutoffs_DT_train[1]
results_optimals_train[11,"DT"] <- cutoffs_DT_train[2]
results_optimals_train[12,"DT"] <- cutoffs_DT_train[3]
results_optimals_train[13,"DT"] <- cutoffs_DT_train[4]

# Iterate all possible cut-offs and calculate performance measures in test set
m1_DT_test <- c()
m2_DT_test <- c()
lowest <- trunc(min(pred_DT_test)*100)/100+0.01
highest <- trunc(max(pred_DT_test)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_DT_test_c <- table(Real=response_test,Predicted=ifelse(pred_DT_test>=c,"1","0"))
  m1_DT_test <- c(m1_DT_test,table_DT_test_c[2,2]/sum(table_DT_test_c[2,]) +
                  table_DT_test_c[1,1]/sum(table_DT_test_c[1,]))
  m2_DT_test <- c(m2_DT_test,sum(diag(table_DT_test_c))/sum(table_DT_test_c))
}

cutoffs_DT_test <- quantile(pred_DT_test,seq(0.25,1,0.25))
quartiles_DT_test <- ifelse(pred_DT_test<=cutoffs_DT_test[1],"Q1",
                            ifelse(pred_DT_test<=cutoffs_DT_test[2],"Q2",
                                   ifelse(pred_DT_test<=cutoffs_DT_test[3],"Q3","Q4")))

results_optimals_test[1,"DT"] <- max(m1_DT_test)
results_optimals_test[2,"DT"] <- max(m2_DT_test)
results_optimals_test[3,"DT"] <- auc(roc(response_test,pred_DT_test))
results_optimals_test[4,"DT"] <- mean(response_test[quartiles_DT_test=="Q1"]==1)
results_optimals_test[5,"DT"] <- mean(response_test[quartiles_DT_test=="Q2"]==1)
results_optimals_test[6,"DT"] <- mean(response_test[quartiles_DT_test=="Q3"]==1)
results_optimals_test[7,"DT"] <- mean(response_test[quartiles_DT_test=="Q4"]==1)
results_optimals_test[8,"DT"] <- seq(lowest,highest,0.01)[which(m1_DT_test==max(m1_DT_test))[1]]
results_optimals_test[9,"DT"] <- seq(lowest,highest,0.01)[which(m2_DT_test==max(m2_DT_test))[1]]
results_optimals_test[10,"DT"] <- cutoffs_DT_test[1]
results_optimals_test[11,"DT"] <- cutoffs_DT_test[2]
results_optimals_test[12,"DT"] <- cutoffs_DT_test[3]
results_optimals_test[13,"DT"] <- cutoffs_DT_test[4]

# 7.3 RANDOM FOREST     ##################################################################################

# Train the optimal model with all training set
model_RF <- ml_random_forest(data_partitions$train,
                             type="classification",
                             response=response,
                             features=c(features_num,features_cat),
                             max_bins=30,
                             max_depth=5,
                             num_trees=30,
                             min_instances_per_node=9,
                             subsampling_rate=1,
                             seed=1)

# Features importance
ml_feature_importances(model_RF)

# Predict training data
pred_RF_train <- sdf_predict(data_partitions$train,model_RF)
pred_RF_train <- data.frame(collect(pred_RF_train %>% select(probability_1)))[,"probability_1"]

# Predict test data
pred_RF_test <- sdf_predict(data_partitions$test,model_RF)
pred_RF_test <- data.frame(collect(pred_RF_test %>% select(probability_1)))[,"probability_1"]

# Iterate all possible cut-offs and calculate performance measures in training set
m1_RF_train <- c()
m2_RF_train <- c()
lowest <- trunc(min(pred_RF_train)*100)/100+0.01
highest <- trunc(max(pred_RF_train)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_RF_train_c <- table(Real=response_train,Predicted=ifelse(pred_RF_train>=c,"1","0"))
  m1_RF_train <- c(m1_RF_train,table_RF_train_c[2,2]/sum(table_RF_train_c[2,]) +
                   table_RF_train_c[1,1]/sum(table_RF_train_c[1,]))
  m2_RF_train <- c(m2_RF_train,sum(diag(table_RF_train_c))/sum(table_RF_train_c))
}

cutoffs_RF_train <- quantile(pred_RF_train,seq(0.25,1,0.25))
quartiles_RF_train <- ifelse(pred_RF_train<=cutoffs_RF_train[1],"Q1",
```

