Multivariate analysis

Final Project

Marketing Campaigns of Portuguese Bank

Denaldo Lapi, Francesco Aristei and Samy Chouiti

June 8, 2022

Contents

List of Figures

List of Tables

1 Introduction

For the Multivariate Analysis course' final project, we decided to work on a classification problem based on the data coming from a marketing campaign of a Portuguese banking institution.

Our goal was to understand the dataset structure and the underlying patterns that make a client subscribing for a term deposit or not. As in any data science project, we first performed an Exploratory Data Analysis (EDA) and then, as a second step, we built several classification algorithms and compare them in order to conclude on how to deliver the best possible results.

2 Dataset

The dataset we used features a direct marketing campaign (through phone calls) of a Portuguese banking institution. During this campaign, 4521 clients were contacted for term deposit commercial propositions, and every contacted individual has 18 features including the predictor variable. The description of each feature can be found in the Table 1. Therefore, our dataset is related to a classification problem under the following question: based on certain features, **will a client subscribe or not a term deposit ?**. More details about the classification problem will be made in a later section 4.

The dataset was taken from *Kaggle*.

Name	Type	Description
Age	Num.	Age of the client
Job	Cat.	Job of the client
Martial Status	Cat.	Marital status of the client
Education	Cat.	Education type of the client
Default	Cat.	If the client has already been in default
Balance	Num.	Balance of the account
Housing	Cat.	Having a housing loan
Loan	Cat.	Having a personal loan
Contact	Cat.	Contacting method
Day	Num.	Last contact day of the week
Month	Num.	Last contact month of the year
Duration	Num.	Duration of last call
Campaign	Num.	Number of contacts (current campaign)
Pdays	Num.	Number of days since last contact (previous campaign)
Previous	Num.	Number of contacts performed (before current campaign)
Poutcome	Cat.	Outcome of the previous marketing campaign
y (Predictor)	**Bin.**	**Did the client subscribed for a term deposit ?**

Table 1: Features description. Campaign, Pdays and Previous are about this same client.

3 Exploratory Data Analysis

The first step of the project was related to understanding the structure and the main patterns of the considered dataset, in order to get some insights on the variables and on how they are related to each other.

3.1 Features specificity

Before analyzing the dataset contents, here are some additional information on certain features values, necessary to understand later conclusions:

- A value of *-1* in the *pdays* variable represents clients who have never been contacted before for marketing campaign
- Most of the values of the variable *poutcome* are *unknown* meaning that the bank doesn't have any data regarding the outcome of previous campaigns for that client. It could be interesting to understand whether the *unknown* values are associated only to *-1* values of the *pdays* variable: if it is the case it means that the bank has no information only about customers who have never been contacted before for a campaign.

3.2 Basic statistics

Some relevant observations that can be drawn from the basic statistic Table 1:

- The dataset represents customers with a medium age of 41 years, which is a pretty realistic scenario in a bank database.
- The *marital* variable shows that the main customers of the bank are families, since most of the *marital* variable values are in the *married* category.
- The variable *balance* assumes also negative values, indicating people with a negative balance in their bank account.
- Variable *default* is highly unbalanced towards the value *yes*, meaning that very few customers have their credit in default.
- The variable *loan* tells us that most of the bank clients don't have any kind of personal loan.
- The variable *previous* is characterized by a very low mean value, meaning that the majority of the clients have been contacted only a few times before this specific campaign.
- For what regard the target variable y, we can see how most of the observations belong to the *no* class, with only 521 observations belonging to the class *yes*.

ID	age	job	marital	education	default	balance	housing	loan	contact
Min. : 1	Min. :19.00	management :969	divorced: 528	primary : 678	no :4445	Min. :-3313	no :1962	no :3830	cellular :2896
1st Qu.:1131	1st Qu.:33.00	blue-collar:946	married :2797	secondary:2306	yes: 76	1st Qu.: 69	yes:2559	yes: 691	telephone: 301
Median :2261	Median :39.00	technician :768	single :1196	tertiary :1350	NA	Median : 444	NA	NA	unknown :1324
Mean :2261	Mean :41.17	admin. :478	NA	unknown : 187	NA	Mean : 1423	NA	NA	NA
3rd Qu.:3391	3rd Qu.:49.00	services :417	NA	NA	NA	3rd Qu.: 1480	NA	NA	NA
Max. :4521	Max. :87.00	retired :230	NA	NA	NA	Max. :71188	NA	NA	NA
NA	NA	(Other) :713	NA	NA	NA	NA	NA	NA	NA

day	month	duration	campaign	pdays	previous	poutcome	y
Min. : 1.00	may :1398	Min. : 4	Min. : 1.000	Min. : -1.00	Min. : 0.0000	failure: 490	no :4000
1st Qu.: 9.00	jul : 706	1st Qu.: 104	1st Qu.: 1.000	1st Qu.: -1.00	1st Qu.: 0.0000	other : 197	yes: 521
Median :16.00	aug : 633	Median : 185	Median : 2.000	Median : -1.00	Median : 0.0000	success: 129	NA
Mean :15.92	jun : 531	Mean : 264	Mean : 2.794	Mean : 39.77	Mean : 0.5426	unknown:3705	NA
3rd Qu.:21.00	nov : 389	3rd Qu.: 329	3rd Qu.: 3.000	3rd Qu.: -1.00	3rd Qu.: 0.0000	NA	NA
Max. :31.00	apr : 293	Max. :3025	Max. :50.000	Max. :871.00	Max. :25.0000	NA	NA
NA	(Other): 571	NA	NA	NA	NA	NA	NA

Figure 1: Basic statistics of the dataset's variables

This last observation is an important characteristic of our dataset. We will have to strongly take into account this class partition when developing our classification models, in order to balance the importance of the 'no' class. More observation about this unbalance problem will be discussed in the following sections.

No missing values were found in the dataset.

3.3 Distribution

As a first step, we will study the distribution of each variable based on the classification of the y variable. This will be useful for a later study of outliers.

From variables distribution plots Figure 2, we made the following observations:

- Is already evident how, for some specific type of occupations, the percentage of adhesion to the term deposit is greater than for other kind of jobs. For example, the individuals performing blue-collar or management jobs, are less likely to adhere to the term deposit, than for example retired people.
- We can observe how people having no housing loan are more prone to subscribe for the term deposit. This observation is coherent given that people who are already under a debt condition, needs more liquidity in the short term and cannot afford to keep it untouched in some kind of deposit instrument.
- Most of the last contacts have happened in the month of may, or more generally, during the summer months, which may indicate also the starting period of the marketing campaign. Interestingly, seems like people contacted in the month immediately after the months described above, have an higher possibility of signing for a term deposit.
- As expected, people who decided to sign for a term deposit in a previous campaign are very prone to re-sign for the current campaign, however, most of the values are *unknown*, meaning that the outcome of the campaign for each client, weren't tracked appropriately by the bank. One thing who should be highlighted, is that there is some ambiguity on the meaning of *other* and *unknown*, specifically, the value *other* seems to be not possible to be interpreted given the information provided, therefore we decided to convert it into *unknown*.

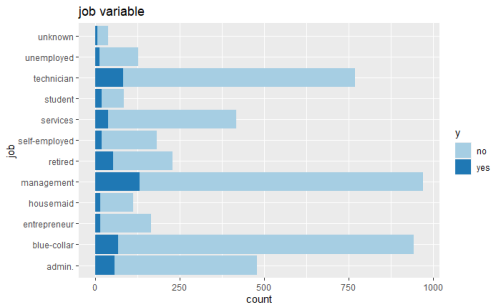
Once the qualitative variables have been inspected, we focused on the distribution and characteristics of the quantitative one. We first started from their distributions, as can be seen in Figure 3.

From the distribution plots Figure 3, we can see:

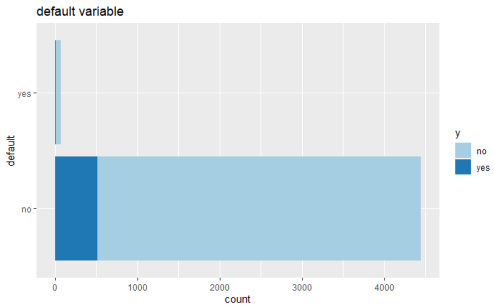
- Most of the clients called by the bank dispose of a bank account with a balance oscillating between 0 to 5000.
- Most of the calls made are done between the second and third week of the month.
- The majority of the calls performed have a really short duration (in second). This may be a crucial variable for understanding the outcome of the campaign for a certain client.
- The majority of the clients are contacted for the first time during this campaign, as can be observed from the distribution of the *pdays* variable (mostly assumes 0 value).

Then we decided to plot the densities of such variables, together with the target variable value.

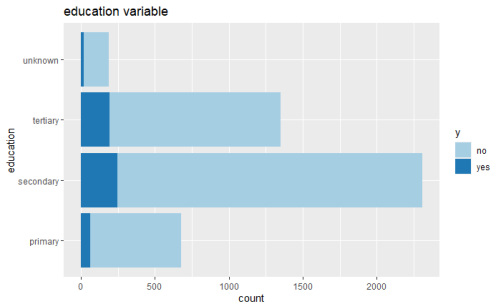
From the densities on Figure 4, here are some insights highlighting the importance of some variables in order to perform the classification task:



