Final report of the Kaggle Competition

Dario Dantoni, Bich Ngoc Doris Le & Gabriel Sallin as TEAM NAME

May 2020

Contents

1 Introduction

In this report, we will present the Kaggle dataset `Online Ads Quality Evaluation`.

This dataset represents online users surfing on a website that choose to subscribe or not to a service proposed by a web banner. We are asked to fulfil a classification task. We want to build the best model in order to predict if a user will convert to the service, according to a set of variables describing each user.

We will first start with the description of the dataset. Then, using different kinds of plots and tools, we will try to understand the underlying relationships between variables. The next step will be the presentation of different predictive models, followed by their results. We will compare which one has the best predictive power, looking at the metric accuracy. It is the ratio of the correct predictions over the total number of predictions. This is also measured by the following formula :

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} 1\{y_i = \hat{y}_i\}$$

2 Description of the dataset and exploratory analysis

2.1 Description

For this problem, we have two sets of data. The first is the training data set, which consists of $n = 8526$ samples (x_i, y_i), with $x_i \in \mathbb{R}^p$, where the number of predictors is equal to 16, and $y_i \in \mathbb{R}$, $i = 1, \ldots, 8526$. The test data set consists of $n = 4263$ samples, with the IDs and the predictors $x_i \in \mathbb{R}^p$.

The predictors are consisting of eight categorical variables : job (type of job), marital (marital status), education (education level), device (from which device does the user see the banner), outcome_old (outcome of a previous online ads campaign), and X1, X2, X3 (variables with no name).

The other variables are numerical : age, day (last day of the month when the user saw the banner), month (last month of the year when the user saw the banner), time_spent (how long the user looked at the banner), banner_views (number of times the user saw the banner), banner_views_old (same as banner_view but for an old campaign), days_elapsed_old (number of days the user saw the banner of an old campaign) and X4 (variable with no name).

The output is a 0-1 variable, taking the value of 0 if the user did not convert and 1 otherwise. It is distributed as almost 60% of 0's and 40% of 1's. Thus, there are more people that did not convert.

2.2 Plots

We first need to check the missing variables in our dataset, by checking for both train and test sets.

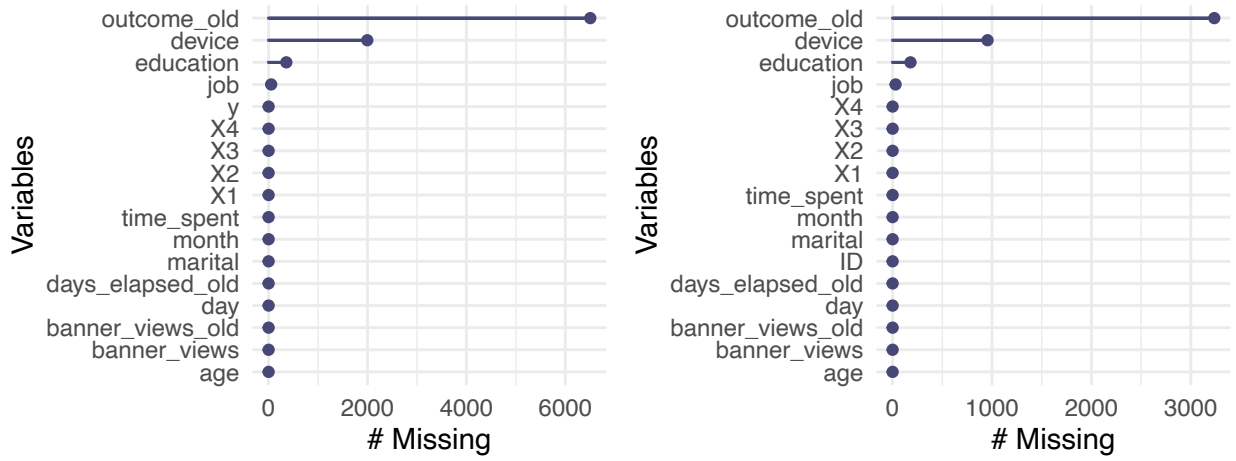

Figure 1: Missing variables of the train and test sets

From these plots, we note that there are only missing variables for categorical variables. It could mean that the individuals did not find the appropriate category when they filled it. It is not an issue, we will just consider the missing variables as an other category, keeping it as "na". For the numerical variable day_elapsed_old, it takes the value -1 if the user never saw the banner of an old campaing. Thus, the same invidivuals that have an "na" for outcome_old also have -1 for days_elapsed_old and 0 for banner_views_old. There is only one exception with the user 2591 of the training set, which has "na" for outcome_old, 528 for day_elapsed_old and 7 for banner_views_old. It could be a mistake.