```r
                             ifelse(pred_RF_train<=cutoffs_RF_train[2],"Q2",
                                    ifelse(pred_RF_train<=cutoffs_RF_train[3],"Q3","Q4")))

results_optimals_train[1,"RF"] <- max(m1_RF_train)
results_optimals_train[2,"RF"] <- max(m2_RF_train)
results_optimals_train[3,"RF"] <- auc(roc(response_train,pred_RF_train))
results_optimals_train[4,"RF"] <- mean(response_train[quartiles_RF_train=="Q1"]==1)
results_optimals_train[5,"RF"] <- mean(response_train[quartiles_RF_train=="Q2"]==1)
results_optimals_train[6,"RF"] <- mean(response_train[quartiles_RF_train=="Q3"]==1)
results_optimals_train[7,"RF"] <- mean(response_train[quartiles_RF_train=="Q4"]==1)
results_optimals_train[8,"RF"] <- seq(lowest,highest,0.01)[which(m1_RF_train==max(m1_RF_train))[1]]
results_optimals_train[9,"RF"] <- seq(lowest,highest,0.01)[which(m2_RF_train==max(m2_RF_train))[1]]
results_optimals_train[10,"RF"] <- cutoffs_RF_train[1]
results_optimals_train[11,"RF"] <- cutoffs_RF_train[2]
results_optimals_train[12,"RF"] <- cutoffs_RF_train[3]
results_optimals_train[13,"RF"] <- cutoffs_RF_train[4]

# Iterate all possible cut-offs and calculate performance measures in test set
m1_RF_test <- c()
m2_RF_test <- c()
lowest <- trunc(min(pred_RF_test)*100)/100+0.01
highest <- trunc(max(pred_RF_test)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_RF_test_c <- table(Real=response_test,Predicted=ifelse(pred_RF_test>=c,"1","0"))
  m1_RF_test <- c(m1_RF_test,table_RF_test_c[2,2]/sum(table_RF_test_c[2,]) +
                  table_RF_test_c[1,1]/sum(table_RF_test_c[1,]))
  m2_RF_test <- c(m2_RF_test,sum(diag(table_RF_test_c))/sum(table_RF_test_c))
}

cutoffs_RF_test <- quantile(pred_RF_test,seq(0.25,1,0.25))
quartiles_RF_test <- ifelse(pred_RF_test<=cutoffs_RF_test[1],"Q1",
                            ifelse(pred_RF_test<=cutoffs_RF_test[2],"Q2",
                                   ifelse(pred_RF_test<=cutoffs_RF_test[3],"Q3","Q4")))

results_optimals_test[1,"RF"] <- max(m1_RF_test)
results_optimals_test[2,"RF"] <- max(m2_RF_test)
results_optimals_test[3,"RF"] <- auc(roc(response_test,pred_RF_test))
results_optimals_test[4,"RF"] <- mean(response_test[quartiles_RF_test=="Q1"]==1)
results_optimals_test[5,"RF"] <- mean(response_test[quartiles_RF_test=="Q2"]==1)
results_optimals_test[6,"RF"] <- mean(response_test[quartiles_RF_test=="Q3"]==1)
results_optimals_test[7,"RF"] <- mean(response_test[quartiles_RF_test=="Q4"]==1)
results_optimals_test[8,"RF"] <- seq(lowest,highest,0.01)[which(m1_RF_test==max(m1_RF_test))[1]]
results_optimals_test[9,"RF"] <- seq(lowest,highest,0.01)[which(m2_RF_test==max(m2_RF_test))[1]]
results_optimals_test[10,"RF"] <- cutoffs_RF_test[1]
results_optimals_test[11,"RF"] <- cutoffs_RF_test[2]
results_optimals_test[12,"RF"] <- cutoffs_RF_test[3]
results_optimals_test[13,"RF"] <- cutoffs_RF_test[4]

# 7.4  GRADIENT BOOSTED TREES     #########################################################################

# Train the optimal model with all training set
formula <- as.formula(paste(response,"~",paste(c(features_num,features_cat),collapse="+")))
model_GBT <- spark.gbt(data=train_SparkR,
                       formula=formula,
                       type="classification",
                       maxDepth=10,
                       maxIter=15,
                       stepSize=0.1,
                       minInstancesPerNode=1,
                       seed=1)

# Predict training data
pred_GBT_train <- predict(model_GBT,train_SparkR)
pred_GBT_train <- unlist(lapply(as.data.frame(pred_GBT_train)[,"probability"],
                                function(x)SparkR:::callJMethod(x,"toArray")[[2]]))

# Predict test data
pred_GBT_test <- predict(model_GBT,test_SparkR)
pred_GBT_test <- unlist(lapply(as.data.frame(pred_GBT_test)[,"probability"],
                               function(x)SparkR:::callJMethod(x,"toArray")[[2]]))