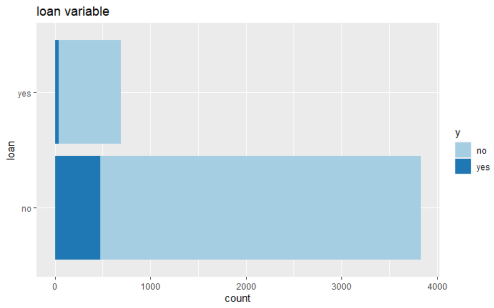
(a) Job variable count



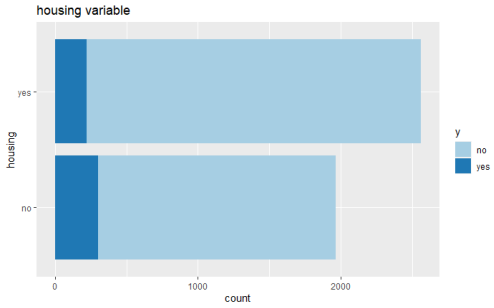
(b) Default variable count



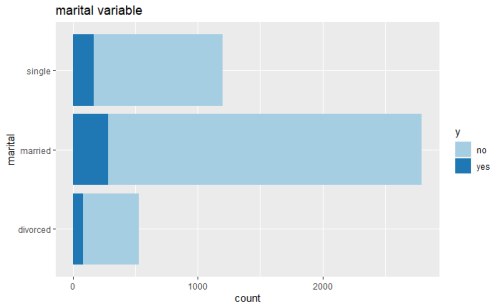
(c) Education variable count



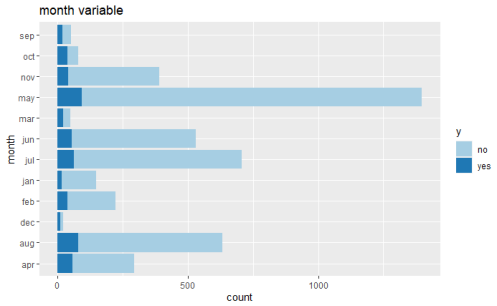
(d) Loan variable count



(e) Housing variable count

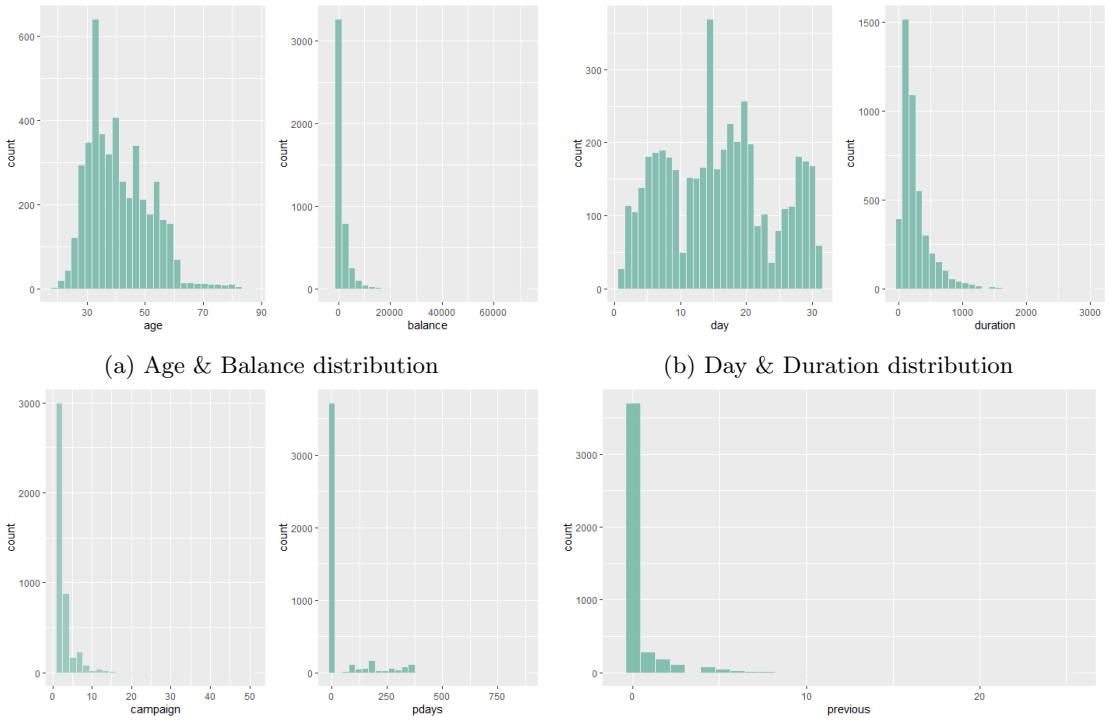


(f) Marital variable count



(g) Month variable count

Figure 2: Categorical variables distribution



(a) Age & Balance distribution

(b) Day & Duration distribution

(c) Camp & Pdays distribution

(d) Previous variable distribution

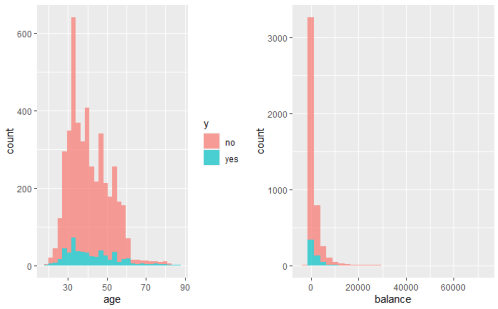
Figure 3: Numerical variables distribution

- As specified before, *duration* variable has a pretty accentuated role in discriminating between the 2 classes of the target variable, the longer the call is, the more likely the client will be subscribing for a term deposit.
- *campaign* distribution is characterized for some small peaks for high values of the variable in corresponding to the class *no*, meaning that the more the client is called, the less it's probable that he will be likely to sign for the term deposit.

Another important step performed is the visualization of the boxplots in Figure 5, which shows the distribution of the values for each continuous attribute.

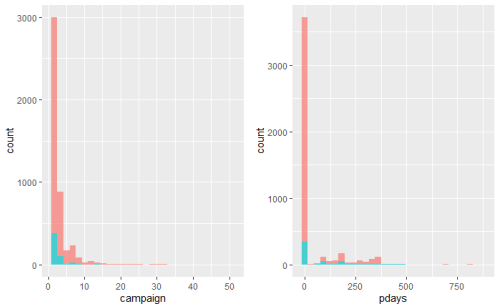
The box plots allowed to come up with more observations on the continuous data analyzed. More particularly:

- For the *balance* variable we can say that we have very sparsed observartions: most of the clients have a very low bank account balance, while we have 2 observations with negative values and other 2 observations with a very high values.
- For what regards the *duration* variable we have 3 samples with very high values of the variable. The distribution of the variable is left skewed, with a very low mean, and there are a lot of values above the inter-quantile range, which we consider significant for the classification task and we will not consider all of them as outliers.

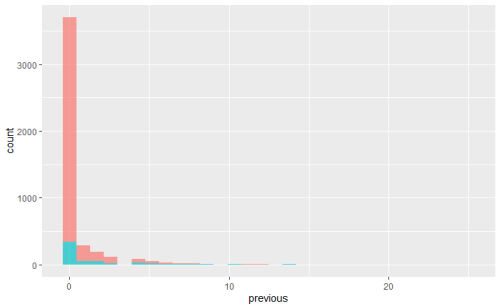


(a) Age & Balance density

(b) Day & Duration density



(c) Campaign & Pdays density



(d) Previous variable density

Figure 4: Numerical variables distribution by class

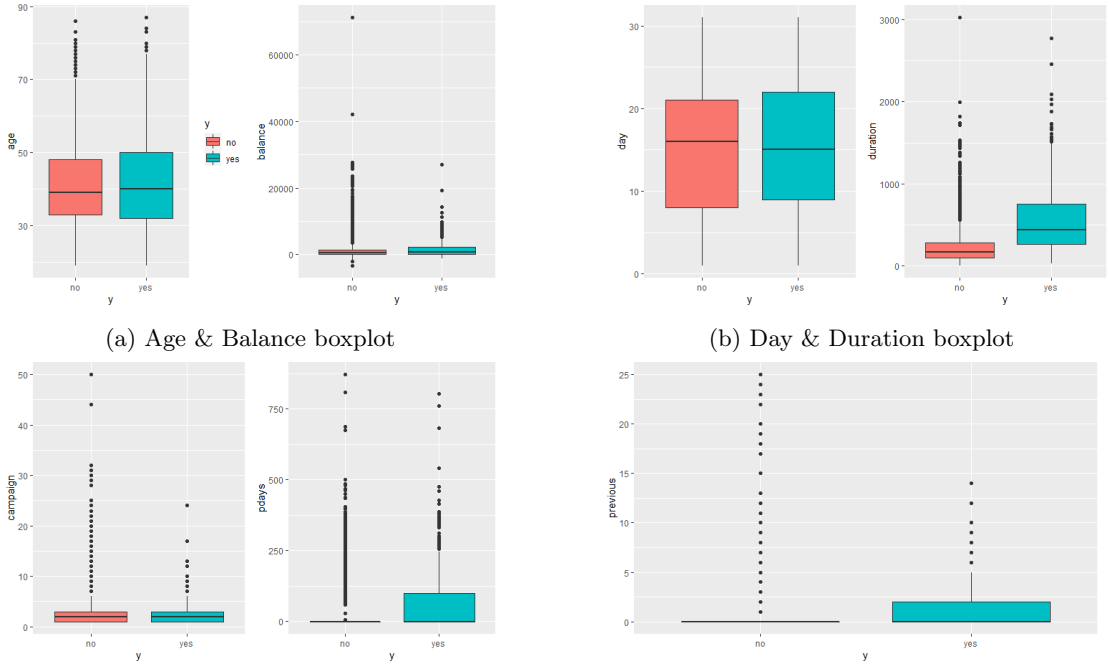
- Also in *campaign* we have some observations with very high value: it seems that there are clients that have been contacted more than 40 times, with respect to an average of around 2.8.

From what we saw in the boxplots, we thought it could have been interesting to see the value on the target variable, for the clients having the highest balance in the bank account, as can be seen in Figure 6.

What we observed is that no one of these clients decided to sign for the term deposit. Regarding the outliers on the balance variable detected above, we decided to rely on the boxplot to spot them in a more accurate way which enabled us to detect 506 observations as outliers. However, we decided to remove only the first two, which already changed the distribution of the variable, as can be seen in Figure 7.

Another variable that needed further inspection is the *campaign* variable, as being a good indicator for predicting the target, since it represent the number of times a client has been contacted during the actual marketing campaign.

What we understood from the boxplots on Figure 5 is that calling many times a client has not a positive effect on their willingness to sign for the term deposit. Therefore, we removed only the first two observations with the highest *campaign* value, since they considered clients called more than 40 times, which is a really high value that cannot be considered realistic, especially considering the mean of such variable (2.8). The distribution of the updated variable can be seen in Figure 8.



(a) Age & Balance boxplot

(b) Day & Duration boxplot

(c) Campaign & Pdays boxplot

(d) Previous variable boxplot

Figure 5: Numerical variables boxplots

The same reasoning done for campaign have been done for *duration*. In fact, as pointed out several times before, we thing that this variable is crucial for discriminating between the 2 classes: an higher call duration indicates an higher probability for the client to apply for the term deposit. We then removed the first three observations with the highest value on *duration*, updating the distribution for this variable, as can be seen in Figure 9.

Correlation Another critical step we performed to study structure of the data, was to plot the correlations among these continuous variables.

On overall, what we can see in the correlation matrix in Figure 10 is that, there are very low correlation values, except for the pair of variables *pdays* and *previous*. With the obtained knowledge on the distribution of the variables, we performed further analysis for each customer characteristics, in order to investigate its influence on the subscription rate, i.e on the rate of subscriptions to the term deposit. At first, we visualized the scatter plot of the variables *campaign* and *duration*, encoding the target class in the plot from Figure 11.

The two variables allow to create approximately two clusters of customers:

- Almost all the *yes* clients are characterized by low value of *campaign* and they may have also high *duration*.
- *no* clients have higher values on *campaign* and low *duration* values.

ID <int>	age <int>	job <fctr>	marital <fctr>	education <fctr>	default <fctr>	balance <int>	housing <fctr>	loan <fctr>
3701	60	retired	married	primary	no	71188	no	no
2990	42	entrepreneur	married	tertiary	no	42045	no	no
1484	43	technician	single	tertiary	no	27733	yes	no
1882	36	management	married	tertiary	no	27359	yes	no
3831	57	technician	married	tertiary	no	27069	no	yes
872	31	housemaid	single	primary	no	26965	no	no
4048	75	retired	married	secondary	no	26452	no	no
3012	50	services	married	secondary	no	26394	no	no
2197	54	management	divorced	tertiary	no	26306	yes	no
1032	49	retired	single	primary	no	25824	no	no

Figure 6: Highest balances clients

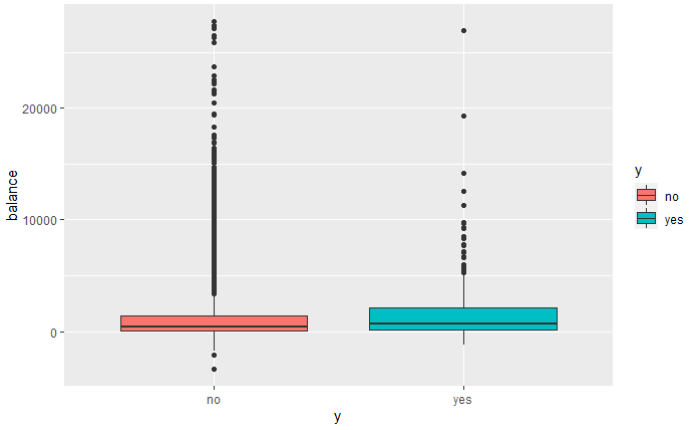


Figure 7: Balance boxplot after outliers removal

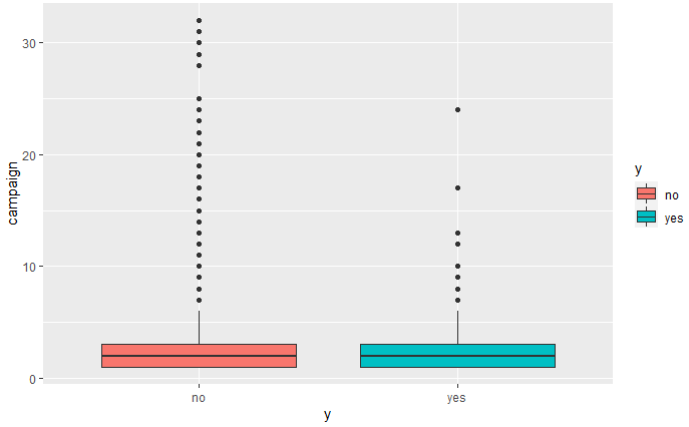


Figure 8: Campaign variable boxplot after outliers removal

An high value of the *campaign* variable makes the clients to refuse to sign for the term deposit: almost all observations with *campaign* value above 10 have *no* as target variable value. When comparing the *contact* variable to the *age*, we found more probable