Then, we will consider the correlations between variables.

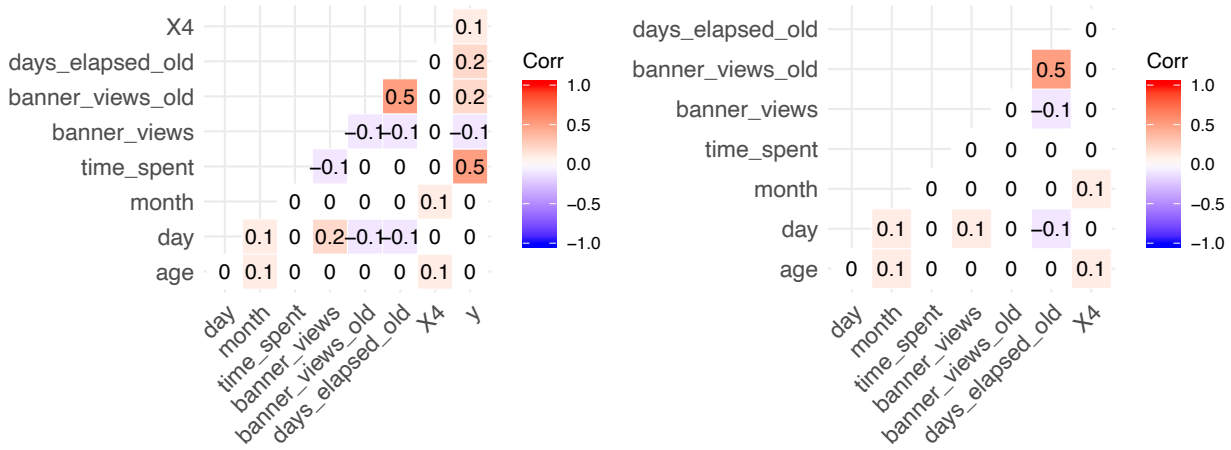

Figure 2: Correlation matrix of the train and test sets

These correlation matrices allow to observe which variables evolve in the same direction (for a value close to +1) or in the opposite direction (for a value close to -1).

Here, we can observe, for the numerical variables, that correlations are not particularly important, as most of them are equal to 0.

However, the number of views of the old banner is positively correlated with the number of days since the user saw this old banner online ads (correlation between banner_views_old and days_elapsed_old is +0.5).

Also, the response variable is positively correlated with the time spent by the user looking at the banner, which seems reasonable (correlation between y and time_spent is +0.5).