# Iterate all possible cut-offs and calculate performance measures in training set
m1_GBT_train <- c()
m2_GBT_train <- c()
lowest <- trunc(min(pred_GBT_train)*100)/100+0.01
highest <- trunc(max(pred_GBT_train)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_GBT_train_c <- table(Real=response_train,Predicted=ifelse(pred_GBT_train>=c,"1","0"))
  m1_GBT_train <- c(m1_GBT_train,table_GBT_train_c[2,2]/sum(table_GBT_train_c[2,]) +
                    table_GBT_train_c[1,1]/sum(table_GBT_train_c[1,]))
  m2_GBT_train <- c(m2_GBT_train,sum(diag(table_GBT_train_c))/sum(table_GBT_train_c))
}

cutoffs_GBT_train <- quantile(pred_GBT_train,seq(0.25,1,0.25))
quartiles_GBT_train <- ifelse(pred_GBT_train<=cutoffs_GBT_train[1],"Q1",
                             ifelse(pred_GBT_train<=cutoffs_GBT_train[2],"Q2",
                                    ifelse(pred_GBT_train<=cutoffs_GBT_train[3],"Q3","Q4")))

results_optimals_train[1,"GBT"] <- max(m1_GBT_train)
results_optimals_train[2,"GBT"] <- max(m2_GBT_train)
results_optimals_train[3,"GBT"] <- auc(roc(response_train,pred_GBT_train))
results_optimals_train[4,"GBT"] <- mean(response_train[quartiles_GBT_train=="Q1"]==1)
results_optimals_train[5,"GBT"] <- mean(response_train[quartiles_GBT_train=="Q2"]==1)
results_optimals_train[6,"GBT"] <- mean(response_train[quartiles_GBT_train=="Q3"]==1)
results_optimals_train[7,"GBT"] <- mean(response_train[quartiles_GBT_train=="Q4"]==1)
results_optimals_train[8,"GBT"] <- seq(lowest,highest,0.01)[which(m1_GBT_train==max(m1_GBT_train))[1]]
results_optimals_train[9,"GBT"] <- seq(lowest,highest,0.01)[which(m2_GBT_train==max(m2_GBT_train))[1]]
results_optimals_train[10,"GBT"] <- cutoffs_GBT_train[1]
results_optimals_train[11,"GBT"] <- cutoffs_GBT_train[2]
results_optimals_train[12,"GBT"] <- cutoffs_GBT_train[3]
results_optimals_train[13,"GBT"] <- cutoffs_GBT_train[4]

# Iterate all possible cut-offs and calculate performance measures in test set
m1_GBT_test <- c()
m2_GBT_test <- c()
lowest <- trunc(min(pred_GBT_test)*100)/100+0.01
highest <- trunc(max(pred_GBT_test)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_GBT_test_c <- table(Real=response_test,Predicted=ifelse(pred_GBT_test>=c,"1","0"))
  m1_GBT_test <- c(m1_GBT_test,table_GBT_test_c[2,2]/sum(table_GBT_test_c[2,]) +
                  table_GBT_test_c[1,1]/sum(table_GBT_test_c[1,]))
  m2_GBT_test <- c(m2_GBT_test,sum(diag(table_GBT_test_c))/sum(table_GBT_test_c))
}

cutoffs_GBT_test <- quantile(pred_GBT_test,seq(0.25,1,0.25))
quartiles_GBT_test <- ifelse(pred_GBT_test<=cutoffs_GBT_test[1],"Q1",
                            ifelse(pred_GBT_test<=cutoffs_GBT_test[2],"Q2",
                                   ifelse(pred_GBT_test<=cutoffs_GBT_test[3],"Q3","Q4")))

results_optimals_test[1,"GBT"] <- max(m1_GBT_test)
results_optimals_test[2,"GBT"] <- max(m2_GBT_test)
results_optimals_test[3,"GBT"] <- auc(roc(response_test,pred_GBT_test))
results_optimals_test[4,"GBT"] <- mean(response_test[quartiles_GBT_test=="Q1"]==1)
results_optimals_test[5,"GBT"] <- mean(response_test[quartiles_GBT_test=="Q2"]==1)
results_optimals_test[6,"GBT"] <- mean(response_test[quartiles_GBT_test=="Q3"]==1)
results_optimals_test[7,"GBT"] <- mean(response_test[quartiles_GBT_test=="Q4"]==1)
results_optimals_test[8,"GBT"] <- seq(lowest,highest,0.01)[which(m1_GBT_test==max(m1_GBT_test))[1]]
results_optimals_test[9,"GBT"] <- seq(lowest,highest,0.01)[which(m2_GBT_test==max(m2_GBT_test))[1]]
results_optimals_test[10,"GBT"] <- cutoffs_GBT_test[1]
results_optimals_test[11,"GBT"] <- cutoffs_GBT_test[2]
results_optimals_test[12,"GBT"] <- cutoffs_GBT_test[3]
results_optimals_test[13,"GBT"] <- cutoffs_GBT_test[4]

# 7.5 NAIVE BAYES     #####################################################################################

# Train the optimal model with all training set
model_NB <- ml_naive_bayes(data_partitions$train,
                           response=response,
                           features=features_cat)

# Predict training data
pred_NB_train <- sdf_predict(data_partitions$train,model_NB)
pred_NB_train <- data.frame(collect(pred_NB_train %>% select(probability_1)))[,"probability_1"]

# Predict test data
pred_NB_test <- sdf_predict(data_partitions$test,model_NB)
pred_NB_test <- data.frame(collect(pred_NB_test %>% select(probability_1)))[,"probability_1"]