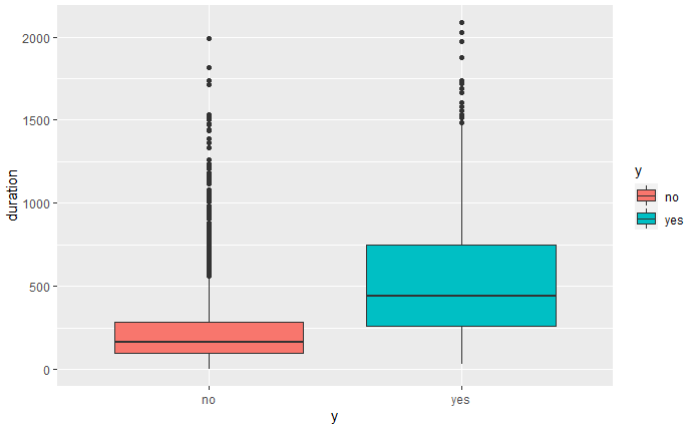


Figure 9: Duration distribution after outliers removal

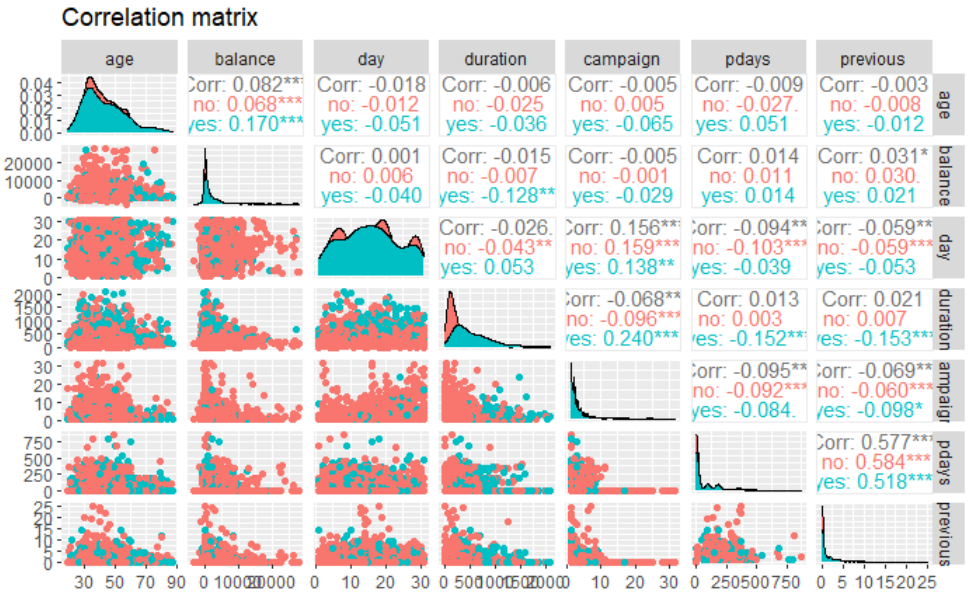


Figure 10: Correlation matrix

that older people are more contacted by telephone rather than cellular, as can be seen in Figure 12.

However, it seems like there is no relationship between the two variables. To demonstrate this insight, we then computed the correlation matrix between the quantitative variables, but as spotted before, only *pdays* and *previous* are positively correlated between each other 10.

As next step in the EDA, we decided to plot the subscription rates depending on different variables. The first plot we performed was the subscription rate per age, as seen on Figure 13.

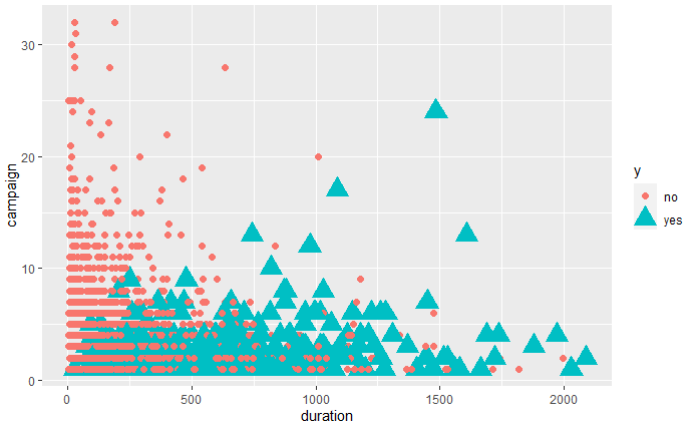


Figure 11: Campaign variable value by duration

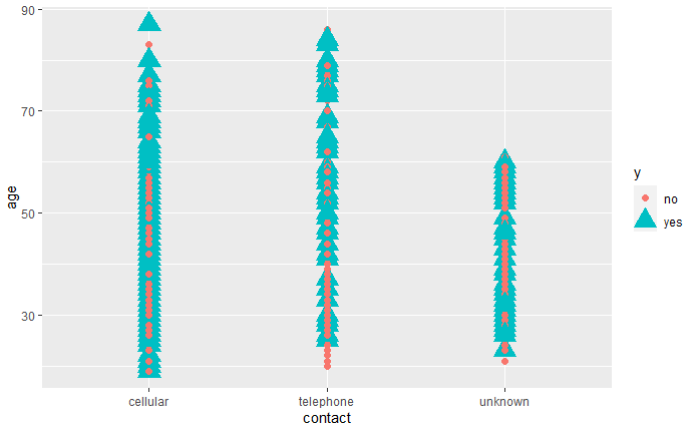


Figure 12: Contact variable value by age

What we observed is that:

- Clients with a age above 60 have the highest subscription rate: since the percentage of *yes* is very close to the one of *no*, meaning that a lot of the people contacted by the bank signed for the term deposit. It is not surprising to see such a pattern because the main investment objective of older people is saving for retirement while the middle-aged group tend to be more aggressive with a main objective of generating high investment income. Term deposits, as the least risky investment tool, are more preferable to the eldest.

- The youngest may not have enough money or professional knowledge to engage in sophisticated investments, such as stocks and mutual funds. Term deposits provide liquidity and generate interest incomes that are higher than the regular saving account, so term deposits are ideal investments for students.

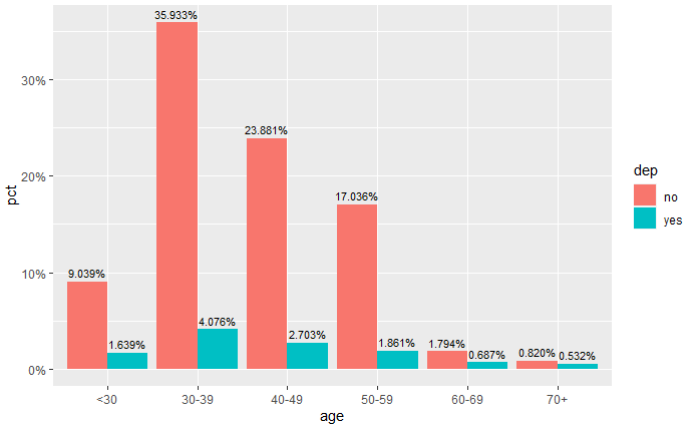


Figure 13: Subscription rate per age

This characteristics have been visualized more properly by taking into account only customers that signed a term deposit.

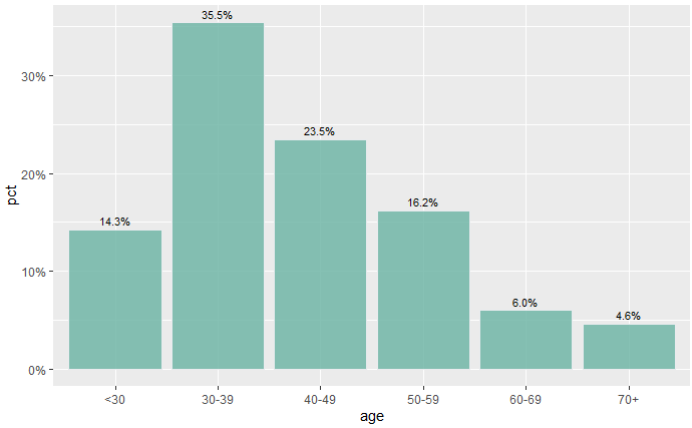


Figure 14: Subscription rate per age groups

The plot on Figure 14 clearly shows that the age range of people most contacted by the bank is the one between 30 and 60, this explains the high percentage of subscriptions. But this may be misleading, as we saw above that older people are much more likely to sign. The second subscription rate we focused on was the one taking into account the job performed by the clients.

What we could deduce from such plot on Figure 17 is that the percentage of *yes* is very high for *retired* and *student* categories, as we already found out above when studying the age ranges. Lastly, in order to better understand the effect of the balance on the probability to sign for a term contract, we visualized it by means of an histogram on Figure 16.

Clients with a balance in the range of 2000 to 5000 are the ones with a highest

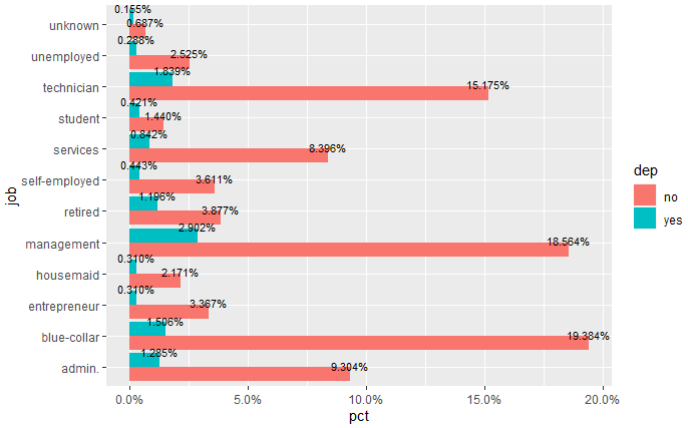


Figure 15: Subscription rate per job

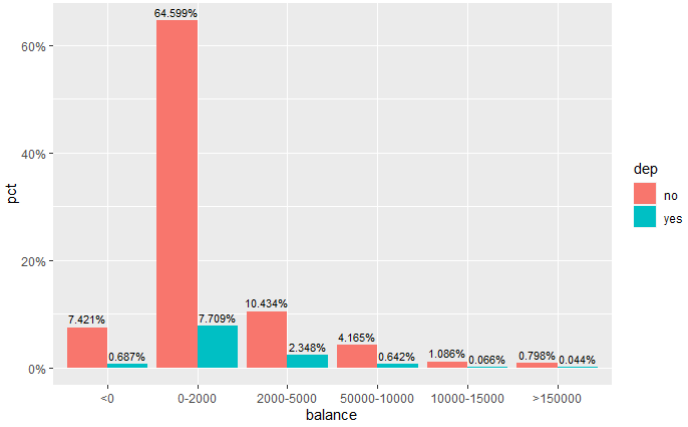


Figure 16: Subscription rate per balance

subscription rate. To confirm this hypothesis, we combined together the information from the ages with that of the balances.

We can see that the highest number of *yes* is in the age range *30-39* for clients with a balance between *0-5000*, since they represent the most called people by the bank.