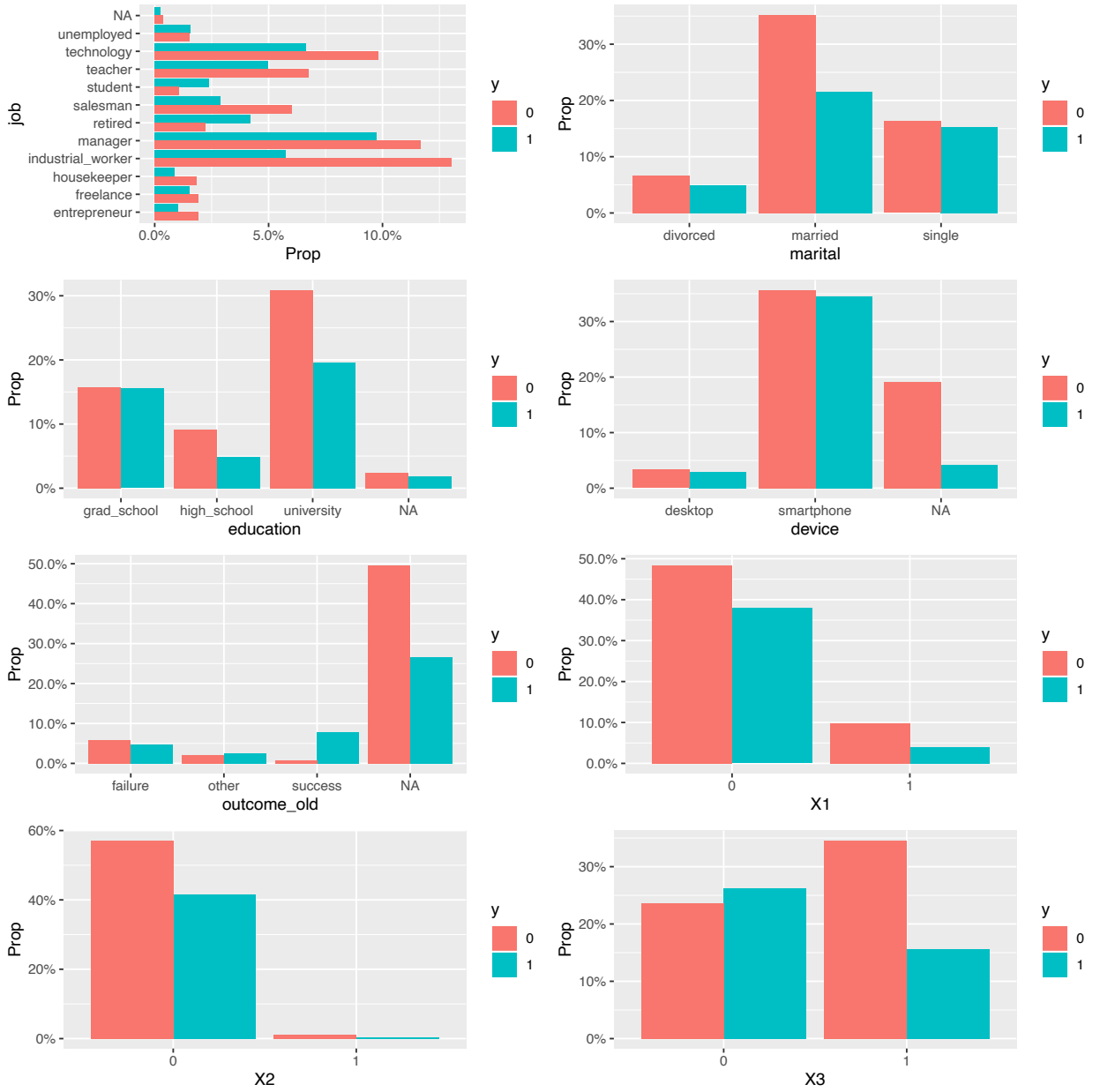

Figure 3: Barplots of the categorical variables with the proportion of target variable for the train set

From these barplots, we can note interesting remarks :

- Students and retired convert more often. Maybe they have more time to spend on the website and they are more sensitive to the service.
- No particular relationship for marital status, except for married that tends to convert less.
- The higher the level of education, the more the proportion to convert decreases.
- Those who did not answer from a smartphone or a desktop usually do not convert. Maybe some devices belonging in the "NA" category do not have the possibility to click on the web banner ?

- More than 6000 users (around 75%) did not participate to a previous ads campaign. For those who did, a majority of conversions for the previous one have also a conversion for this one. This point needs to be explored by the service provider.
- Although we do not know what are X1, X2 and X3, we can still see that the value 1 is very scarce for the X2 variable.