# Iterate all possible cut-offs and calculate performance measures in training set
m1_NB_train <- c()
m2_NB_train <- c()
lowest <- trunc(min(pred_NB_train)*100)/100+0.01
highest <- trunc(max(pred_NB_train)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_NB_train_c <- table(Real=response_train,Predicted=ifelse(pred_NB_train>=c,"1","0"))
  m1_NB_train <- c(m1_NB_train,table_NB_train_c[2,2]/sum(table_NB_train_c[2,]) +
                   table_NB_train_c[1,1]/sum(table_NB_train_c[1,]))
  m2_NB_train <- c(m2_NB_train,sum(diag(table_NB_train_c))/sum(table_NB_train_c))
}

cutoffs_NB_train <- quantile(pred_NB_train,seq(0.25,1,0.25))
quartiles_NB_train <- ifelse(pred_NB_train<=cutoffs_NB_train[1],"Q1",
                             ifelse(pred_NB_train<=cutoffs_NB_train[2],"Q2",
                                    ifelse(pred_NB_train<=cutoffs_NB_train[3],"Q3","Q4")))

results_optimals_train[1,"NB"] <- max(m1_NB_train)
results_optimals_train[2,"NB"] <- max(m2_NB_train)
results_optimals_train[3,"NB"] <- auc(roc(response_train,pred_NB_train))
results_optimals_train[4,"NB"] <- mean(response_train[quartiles_NB_train=="Q1"]==1)
results_optimals_train[5,"NB"] <- mean(response_train[quartiles_NB_train=="Q2"]==1)
results_optimals_train[6,"NB"] <- mean(response_train[quartiles_NB_train=="Q3"]==1)
results_optimals_train[7,"NB"] <- mean(response_train[quartiles_NB_train=="Q4"]==1)
```

```
results_optimals_train[8,"NB"] <- seq(lowest,highest,0.01)[which(m1_NB_train==max(m1_NB_train))[1]]
results_optimals_train[9,"NB"] <- seq(lowest,highest,0.01)[which(m2_NB_train==max(m2_NB_train))[1]]
results_optimals_train[10,"NB"] <- cutoffs_NB_train[1]
results_optimals_train[11,"NB"] <- cutoffs_NB_train[2]
results_optimals_train[12,"NB"] <- cutoffs_NB_train[3]
results_optimals_train[13,"NB"] <- cutoffs_NB_train[4]

# Iterate all possible cut-offs and calculate performance measures in test set
m1_NB_test <- c()
m2_NB_test <- c()
lowest <- trunc(min(pred_NB_test)*100)/100+0.01
highest <- trunc(max(pred_NB_test)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_NB_test_c <- table(Real=response_test,Predicted=ifelse(pred_NB_test>=c,"1","0"))
  m1_NB_test <- c(m1_NB_test,table_NB_test_c[2,2]/sum(table_NB_test_c[2,]) +
                  table_NB_test_c[1,1]/sum(table_NB_test_c[1,]))
  m2_NB_test <- c(m2_NB_test,sum(diag(table_NB_test_c))/sum(table_NB_test_c))
}

cutoffs_NB_test <- quantile(pred_NB_test,seq(0.25,1,0.25))
quartiles_NB_test <- ifelse(pred_NB_test<=cutoffs_NB_test[1],"Q1",
                            ifelse(pred_NB_test<=cutoffs_NB_test[2],"Q2",
                                   ifelse(pred_NB_test<=cutoffs_NB_test[3],"Q3","Q4")))

results_optimals_test[1,"NB"] <- max(m1_NB_test)
results_optimals_test[2,"NB"] <- max(m2_NB_test)
results_optimals_test[3,"NB"] <- auc(roc(response_test,pred_NB_test))
results_optimals_test[4,"NB"] <- mean(response_test[quartiles_NB_test=="Q1"]==1)
results_optimals_test[5,"NB"] <- mean(response_test[quartiles_NB_test=="Q2"]==1)
results_optimals_test[6,"NB"] <- mean(response_test[quartiles_NB_test=="Q3"]==1)
results_optimals_test[7,"NB"] <- mean(response_test[quartiles_NB_test=="Q4"]==1)
results_optimals_test[8,"NB"] <- seq(lowest,highest,0.01)[which(m1_NB_test==max(m1_NB_test))[1]]
results_optimals_test[9,"NB"] <- seq(lowest,highest,0.01)[which(m2_NB_test==max(m2_NB_test))[1]]
results_optimals_test[10,"NB"] <- cutoffs_NB_test[1]
results_optimals_test[11,"NB"] <- cutoffs_NB_test[2]
results_optimals_test[12,"NB"] <- cutoffs_NB_test[3]
results_optimals_test[13,"NB"] <- cutoffs_NB_test[4]