3.4 Dataset preparation

3.4.1 Features selection

What we observed is that variables *contact* and *day* are not very useful for the classification task. For what regards the *day* variable, we have seen how it's density is pretty uniform along the days of the month, therefore, we have not considered this variable when performing the classification algorithm. The same reasoning is valid for the variable *contact*, which indicates the type of mean used to contact the customers. For what we seen in the above plots, is clear how such variable is not an interesting feature for

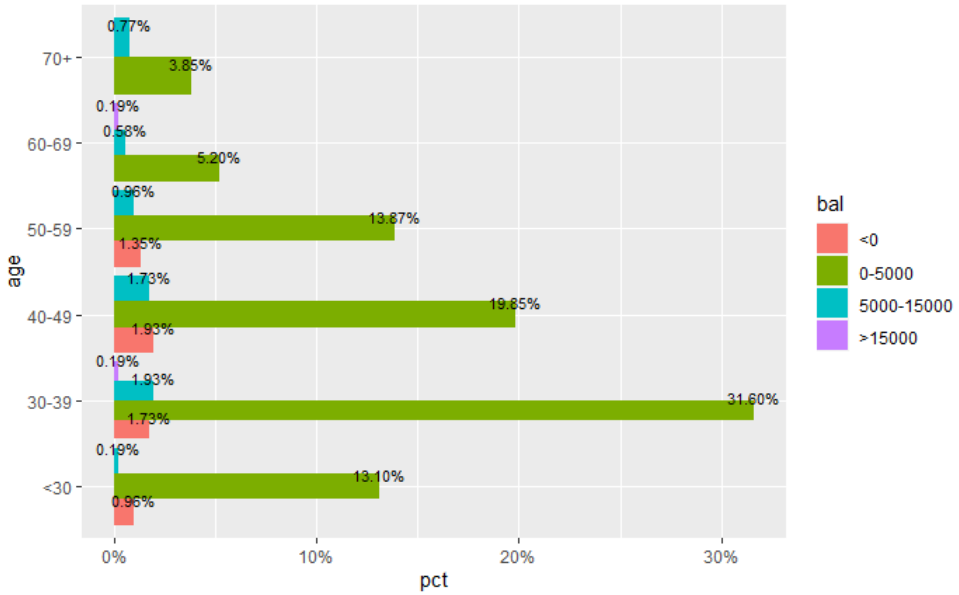


Figure 17: Subscription rate per job

our classification task.

3.4.2 Features engineering

As a second step, we tried to transform variables we thought would have affected negatively the performances of the algorithms with the values they were assuming in the dataset. Specifically, since the variable *pdays* is very spread in the past and has a lot of '-1' values (which makes the classification task harder as compromising the training), we decided to group values into 7 bins. Another observation we made was that, the majority of the classification algorithms work with continuous variables, therefore we needed apply one-hot encoding to the categorical variables.

In order to do that, we converted all the categorical variables into *dummy* variables, i.e creating one variable per category and having a 1 for an observation if the latter was being in that category, before one-hot encoding. The last operation we performed was to convert the string value of the predictor variable to transform it as a factorial variable, being either 1 for*yes* and 0 for *no*.

3.4.3 Multivariate Outliers

Once the dataset has been appropriately explored and converted into its final format, the next step to perform is to catch and remove some possible outliers applying specific detection algorithms.

One way to check for multivariate outliers (non-parametric approach) is the use of the *LOF* (Local outlier factor) algorithm, which identifies density-based local outliers. Specifically, the algorithm computes a local density for observations with a given k-nearest neighbors[1]. This local density is compared to the density of the respective nearest neighbors, resulting in the local outlier factor. Therefore, the function returns a vector of *LOF* scores for each observation: the greater the *LOF*, the greater the outlierness of the data point. The first thing we did, was to visualize the distribution of the outlier scores and the sorted *LOF* scores as can be seen in Figures 18 and 19.

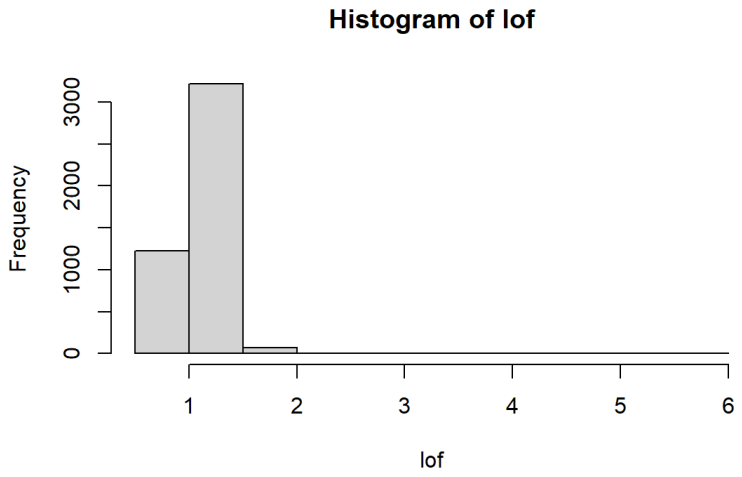


Figure 18: LOF Histogram

From above we understood that the outliers start around a *LOF* value of 2.0. The LOF algorithm detected 5 outliers in total seen in Figure 20, that we later removed. This way, algorithm such as k-means who are sensitive to outliers, won't be affected by extreme observations.

3.4.4 Splitting

About the training and testing split, we used:

- Train: 80 % of the original data will be used for training
- Test: 20% will be used to evaluate the final performances of the classifier

An important remark is that the considered dataset is highly unbalanced: most of the customers belong to the *no* values of the target variable. For that reason, and in order to have a fair evaluation of our algorithms we performed a stratified splitting, to keep the same *yes*/*no* partitions, in both training and testing sets.

[1]We choosed k = 5

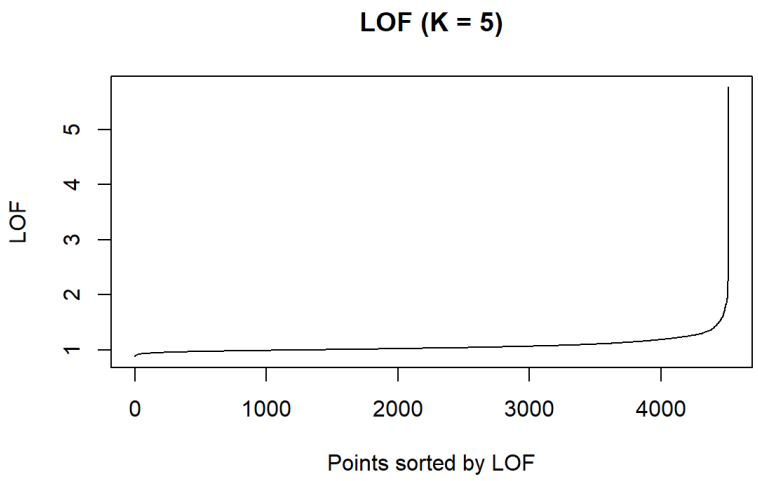


Figure 19: LOF Plot

```
    4511     2967     3413     3542      856
5.766932 2.622858 2.299728 2.248045 2.058318
```

Figure 20: Detected outliers by LOF

In this way we had the same distribution of observations inside the two classes in both the training and test set, as seen in the Table 2.

3.5 PCA

As a preliminary step for classification, we applied dimensionality reduction. Since the absence of specific groups of variables, we performed first a PCA, as a factor analysis method, using continuous features of the dataset. Our purpose was to find some useful relationships between variables and individuals, with respect to the classes, in order to exploit them when performing classification.

Given the high number of variables present in the dataset, the first two Principal Components (PC from now on) won't be enough to completely summarize all the information. As a matter of fact, we can see that the variance is uniformly spreaded along more dimensions as shown in Figure 21

	"no"	"yes"
Training	3185	412
Test	797	104

Table 2: Train/Test repartition

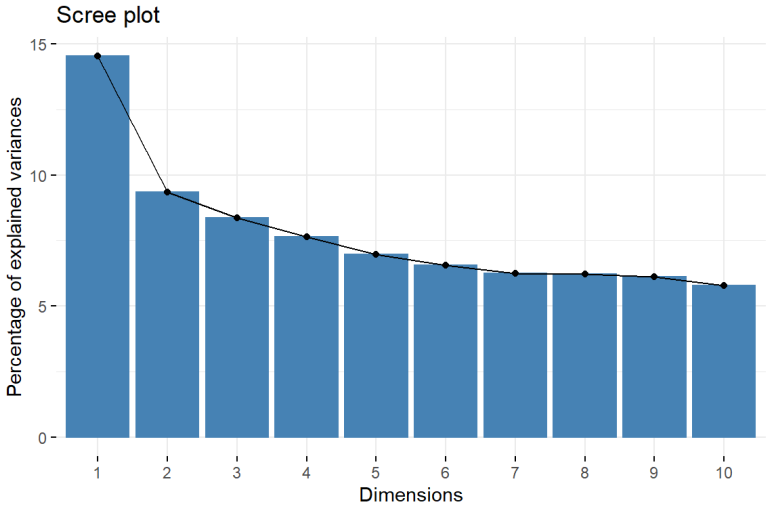


Figure 21: Screeplot

We then plotted the individuals plot as can be seen in Figure 22

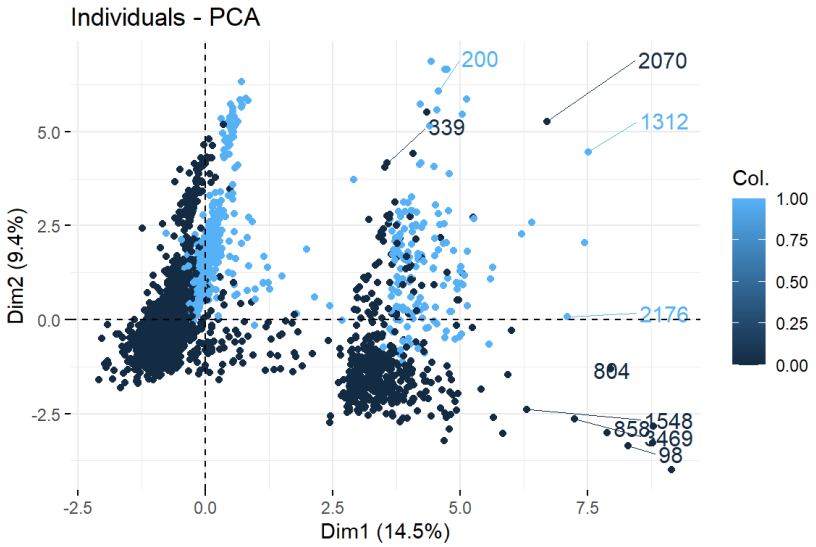


Figure 22: Individuals plot

We can see that the 1st factorial plane separates pretty well the individuals according to the 2 classes. The most interesting plots are the ones regarding the variables. As can be seen in Figure 23 and 24.

The above PCA plot shows an high correlation for the variables *previous* and *poutcome_failure*. The same can be said for *job_retired* and *age*: indeed the higher the age is, the more likely the client will be retired. As explained before about variance repartition

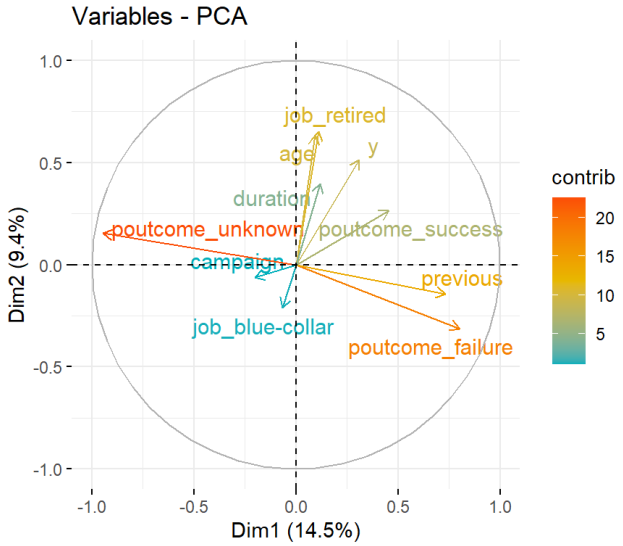


Figure 23: Correlation circle for 1st and 2nd dimensions

on all PC's, it is relevant to visualize the correlation circle for the 3rd and 4th dimensions as well.