# 7.6 NEURAL NETWORK      ################################################################################

# Train the optimal model with all training set
layer_output <- 2
layer_hidden_1 <- 12
layer_hidden_2 <- 4
layer_input <- sum(sapply(as.data.frame(train_SparkR[,features_cat]),function(x)length(unique(x))-1),
                length(features_num))
formula <- as.formula(paste(response,"~",paste(c(features_num,features_cat),collapse="+")))
model_NN <- spark.mlp(train_SparkR,
                      formula=formula,
                      layers=c(layer_input,layer_hidden_1,layer_hidden_2,layer_output),
                      seed=1)

# Predict training data
pred_NN_train <- predict(model_NN,train_SparkR)
pred_NN_train <- unlist(lapply(as.data.frame(pred_NN_train)[,"probability"],
                               function(x)SparkR:::callJMethod(x,"toArray")[[2]]))
# Predict test data
pred_NN_test <- predict(model_NN,test_SparkR)
pred_NN_test <- unlist(lapply(as.data.frame(pred_NN_test)[,"probability"],
                              function(x)SparkR:::callJMethod(x,"toArray")[[2]]))

# Iterate all possible cut-offs and calculate performance measures in training set
m1_NN_train <- c()
m2_NN_train <- c()
lowest <- trunc(min(pred_NN_train)*100)/100+0.01
highest <- trunc(max(pred_NN_train)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_NN_train_c <- table(Real=response_train,Predicted=ifelse(pred_NN_train>=c,"1","0"))
  m1_NN_train <- c(m1_NN_train,table_NN_train_c[2,2]/sum(table_NN_train_c[2,]) +
                   table_NN_train_c[1,1]/sum(table_NN_train_c[1,]))
  m2_NN_train <- c(m2_NN_train,sum(diag(table_NN_train_c))/sum(table_NN_train_c))
}

cutoffs_NN_train <- quantile(pred_NN_train,seq(0.25,1,0.25))
quartiles_NN_train <- ifelse(pred_NN_train<=cutoffs_NN_train[1],"Q1",
                             ifelse(pred_NN_train<=cutoffs_NN_train[2],"Q2",
                                    ifelse(pred_NN_train<=cutoffs_NN_train[3],"Q3","Q4")))

results_optimals_train[1,"NN"] <- max(m1_NN_train)
results_optimals_train[2,"NN"] <- max(m2_NN_train)
results_optimals_train[3,"NN"] <- auc(roc(response_train,pred_NN_train))
results_optimals_train[4,"NN"] <- mean(response_train[quartiles_NN_train=="Q1"]==1)
results_optimals_train[5,"NN"] <- mean(response_train[quartiles_NN_train=="Q2"]==1)
results_optimals_train[6,"NN"] <- mean(response_train[quartiles_NN_train=="Q3"]==1)
results_optimals_train[7,"NN"] <- mean(response_train[quartiles_NN_train=="Q4"]==1)
results_optimals_train[8,"NN"] <- seq(lowest,highest,0.01)[which(m1_NN_train==max(m1_NN_train))[1]]
```

```r
results_optimals_train[9,"NN"] <- seq(lowest,highest,0.01)[which(m2_NN_train==max(m2_NN_train))[1]]
results_optimals_train[10,"NN"] <- cutoffs_NN_train[1]
results_optimals_train[11,"NN"] <- cutoffs_NN_train[2]
results_optimals_train[12,"NN"] <- cutoffs_NN_train[3]
results_optimals_train[13,"NN"] <- cutoffs_NN_train[4]

# Iterate all possible cut-offs and calculate performance measures in test set
m1_NN_test <- c()
m2_NN_test <- c()
lowest <- trunc(min(pred_NN_test)*100)/100+0.01
highest <- trunc(max(pred_NN_test)*100)/100

for(c in seq(lowest,highest,0.01)){
  table_NN_test_c <- table(Real=response_test,Predicted=ifelse(pred_NN_test>=c,"1","0"))
  m1_NN_test <- c(m1_NN_test,table_NN_test_c[2,2]/sum(table_NN_test_c[2,]) +
                  table_NN_test_c[1,1]/sum(table_NN_test_c[1,]))
  m2_NN_test <- c(m2_NN_test,sum(diag(table_NN_test_c))/sum(table_NN_test_c))
}

cutoffs_NN_test <- quantile(pred_NN_test,seq(0.25,1,0.25))
quartiles_NN_test <- ifelse(pred_NN_test<=cutoffs_NN_test[1],"Q1",
                            ifelse(pred_NN_test<=cutoffs_NN_test[2],"Q2",
                                   ifelse(pred_NN_test<=cutoffs_NN_test[3],"Q3","Q4")))