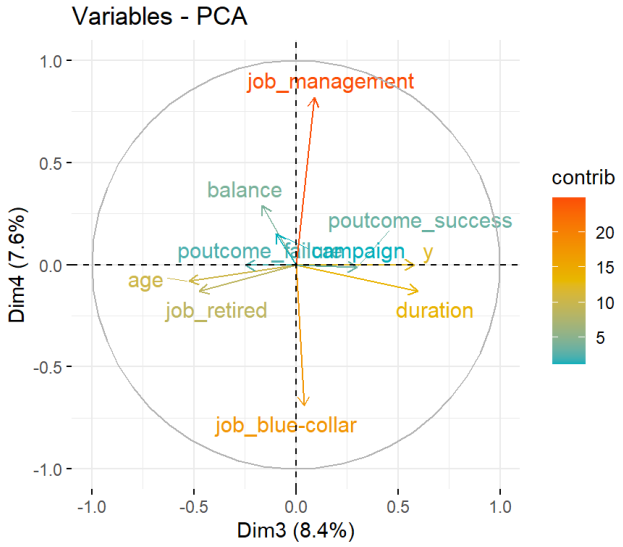


Figure 24: Correlation circle for 3rd and 4th dimensions

Again, we see the same correlation among *age* and *job_retired*. There is also a positive correlation between the target variable *y* and the variable *duration*. Combining both the correlation circle with the individuals plot, we visualized the biplot: Figure 25

However, PCA reaches its limits as trying to maximize the variance inside the dataset,

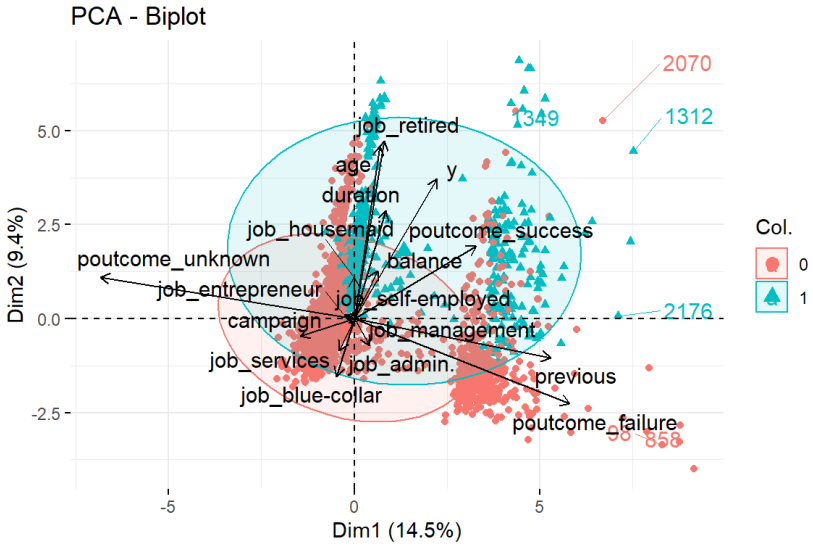


Figure 25: Biplot

thus making it not useful for a classification task. Furthermore, it makes much more sense to apply LDA, for better data visualization. More specifically, we did it to understand if the two classes could be linearly separated, as seen in Figure 26.

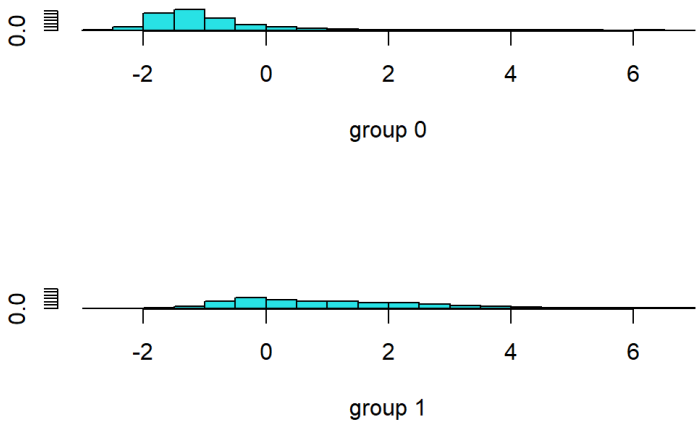


Figure 26: LDA

The new dimension created by LDA is not really able to separate the points into 2 clusters as we can see a lot of overlapping around the value of 0 of the axes.

4 Performance metrics

Because this is a real life problem, we have to give a significant attention to the performance metrics we are going to use.

Among all the performances metrics that we could use: accuracy, precision, F_β-score or ROC to name a few, some will be more or less relevant depending on our goals for the classification and the algorithm used for classification.

In this case, we want to **find which characteristics makes a client signing for a loan** which will allow the bank to improve their processes in order to increase the number of clients whether, for instance, it is by contacting clients at certain periods of the year or to spend more focus on a certain age range of people.

Even though no indications we given by the bank objectives, we will assume that we will focus on having an important number of positive classes as, to contextualize, it is better to trying to call clients that are not going to take a loan than to actually miss clients that could take one. Considering our highly unbalanced dataset, we will evaluate all of our iterations over two performances metrics: **F1-score** and **balanced accuracy**.

The F1-score is a very popular performance metric, being a specific case of the F_β-score based on the following formula:

$$F\beta = (1+\beta^2)\frac{precision * recall}{\beta^2 precision + recall}$$

In the case of the F1-score, it is in fact an harmonic average of the recall (rate of correct positive classification) and the precision (rate of positives correctly predicted). Because it takes into account both later metrics, it allows to draw relevant conclusions without having to be affected by the unbalance of our dataset.

The balanced accuracy score is a simple accuracy score balanced with class partition:

$$BalAcc = \frac{NC_{yes}(1-\alpha_{yes}) + NC_{no}\alpha_{no}}{N}$$

with NC_{yes} being the number of correct predictions for "yes" classes and α_{yes} the proportion of "yes" in the dataset, respectively for N_{no} for α_{no} and N being the total number of predictions.

This is an important metrics because our algorithm will necessarily predict better *no*'s than yes as the dataset provide more negative than positive responses. Therefore, results on the *yes* classifications have to be weighted higher to ensure that we don't induce any bias in our conclusion due to the imbalance.

5 Classification algorithms

In this section we'll go through the algorithms and models that we used for our classification problem. We'll explain the main choices we made and how we achieved improvements in the classification performance metrics we considered.

5.1 K-nearest neighbors (KNN)

The first model we developed is the KNN model, which is a non-parametric, supervised learning classifier, which uses proximity to make classifications about the grouping of an individual data point. It is based on the assumption that similar points can be found near one another. This model has no training at all, since it has no parameters, while he has a quite expensive prediction phase. In particular, a class label is assigned on the basis of a majority vote, i.e. the label that is most frequently represented around a given data point, will be the one used. The nearest neighbours of a given sample are typically chosen by using a distance measure, such as the Euclidean distance. The only hyperparameter of this model is the number of neighbours to consider when making a prediction, and it is indicated with K.

The implementation of the method in R is pretty straightforward, thanks to the *Caret* package [2]. Once obtained the model, we predicted the test dataset and we printed the confusion matrix together with the main evaluation metrics usually used for a binary classification problem.

We run KNN on the training dataset we built and we used k-fold cross-validation[3] to select the best value of the K parameter, with a number of folds set to 10. For what regard the performance metrics to optimize in the cross-validation error, we used the *F1-score*, for reasons stated above in the Performance section 4.

The *Caret package* allows to specify the validation procedure by defining the method *trainControl*, which is then specified as a parameter of the *train* function which performs the training, i.e. the selection of the best value of the k parameter. An important remark is that the KNN method is based on Euclidean distances, therefore we need to standardize the data in order to not give more importance to variables with higher magnitude than others. Again, this preprocessing of the training data is simply done by specifying the *preProcess* parameter in the *train* function of *Caret*.

We then defined a grid of possible values for the K parameters over which to perform hyperparameter tuning.

At the beginning, we ran the model by simply using accuracy as a validation metric[4].

[2] *Caret* documentation

[3] *K-fold cross-validation*

[4] Which is the standard evaluation metrics for a classification task in the *Caret* library

In the Figure 27 it is shown the accuracy value in function of the parameter K. The plot shows a behaviour in which the accuracy tends to improve with the value of the K parameter. This is due to the fact that, according to majority voting, an high number of neighbours pushes the classification of each sample towards the negative values, giving very high accuracy values, but very low recall.

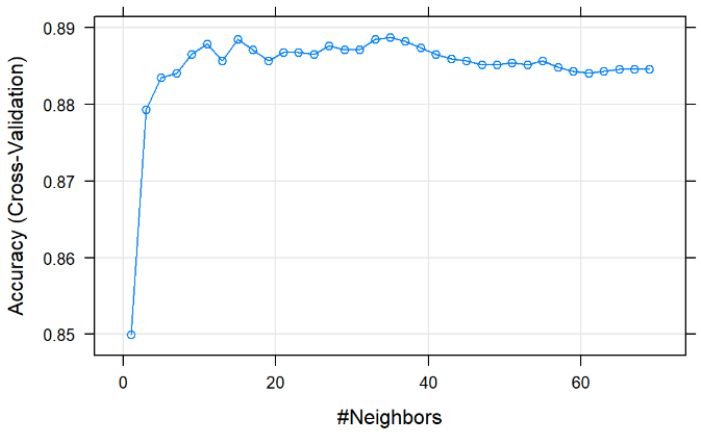


Figure 27: Accuracy with different values of the K parameter

In Figure 28, we reported the confusion matrix and the main error metrics computed on the test dataset for this KNN model.

The confusion matrix shows that the model is very good at predicting the negative class, since it is the most represented in the dataset; while it predicts only 1 correct sample in the positive class. Therefore, the accuracy is very high (88.58%) and also the precision, while the recall and the F1-score are very low.

Indeed, KNN is performed by default using simple accuracy[5] as an evaluation metric. In our case, it may be misleading as the dataset is being highly unbalanced, thus we used the F1-score to tune the value of K. *Caret* contains a summary function called *prSummary* that provides the F1-score.

We therefore built the model by using the F1-score to compute the optimal value of the K parameter. The plot of the F1-score according to the number of neighbours is shown in the Figure 29, while the final model performance on the test dataset is reported in the Figure 30.

The F1-score plot shows that the F1-score decreases by increasing the number of neighbours: this is again due to the imbalance of the dataset, which pushes the model to predict negative classes as the number of neighbours increases, since we'll for sure have a majority of negative 'votes'.

The confusion matrix and the main statistics show that this model produces a quite

[5]Opposed to the balanced accuracy

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes   1   0
       no  103 798

               Accuracy : 0.8858
                 95% CI : (0.8632, 0.9058)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 0.4845

                  Kappa : 0.0169

 Mcnemar's Test P-Value : <2e-16

              Precision : 1.000000
                 Recall : 0.009615
                     F1 : 0.019048
             Prevalence : 0.115299
         Detection Rate : 0.001109
   Detection Prevalence : 0.001109
      Balanced Accuracy : 0.504808
```

Figure 28: Baseline KNN

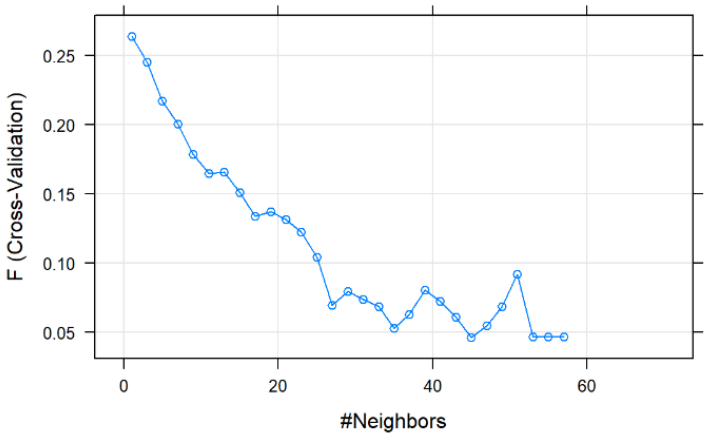


Figure 29: F1-score with different values of the K parameter

big improvement with respect to the previous model both in terms of F1-score and of balanced accuracy, which achieves a pretty good value around 64%.

However, we are not satisfied with the results of this model, with the F1-score being still pretty low, as the model is well predicting the negative class but not the positive class, which is our actual goal[6].

[6]Please refer to the Performance section 4

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  35  45
       no   69 753

               Accuracy : 0.8736
                 95% CI : (0.8501, 0.8946)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 0.86267

                  Kappa : 0.3114

 Mcnemar's Test P-Value : 0.03123

              Precision : 0.43750
                 Recall : 0.33654
                     F1 : 0.38043
             Prevalence : 0.11530
         Detection Rate : 0.03880
   Detection Prevalence : 0.08869
      Balanced Accuracy : 0.64007
```

Figure 30: KNN optimized on the F1-score

5.2 Support Vector Machine

After being unsatisfied with KNN's results, we focused our attention on another classification algorithm, i.e. *Support Vector Machine* [7]. Again, this method is already implemented in the *Caret* package which simplifies a lot the training. It is a supervised classification method, whose main characteristic is to project the data points into an high dimensional feature space, where the points can be separated by a linear decision boundary: this is achieved by the so-called *kernel trick*. This property consists in choosing a function for mapping the data into a high dimensional space with a very low computational complexity and then a linear decision boundary can be easily found in this new space. Several possible choices of the kernel function are possible: the *linear* kernel, the *polynomial* kernel and the *radial* kernel.