results_optimals_test[1,"NN"] <- max(m1_NN_test)
results_optimals_test[2,"NN"] <- max(m2_NN_test)
results_optimals_test[3,"NN"] <- auc(roc(response_test,pred_NN_test))
results_optimals_test[4,"NN"] <- mean(response_test[quartiles_NN_test=="Q1"]==1)
results_optimals_test[5,"NN"] <- mean(response_test[quartiles_NN_test=="Q2"]==1)
results_optimals_test[6,"NN"] <- mean(response_test[quartiles_NN_test=="Q3"]==1)
results_optimals_test[7,"NN"] <- mean(response_test[quartiles_NN_test=="Q4"]==1)
results_optimals_test[8,"NN"] <- seq(lowest,highest,0.01)[which(m1_NN_test==max(m1_NN_test))[1]]
results_optimals_test[9,"NN"] <- seq(lowest,highest,0.01)[which(m2_NN_test==max(m2_NN_test))[1]]
results_optimals_test[10,"NN"] <- cutoffs_NN_test[1]
results_optimals_test[11,"NN"] <- cutoffs_NN_test[2]
results_optimals_test[12,"NN"] <- cutoffs_NN_test[3]
results_optimals_test[13,"NN"] <- cutoffs_NN_test[4]

#####################################################################################################

# 8. PROTOCOL OF MODEL VALIDATION PHASE 4: Compare the models with performance measures

# Save and print the results table of the training set
save(results_optimals_train,file="results_optimals_train.RData")
results_optimals_train

# Save and print the results table of the test set
save(results_optimals_test,file="results_optimals_test.RData")
results_optimals_test

# 8.1 PRINCIPAL COMPONENTS ANALYSIS     ##################################################################

# Merge the training and test sets
data_pca <- rbind(data_partitions$train,data_partitions$test)

# Normalize numerical variables for PCA
data_pca <- data_pca %>% mutate(
  Amount=(Amount-mean(Amount))/sd(Amount),
  Maturity=(Maturity-mean(Maturity))/sd(Maturity),
  Postal_Code_ASNEF=(Postal_Code_ASNEF-mean(Postal_Code_ASNEF))/sd(Postal_Code_ASNEF),
  Age=(Age-mean(Age))/sd(Age),
  Seniority=(Seniority-mean(Seniority))/sd(Seniority),
  Housing_Seniority=(Housing_Seniority-mean(Housing_Seniority))/sd(Housing_Seniority),
  Income=(Income-mean(Income))/sd(Income),
  Additional_Income=(Additional_Income-mean(Additional_Income))/sd(Additional_Income),
  Rent=(Rent-mean(Rent))/sd(Rent),
  Partner_Income=(Partner_Income-mean(Partner_Income))/sd(Partner_Income),
  Mortgage=(Mortgage-mean(Mortgage))/sd(Mortgage),
  Amount_of_Ongoing_Credits=(Amount_of_Ongoing_Credits-mean(Amount_of_Ongoing_Credits))/
    sd(Amount_of_Ongoing_Credits))

# Run PCA and project every observation into the new space
pca <- data_pca %>% ml_pca(k=2,features=features_num)
pca_projections <- as.data.frame(pca %>% sdf_project() %>% select(PC1,PC2))

# Plot the 2 main principal components
ggplot(as.data.frame(pca$pc),aes(PC1,PC2)) +
  geom_text(aes(label=row.names(pca$pc)),size=3,alpha=1,vjust =-1,hjust=0.5) +
  geom_segment(aes(x=0,xend=PC1,y=0,yend=PC2),arrow=arrow(length=unit(0.3,"cm")),col="darkblue") +
  labs(title="PCA",
       x=paste("PC1:",round(pca$explained_variance[1]*100,2),"% projected variance"),
       y=paste("PC2:",round(pca$explained_variance[2]*100,2),"% projected variance"))

# Plot PCA projections coloured by predictions of the model Logistic Regression
Prediction_LR <- c(pred_LR_train,pred_LR_test)
ggplot(pca_projections,aes(PC1,PC2,color=Prediction_LR)) +
  geom_point() + scale_colour_gradient(low="green",high="red") +
  xlim(-5,5) + ylim(-5,4) +
  labs(title="PCA Projections: Model Logistic Regression",
       x=paste("PC1:",round(pca$explained_variance[1]*100,2),"% variance"),
       y=paste("PC2:",round(pca$explained_variance[2]*100,2),"% variance"))

# Plot PCA projections coloured by predictions of the model Decision Tree
Prediction_DT <- c(pred_DT_train,pred_DT_test)
ggplot(pca_projections,aes(PC1,PC2,color=Prediction_DT)) +
  geom_point() + scale_colour_gradient(low="green",high="red") +
  xlim(-5,5) + ylim(-5,4) +
  labs(title="PCA Projections: Model Decision Tree",
       x=paste("PC1:",round(pca$explained_variance[1]*100,2),"% variance"),
       y=paste("PC2:",round(pca$explained_variance[2]*100,2),"% variance"))

# Plot PCA projections coloured by predictions of the model Random Forest
Prediction_RF <- c(pred_RF_train,pred_RF_test)
ggplot(pca_projections,aes(PC1,PC2,color=Prediction_RF)) +
  geom_point() + scale_colour_gradient(low="green",high="red") +
  xlim(-5,5) + ylim(-5,4) +
  labs(title="PCA Projections: Model Random Forest",
       x=paste("PC1:",round(pca$explained_variance[1]*100,2),"% variance"),
       y=paste("PC2:",round(pca$explained_variance[2]*100,2),"% variance"))

# Plot PCA projections coloured by predictions of the model Gradient Boosted Trees
Prediction_GBT <- c(pred_GBT_train,pred_GBT_test)
ggplot(pca_projections,aes(PC1,PC2,color=Prediction_GBT)) +
  geom_point() + scale_colour_gradient(low="green",high="red") +
  xlim(-5,5) + ylim(-5,4) +
  labs(title="PCA Projections: Model Gradient Boosted Trees",
       x=paste("PC1:",round(pca$explained_variance[1]*100,2),"% variance"),
       y=paste("PC2:",round(pca$explained_variance[2]*100,2),"% variance"))

# Plot PCA projections coloured by predictions of the model Naive Bayes
Prediction_NB <- c(pred_NB_train,pred_NB_test)
ggplot(pca_projections,aes(PC1,PC2,color=Prediction_NB)) +
  geom_point() + scale_colour_gradient(low="green",high="red") +
  xlim(-5,5) + ylim(-5,4) +
  labs(title="PCA Projections: Model Naive Bayes",
       x=paste("PC1:",round(pca$explained_variance[1]*100,2),"% variance"),
       y=paste("PC2:",round(pca$explained_variance[2]*100,2),"% variance"))

# Plot PCA projections coloured by predictions of the model Multilayer Perceptron
Prediction_NN <- c(pred_NN_train,pred_NN_test)
ggplot(pca_projections,aes(PC1,PC2,color=Prediction_NN)) +
  geom_point() + scale_colour_gradient(low="green",high="red") +
  xlim(-5,5) + ylim(-5,4) +
  labs(title="PCA Projections: Model Multilayer Perceptron",
       x=paste("PC1:",round(pca$explained_variance[1]*100,2),"% variance"),
       y=paste("PC2:",round(pca$explained_variance[2]*100,2),"% variance"))

#####################################################################################################