The *Caret* package allow to run all of the above mentioned kernels by simply specifying the *method* parameter in the *train* function.

We would like to remark that we used the same validation procedure for running the model that we used in the KNN method, i.e. a 10-fold cross-validation on the training data, based on the optimization of the *F1-score* metric. Again, we centered and scaled the data, since the algorithm is based on distances, and we don't want it to be influenced by the magnitude of the variables.

[7] Support Vector-Machine

As a first step, we run the SVM by using a simple linear kernel, which is a linear classifier on the feature space and without any kind of hyperparameter tuning. We used this first model as a baseline for the subsequent ones, in order to improve its performance. Its final confusion matrix and main performance metrics calculated on the test dataset are shown in the Figure 31.

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  19   5
       no   85 793

               Accuracy : 0.9002
                 95% CI : (0.8788, 0.919)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 0.07755

                  Kappa : 0.2651

 Mcnemar's Test P-Value : < 2e-16

              Precision : 0.79167
                 Recall : 0.18269
                     F1 : 0.29688
             Prevalence : 0.11530
         Detection Rate : 0.02106
   Detection Prevalence : 0.02661
      Balanced Accuracy : 0.58821
```

Figure 31: Baseline SVM

As reported in Figure 31, this model has lower performances that our best KNN model: its F1 score is only 29% and the balanced accuracy is around 59%. Indeed the model still predicts a lot of samples as negative ones, even if they are positive, i.e. it has an high false negative rate.

As a subsequent step, we run the same model, by applying tuning on the parameter *C*, which imposes a penalty to the model for making an error: the higher the value of C, the less likely it is for the SVM algorithm to misclassify a point. For this model we plotted also the F1-score variation with different values of the cost and the plot is shown in Figure 32 .

As shown in the plot the best value for C is obtained around 1.4.

The model's performances on the test set are reported in the Figure 33.

As shown in the performance Figure 33, parameter C does not changes the performances.

We then moved to the *radial kernel*, which allows to find a non-linear classification boundary among the samples belonging to the 2 classes. In this case, we used the *tuneLength* parameter of the *train* function which randomly tries several parameter's combinations, without the need of specifying a grid for each parameter.

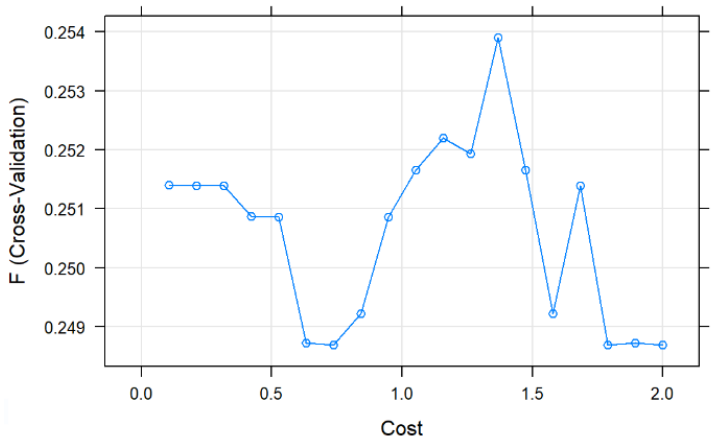


Figure 32: F1-score with different values of the C parameter

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  19   5
       no   85 793

               Accuracy : 0.9002
                 95% CI : (0.8788, 0.919)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 0.07755

                  Kappa : 0.2651

 Mcnemar's Test P-Value : < 2e-16

              Precision : 0.79167
                 Recall : 0.18269
                     F1 : 0.29688
             Prevalence : 0.11530
         Detection Rate : 0.02106
   Detection Prevalence : 0.02661
      Balanced Accuracy : 0.58821
```

Figure 33: SVM with tuning on C

The F1-score plot and the final performances of the model are show on the Figures 34 and 35.

As we can see from the plots, this model allows to slightly improve the F1 score, reaching a value of around 34% and to improve the balanced accuracy by 2%.

The next step was to run SVM with a *polynomial kernel*, i.e. by applying a polynomial transformation to the original variables. As the *radial kernel*, the polynomial one projects the data into a high dimensional feature space where the data points can be easily

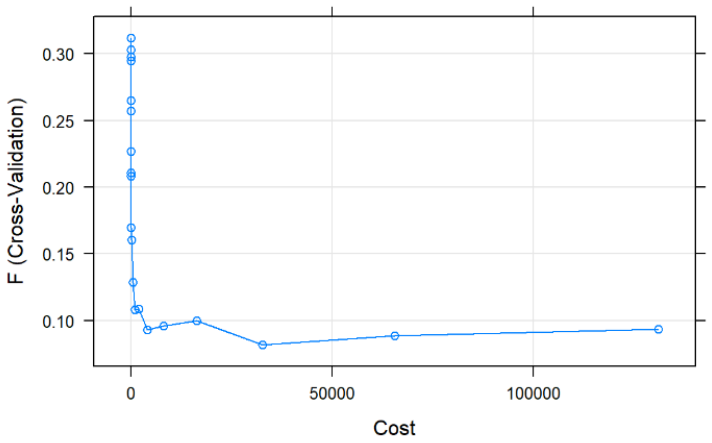


Figure 34: F1-score with different values of the C parameter

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  23   7
       no   81 791

               Accuracy : 0.9024
                 95% CI : (0.8812, 0.921)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 0.05056

                  Kappa : 0.3075

 Mcnemar's Test P-Value : 7.149e-15

              Precision : 0.76667
                 Recall : 0.22115
                     F1 : 0.34328
             Prevalence : 0.11530
         Detection Rate : 0.02550
   Detection Prevalence : 0.03326
      Balanced Accuracy : 0.60619
```

Figure 35: SVM with radial kernel

separated into 2 classes. In this case, we used the *tuneLength* parameter of the *train* function, which allows to randomly try several parameter configurations.

The F1-score plot and the final performances of the model are show on the Figures 36 and 37.

As we can see, this polynomial SVM model achieves approximately the same performances of the kernel-based one, with a very small improvement on the F1-score and on the balanced accuracy.

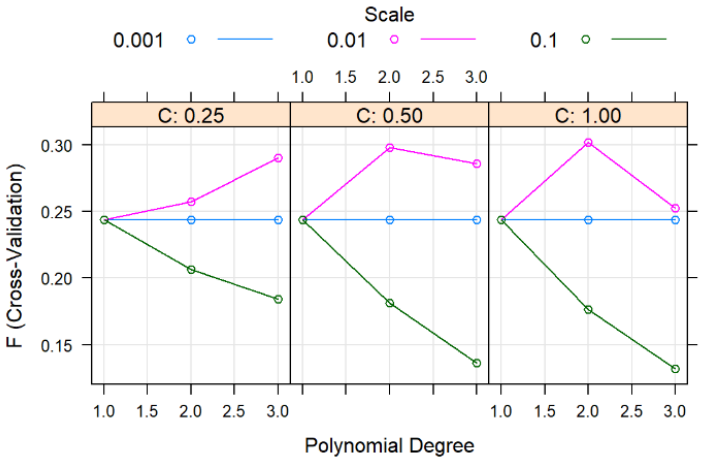


Figure 36: F1-score with different values of the C parameter and of the scaling factor parameter of the polynomial function

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  25   9
       no   79 789

               Accuracy : 0.9024
                 95% CI : (0.8812, 0.921)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 0.05056

                  Kappa : 0.3239

 Mcnemar's Test P-Value : 1.903e-13

              Precision : 0.73529
                 Recall : 0.24038
                     F1 : 0.36232
             Prevalence : 0.11530
         Detection Rate : 0.02772
   Detection Prevalence : 0.03769
      Balanced Accuracy : 0.61455
```

Figure 37: SVM with polynomial kernel

However, the SVM models built until now obtained very bad performances on the test data, especially on the prediction of the 'yes' cases. Indeed, all the models tend to assign 'no' classes to almost all the test data points which is due to the fact that the dataset is very unbalanced. The classification problem we are facing is indeed very challenging due to the fact that one class heavily out-weights the other. For this reason, we tried to adopt some strategies to reduce the effect of this class imbalance on the

training of the algorithm.

5.2.1 Down-sampling

The first strategy we applied to improve the performance of the SVM on the unbalanced dataset was the down-sampling technique which consists in removing instances in the majority class in order to obtain a balance among the number of samples in the 2 classes. The technique can be easily implemented by using the *sampling* argument in the *trainControl* function of *Caret*.

We therefore run again the SVM model with radial kernel by applying this technique and we obtained a great improvement on the final performance of the model on the test dataset. We used the radial kernel since the training was faster with respect to the polynomial one. Again, we used a random selection of the parameters combinations through the *tuneLength* parameter of the *train* method.

The F1-score plot and the final model's performance on the test dataset are reported in the Figures 38 and 39.

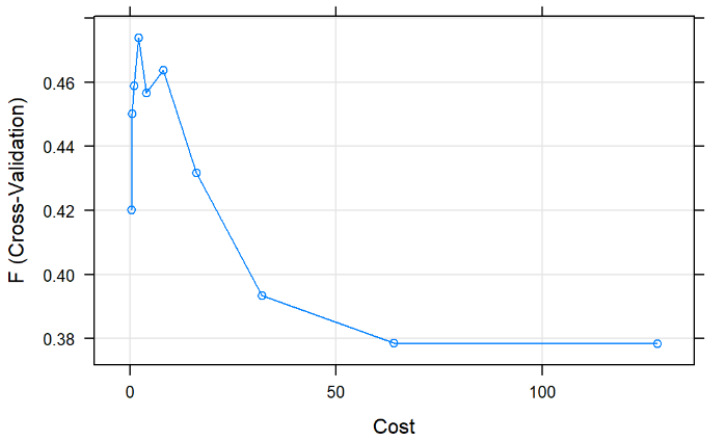


Figure 38: F1-score with different values of the C parameter

As it is shown in the statistics figure, this model allowed us to achieve a very nice result result and a great improvement with respect to all the previously analyzed models: we have now an F1 score which is 50% and a balanced accuracy of almost 82%, which outperforms all the previous methods.

5.2.2 Smote

A technique similar to down-sampling is the so-called Synthetic minority sampling technique (SMOTE), which down-samples the majority class and synthesizes new minority instances by interpolating between existing ones. As before, the technique can be easily implemented by using the sampling argument in the *trainControl* function of *Caret*.