# 9. DISCONNECTION OF SPARK CLUSTERS

# Disconnect the Spark session of both "SparkR" and "sparklyr" packages
spark_disconnect(sc_sparklyr)
sparkR.session.stop()
```

8.4. Models feature importance

8.4.1. *Logistic regression*

Variable	Estimated Coefficient
Postal_Code_ASNEF	2.0552
Profession_Code_MIDDLEGRADEMANAGER	0.9489
Profession_Code_ADMINISTRATIVE	0.8904
Profession_Code_OPERATOR	0.8643
Profession_Code_OTHERS	0.7602
Profession_Code_TECHNICIAN	0.7396
Num_Ongoing_Credits_3	-0.6847
Contract_Type_PERMANENT	-0.6771
Province_Barcelona	-0.5195
(Intercept)	-0.4850
Application_Week_Day_4	0.4839
Contract_Type_PENSION	-0.4685
Purpose_USEDCAR	0.4653
Purpose_DEBTS	-0.4370
Province_OTHERS	-0.4093
Housing_Type_TENANT	0.3966
Purpose_FURNITURE_AND_APPLIANCES	0.3779
Purpose_LIQUIDITY	0.3421
Purpose_HOMEIMPROVEMENT	0.3276
Marital_Status_COHABITING	-0.3237
Application_Week_Day_5	0.3188
Application_Week_Day_6	0.2716
Housing_Type_HOME_OWNERSHIP_WITHOUT_MORTGAGE	0.2704
Province_Madrid	-0.2404
Num_Ongoing_Credits_2	-0.2351
Application_Week_Day_1	0.2277
Application_Hour_Group_[20H,23H)	-0.2204
Marital_Status_SINGLE	-0.2041
Housing_Type_HOME_OWNERSHIP_WITH_MORTGAGE	0.1839
Num_Ongoing_Credits_1	-0.1830
Num_Ongoing_Credits_0	0.1159
Marital_Status_DIVORCED	-0.1133
People_in_Household_0	0.1031
Marital_Status_MARRIED	-0.0999
Application_Week_Day_2	0.0988
Housing_Type_THIRD_PARTY_PROVIDED_LODGING	0.0956
Application_Week_Day_3	0.0816
Purpose_OTHERS	0.0781
People_in_Household_1	-0.0610
Profession_Sector_PRIVATE_SECTOR	-0.0566
People_in_Household_2	-0.0559
Purpose_VACATION	0.0519
Gender_MALE	0.0511
Age	-0.0197
Application_Hour_Group_[7H,20H)	-0.0084
Maturity	0.0024

Mortgage	-0.0009
Rent	-0.0008
Housing_Seniority	-0.0001
Seniority	0.0001
Amount_of_Ongoing_Credits	-0.0001
Amount	0.0000
Additional_Income	0.0000
Partner_Income	0.0000
Income	0.0000

8.4.2. *Decision tree*

Variable	Importance
Age	0.1594
Amount_of_Ongoing_Credits	0.1282
Amount	0.1129
Housing_Seniority	0.0841
Mortgage	0.0772
Rent	0.0543
Partner_Income	0.0529
Housing_Type_THIRD_PARTY_PROVIDED_LODGING	0.0488
Income	0.0481
Seniority	0.0415
Purpose_DEBTS	0.0412
Application_Week_Day_4	0.0408
Province_Barcelona	0.0380
People_in_Household_0	0.0319
Postal_Code_ASNEF	0.0175
Purpose_HOMEIMPROVEMENT	0.0107
Contract_Type_PERMANENT	0.0076
Num_Ongoing_Credits_1	0.0050
Maturity	0.0000
Additional_Income	0.0000
Application_Hour_Group_[7H, 20H)	0.0000
Application_Hour_Group_[20H, 23H)	0.0000
Application_Week_Day_3	0.0000
Application_Week_Day_1	0.0000
Application_Week_Day_2	0.0000
Application_Week_Day_5	0.0000
Application_Week_Day_6	0.0000
Purpose_LIQUIDITY	0.0000
Purpose_OTHERS	0.0000
Purpose_USEDCAR	0.0000
Purpose_VACATION	0.0000
Purpose_FURNITURE_AND_APPLIANCES	0.0000
Province_OTHERS	0.0000
Province_Madrid	0.0000
Gender_MALE	0.0000
Profession_Code_OTHERS	0.0000
Profession_Code_OPERATOR	0.0000
Profession_Code_ADMINISTRATIVE	0.0000

Profession_Code_TECHNICIAN	0.0000
Profession_Code_MIDDLEGRADEMANAGER	0.0000
Profession_Sector_PRIVATE_SECTOR	0.0000
Contract_Type_PENSION	0.0000
Housing_Type_HOME_OWNERSHIP_WITH_MORTGAGE	0.0000
Housing_Type_HOME_OWNERSHIP_WITHOUT_MORTGAGE	0.0000
Housing_Type_TENANT	0.0000
Marital_Status_MARRIED	0.0000
Marital_Status_SINGLE	0.0000
Marital_Status_DIVORCED	0.0000
Marital_Status_COHABITING	0.0000
People_in_Household_1	0.0000
People_in_Household_2	0.0000
Num_Ongoing_Credits_2	0.0000
Num_Ongoing_Credits_0	0.0000
Num_Ongoing_Credits_3	0.0000

8.4.3. *Random forest*

Variable	Importance
Amount	0.1064
Amount_of_Ongoing_Credits	0.0973
Age	0.0864
Housing_Seniority	0.0570
Seniority	0.0560
Income	0.0534
Postal_Code_ASNEF	0.0532
Mortgage	0.0410
Maturity	0.0302
Purpose_DEBTS	0.0278
Partner_Income	0.0273
Additional_Income	0.0222
Num_Ongoing_Credits_3	0.0201
Housing_Type_THIRD_PARTY_PROVIDED_LODGING	0.0198
People_in_Household_1	0.0172
Rent	0.0171
Housing_Type_HOME_OWNERSHIP_WITH_MORTGAGE	0.0159
Profession_Sector_PRIVATE_SECTOR	0.0150
Num_Ongoing_Credits_0	0.0146
Purpose_USEDCAR	0.0145
Profession_Code_OPERATOR	0.0141
Profession_Code_OTHERS	0.0116
Province_Barcelona	0.0099
Contract_Type_PENSION	0.0095
Application_Week_Day_5	0.0085
Housing_Type_TENANT	0.0084
Gender_MALE	0.0080
Purpose_OTHERS	0.0077
People_in_Household_2	0.0076
Profession_Code_TECHNICIAN	0.0075
Marital_Status_SINGLE	0.0073

Num_Ongoing_Credits_1	0.0070
Application_Week_Day_3	0.0069
Application_Week_Day_4	0.0066
Marital_Status_MARRIED	0.0064
Housing_Type_HOME_OWNERSHIP_WITHOUT_MORTGAGE	0.0064
Application_Week_Day_2	0.0063
People_in_Household_0	0.0061
Province_OTHERS	0.0056
Purpose_HOMEIMPROVEMENT	0.0053
Province_Madrid	0.0051
Purpose_FURNITURE_AND_APPLIANCES	0.0050
Contract_Type_PERMANENT	0.0046
Application_Week_Day_1	0.0046
Purpose_LIQUIDITY	0.0045
Application_Hour_Group_[7H, 20H)	0.0042
Marital_Status_DIVORCED	0.0040
Profession_Code_ADMINISTRATIVE	0.0034
Purpose_VACATION	0.0032
Marital_Status_COHABITING	0.0032
Application_Hour_Group_[20H, 23H)	0.0030
Application_Week_Day_6	0.0024
Profession_Code_MIDDLEGRADEMANAGER	0.0023
Num_Ongoing_Credits_2	0.0009