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  86 150
       no   18 648

               Accuracy : 0.8137
                 95% CI : (0.7868, 0.8386)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4117

 Mcnemar's Test P-Value : <2e-16

              Precision : 0.36441
                 Recall : 0.82692
                     F1 : 0.50588
             Prevalence : 0.11530
         Detection Rate : 0.09534
   Detection Prevalence : 0.26164
      Balanced Accuracy : 0.81948
```

Figure 39: SVM with down-sampling

The obtained model has very similar performances as the previous one. The results obtained on the test dataset for this model are reported in the Figure 40.

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  59  58
       no   45 740

               Accuracy : 0.8858
                 95% CI : (0.8632, 0.9058)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 0.4845

                  Kappa : 0.4691

 Mcnemar's Test P-Value : 0.2370

              Precision : 0.50427
                 Recall : 0.56731
                     F1 : 0.53394
             Prevalence : 0.11530
         Detection Rate : 0.06541
   Detection Prevalence : 0.12971
      Balanced Accuracy : 0.74731
```

Figure 40: SVM with SMOTE sampling

This model reaches a slightly worse balanced accuracy than the previous one, but achieves a pretty high F1-score of 53%.

As a further step, we could also have tried the up-sampling technique, which consists in randomly replicating instances in the minority class in order to obtain a balanced dataset. However, due to the high size of the training dataset, the training process and the cross-validation may require a lot of time, therefore we didn't apply this technique and we decided to focus on doing some more hyperparameter tuning on the SVM model built with under-sampling.

5.2.3 More hyperparameter tuning

The SVM model based on the *radial kernel* function is mainly characterized by 2 parameters:

- *sigma*: it defines the *radial kernel* function
- C: it represents the cost parameter we mentioned before

We therefore defined a list of possible values for each parameter and we re-performed the training for 10 random combinations of these parameters. The final model performance on the test set, based on the optimal selected parameters, is shown in the Figure 41.

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  83 130
       no   21 668

               Accuracy : 0.8326
                 95% CI : (0.8066, 0.8564)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4363

 Mcnemar's Test P-Value : <2e-16

              Precision : 0.38967
                 Recall : 0.79808
                     F1 : 0.52366
             Prevalence : 0.11530
         Detection Rate : 0.09202
   Detection Prevalence : 0.23614
      Balanced Accuracy : 0.81758
```

Figure 41: SVM with under-sampling and further tuning

As it is shown, this model achieves satisfying results: it has an balanced accuracy of

82% and a F1-score of 52%. Therefore this is overall our best model so far in terms of both F1-score and balanced accuracy.

5.3 Random Forest

One of the first classification algorithm we tried was Random Forest[8]. We will go over all iterations we made which enabled us to improve our dataset using some feature engineering and fine tuning of the algorithm; all of this in the goal on having better performances. The package *randomForest*[9] was used for all the rest of this section.

A note about the OOB error: Random Forest algorithms are generally assessed using the **Out-of-bag** error. At each iteration, the algorithm uses the bootstrap aggregating technique (also called bagging) to create training samples set from the whole dataset. The Out-Of-Bag error is the mean prediction error over all trees not having a certain training sample in their bag.

5.3.1 Default run

For the first iteration with default parameters with ntree=500 trees and mtry=7[10] variables tried at each split, we had an out-of-bag error of 10.43% but we have a balanced accuracy score of 62.24%, which is relatively low. Interestingly, we have a high F1-score of 94.35% but as our goal is to have a good compromise of both score, we decided to optimize parameters.

5.3.2 Optimal number of variable

As a first step toward the improvement of our Random Forest model, we will try including more variables in trees. As a matter of fact, we introduced a lot of variables due to the one-hot encoding that should be more taken into account when running our RF algorithm.

To do so, we computed the OOB error over different mtry values ranging from all possible values: 1 to 54, in Figure 42.

Having a average minimum around 30, we then computed the Random Forest algorithm using this mtry value and got: 67% on balanced accuracy (7% improvement), 10.07% on OOB error (0.36% improvement) and a F1 score of 94.29% (0.06% decrease). Even though the balanced accuracy was improved, we still made very little improvement in overall, considering all metrics.

[8] We preferred to spend time on optimizing this model instead of going for basic classification trees first.

[9] https://www.rdocumentation.org/packages/randomForest/versions/4.7-1.1/topics/randomForest

[10] Formula used for default value of mtry: $m_{try} = \sqrt{n_{features}}$. We have 48 features, including one-hot encoded variables.

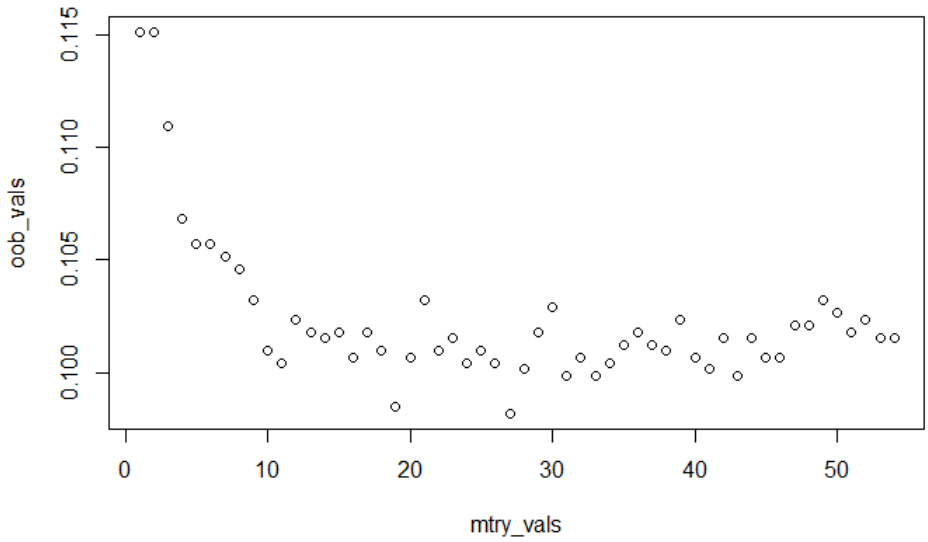


Figure 42: OOB error per Number of variable used

5.3.3 Default run, with categorical

As said before, we introduced a high number of features when one-hot encoding our variables which could make the training harder for the Random Forest as introducing more unnecessary splits. We therefore tried to run the Random Forest with default parameters (ntree=500 and mtry=3[11]

We have an OOB error of 9.94% which represents a 0.5% improvement, a balanced accuracy of 67.35% which is 5% better that the model performed on the dataset including one hot encoding. The F1-score is now being 94.24% which doesn't demonstrate any significant improvement.

Therefore, we will perform the rest of the Random Forest optimisation with the categorical variables instead of the one-hot encoded variables[12].

5.3.4 Optimal number of variables, with categorical

After founding that converting back the variable to factors instead of the one-hot encoding performs better even on default runs, we ran the optimizations again, starting with the number of variable.

[11] Having now 15 features. Please refer to the above footnote on default mtry formula).

[12] A note for the reader: it would have been easier to run the Random Forest computation before one-hot encoding. But we made the choice to first prepare the data with one-hot, thus we preferred the report to follow our scientific reasoning instead of directly following the perfect path.

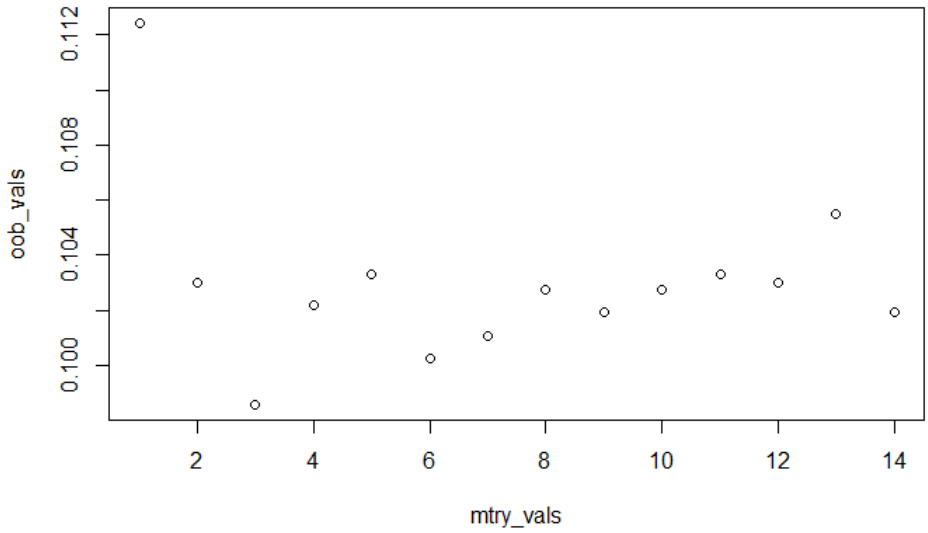


Figure 43: OOB error per Number of variable used. With categorical variables

After running the Random Forest algorithm with the minimal mtry=3, we got: an OOB error of 9.94% (no significant improvement), 69.48% on balanced accuracy (2% improvement compared to default run with categorical) and 94.06% on F1-score (0.2% decrease in performance). To sum up, we managed to improve our balanaced accuracy without loosing that much on the F1-score.

5.3.5 Optimal number of trees, with categorical

After optimizing the number of variables, the last parameter to optimize is the number of trees taken into account in the computation of the forests. In order to do this, we followed a similar procedure as for the mtry optimization: for ntree ranging from 100 to 2000 with a step of 150 trees, we averaged the OOB error over 10 iteration for each ntree value, as can be seen in the resulting Figure 44.

Even though we can see improvements on the OOB error rate after 1000, there is no visible minimum after that value. When running a Random Forest algorithm with ntree=2000, we were able to reach 9.94% on OOB error, 67.83% on balanced accuracy and 94.29% on F1-score. A confusion matrix for this model has been provided in Table 3

Interestingly there is no visible improvement on the balanced accuracy but rather a slight decrease in performances. Due to the high randomness of Random Forests, this score has to be taken into account as approximate as we can observe variations if we run different the algorithm different times.

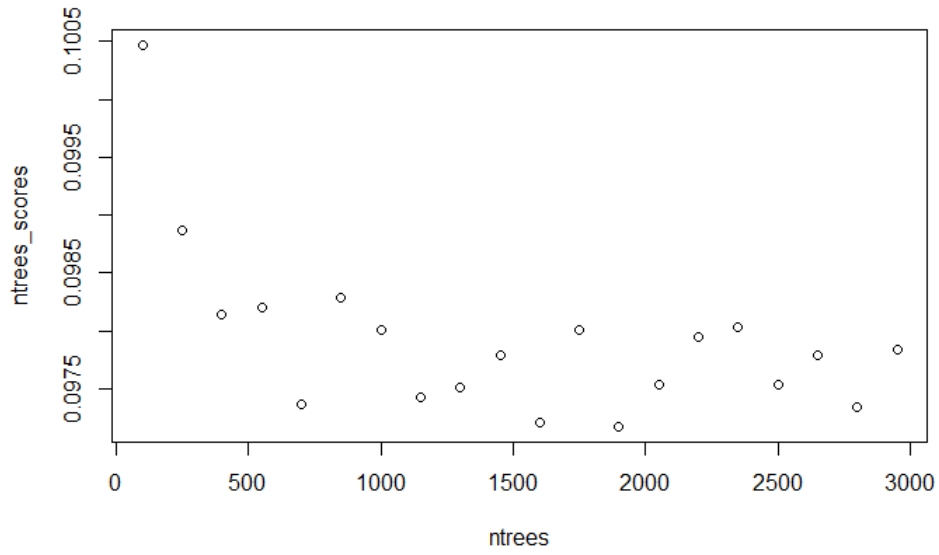


Figure 44: OOB error rate per Number of trees
OOB error rate per Number of trees. 10 Random Forest iteration per ntree value.

Pred/True	No	Yes
No	770	66
Yes	29	38

Table 3: Confusion matrix of final model

5.3.6 Importance of variables

Another insight we can extract from the Random Forest computation is the importance of variables. Two metrics can be used to illustrate the importance of a variable: the mean decrease accuracy and the mean decrease Gini score.

The mean decrease accuracy measure how much accuracy the model is losing when excluding each variable before bagging. A higher mean decrease accuracy for a variable makes it more important in the training than others.

The mean decrease Gini score is related to the Gini impurity of each variable. The higher the Gini impurity is, the more the variable contributes to nodes split stability.

In our case, both "duration" and "month" are the most important variables, which makes sense: for the "duration" variable, the longer the call is with the client, the more chances it has to sign for a term deposit. For the "month" variable as well, people might be more likely to sign for a deposit in periods for money accumulation than on periods

where there might be less prone to term deposits (as described in the EDA, in December for example).

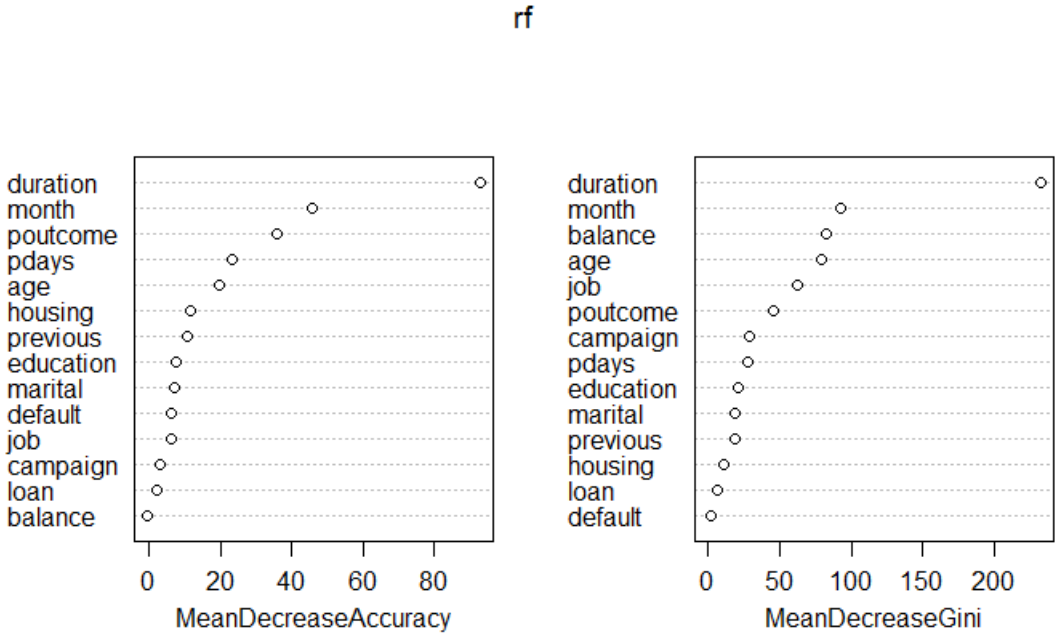


Figure 45: Variable importance

5.3.7 Conclusion

To sum up, we can see on the recap Table 4, the Random Forest algorithm performs very well when considering the F1-score but less than other methods when compared on the balanced accuracy score.

Method	OOB error	Balanced Accuracy	F1-Score
Default run	10.43	62.24	94.35
Opt. mtry	10.07	67	94.29
Default run, w/ cat.	9.94	67.35	94.24
Opt. mtry, w/ cat.	9.94	69.48	94.06
Opt. ntree, w/ cat.	9.94	67.83	94.29

Table 4: Random Forest Performances

Pred/True	No	Yes
No	770	66
Yes	29	38

Table 5: Confusion matrix

5.4 Neural Network

As a last model, we decided to train a neural network classifier. We used the method *nnet* of the *Caret* package which implements a very trivial feed-forward neural network with a single hidden layer. This neural network model is composed by 3 layers:

- *input layer*: it simply represents the input features of each data observation
- *hidden layer*: layer containing units computing a sigmoidal activation function over a linear combination of the input layer values
- *output layer*: single output unit predicting the probability of the positive class

In particular, this simple feed-forward neural network has the following hyperparameters that needs to be tuned:

- Number of neurons in the hidden layer
- *Weight decay*: parameter that allows to apply regularization in order to prevent overfitting by constraining the weights to low values.

As done for SVMs, we trained the NN model on the training dataset by applying 10-fold cross-validation and by optimizing the *F1-score* metric.

We therefore defined a list of values for each one of the parameters, i.e. for the weight decay and for the number of hidden neurons:

- For the weight decay we chose negative powers of 10, from 0.0001 to 0.1
- For the neurons in the hidden layer we chose 4 small integer values from 5 to 20 with a step size of 5

We then applied the *train* function using again the parameter *preProcess* to standardize the data. Also, in this case it is suggested to scale the data in order to not weight more the error for variables with high magnitudes, and to balance the influence of each variable during the training process.

After training the model we printed the optimal parameters found with the 10-fold cross-validation procedure:

- hidden layer size: 5
- weight decay: 1e-04

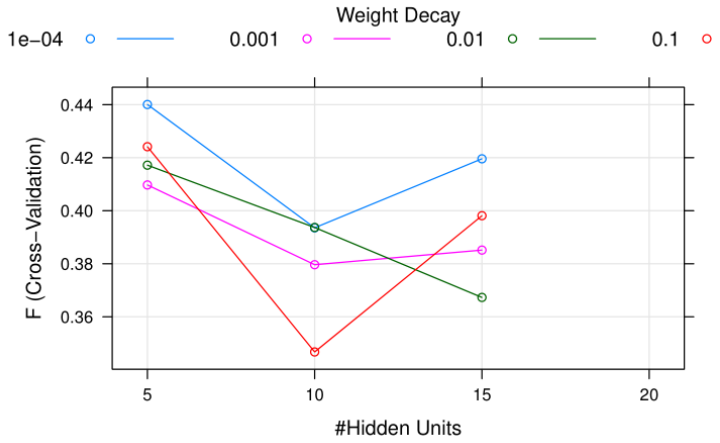


Figure 46: F1-score with different values of the hyperparameters

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  37  28
       no   67 770

               Accuracy : 0.8947
                 95% CI : (0.8728, 0.914)
    No Information Rate : 0.8847
    P-Value [Acc > NIR] : 0.1885

                  Kappa : 0.3832

 Mcnemar's Test P-Value : 9.67e-05

              Precision : 0.56923
                 Recall : 0.35577
                     F1 : 0.43787
             Prevalence : 0.11530
         Detection Rate : 0.04102
   Detection Prevalence : 0.07206
      Balanced Accuracy : 0.66034
```

Figure 47: Neural Network baseline

We also performed the plot of the F1-score depending on the 2 hyperparameters of the model, which is shown in Figure 46.

The final model performance in predicting the test dataset is shown in the Figure 47

The main statistics show that this trivial neural network achieves a pretty high balanced accuracy value of around 66%, and an F1-score of 43% which is still lower compared to the values we obtained with SVM.

5.4.1 Applying weighting

In order to improve the performance of the baseline neural network model we build in the previous step, we decided to adopt the weighting technique, which is an alternative method to be used when dealing with unbalanced datasets.

This technique consists in assigning weights to each training observation: in particular, for samples in the negative class we assign a lower value of the weight, while we assign higher weights to positive samples. The main purpose is to give more importance to the correct classification of the positive samples with respect to that of the negative ones, since the 'yes' observations are much less in number.

The weighting of samples is simply applied by specifying the *weights* parameters in the *train* method of *Caret*.

Before performing the training, we slightly reduced the grid of hyperparameters to use, by taking into account a smaller number of hidden neurons (since the previous model was optimal with a value of 5).

The optimal chosen parameters for this model are:

- Hidden layer size: 12
- Weight decay: 0.01

The F1-score plot is shown in the Figure 48, while the model's performance in the test dataset is shown in the Figure 49

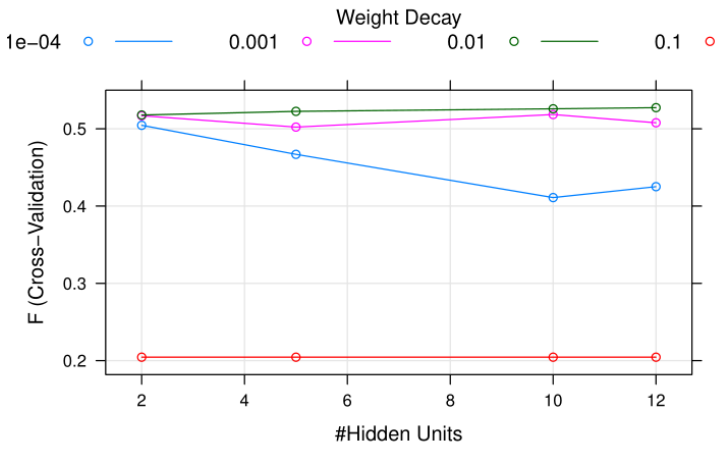


Figure 48: F1-score with different values of the hyperparameters

As shown in the figure, this final model achieves a balanced accuracy value of 80% and a F1-score of 52%, therefore it is comparable to our best SVM model.

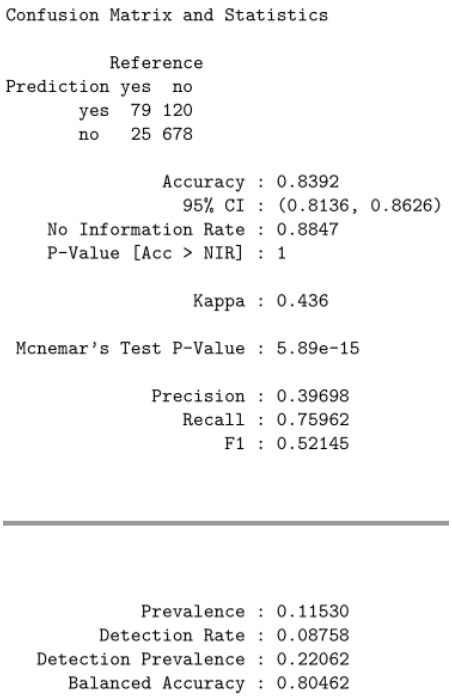


Figure 49: Neural Network with sample weighting

As a final remark, we would like to notice that for each trained model we printed also the importance of each variable for performing the classification. The results can be shown in the attached *html* files, and we report in Figure 50 the values obtained for the case of this final neural network model

As a further step, we could have trained some neural networks with more hidden layers, with different regularization procedures (such as dropout), different optimization algorithms and learning rates.

```
nnet variable importance

  only 20 most important variables shown (out of 53)

                        Overall
duration                100.000
poutcome_success         27.846
month_oct                24.890
month_mar                18.248
month_may                15.619
campaign                 15.063
job_retired              14.175
‘pdays_Under 1.5yrs‘     13.051
‘pdays_Never contacted‘  12.527
month_sep                11.500
month_apr                10.957
loan_no                   9.792
loan_yes                  9.791
‘pdays_Under 6months‘     8.697
‘job_blue-collar‘         8.346
housing_yes               8.137
housing_no                8.137
‘pdays_Under 3months‘     7.959
poutcome_failure          7.823
‘pdays_Over 1.5yrs‘       7.496
```

Figure 50: Neural Network variables importance

6 Conclusion

In this project, we have been trying the best classification method for our dataset, based on our knowledge and on the issue that the bank would have wanted us to tackle.

Every model has been fine tuned in some way in order to ensure that we extracted the best possible results for each models, based on the available parameters.

In the Table 6 we report the best results obtained with each one of the models

Method	Balanced Accuracy	F1-Score
K-NN	64.01%	38.04%
SVM	81.76%	52.37%
Random Forest	67.35%	94.24%
Neural Network	80.46%	52.15%

Table 6: Overall Performances

Random Forest seems to be the best model for what regards the F1-score: as we saw from its confusion matrix in the TableFigure 3, it is able to predict quite well also the positive samples, which our main goal.

This is a reasonable result, being Random Forest and ensemble method based on bagging, which is based on averaging the predictions produced by many simple classification trees. This allows to reduce a lot the variance of the final model, as well as to control overfitting.

However, when considering the balanced accuracy, more modern methods such as Neural Network (even though having a basic structure) are performing well, due to their ability to learn from the data.