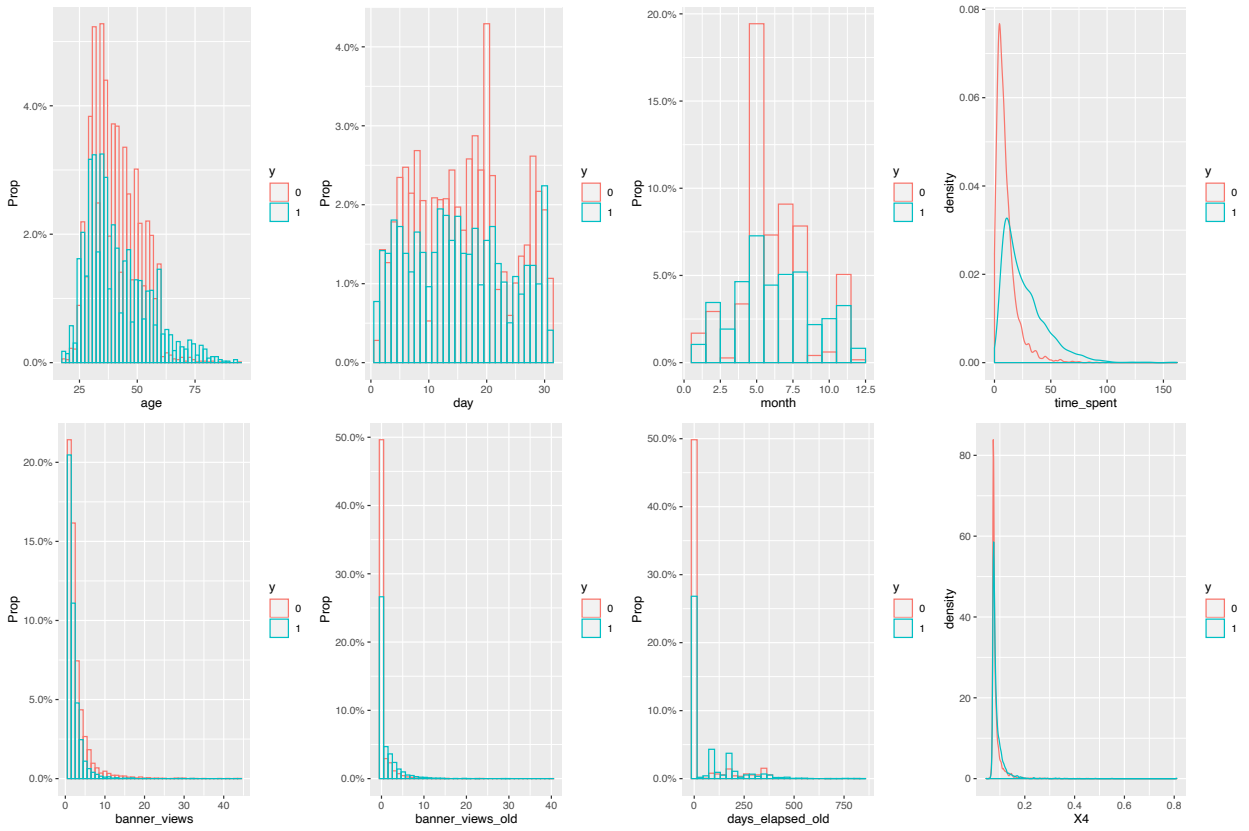

Figure 4: Density plots of the numerical variables with the proportion of target variable for the train set

Findings :

- Young and old people convert more often. This is similar as what we have seen before with the students and retired.
- The 20^{th} of each month is the day where least people convert, 30^{th} is the day with most conversions.
- Most people where recorded in May, thus it is the month with both the most conversions and the least conversions.
- The more time spent, the more conversions.
- The number of time an individual sees the banner does not seem to be correlated with the conversion to the service.
- Nothing particular on the number of times the user saw the banner during an old campaign.
- Nothing particular on the number of days since the user saw the banner of an old campaign.

- Difficult to draw conclusion with the unknown variable X4.

The same analysis for the variables of the test set can be found in Appendix 1 : Exploratory analysis for the test set.

2.3 Data Transformation

We need to use the one-hot encoding for categorical variables to create dummies. With this method, new variables are created for each category of the categorical variable, and they take the value of 0 or 1 if the individual belongs to the category or not. When the variable has "na" values, a variable with the category "na" is also created.

Most of machine learning techniques need to use scale variables. It will be done using the option `PreProcess(method = "center", "scale")` in `caret` function `train`.

We will specifiy for each method what treatment has been applied.

3 Models

We will present at least one model of each of these three categories : Generative, Discriminative and Geometric.

- Generative models estimate the joint distribution (Y, X) and find the posterior class probabilities for $j = 0, 1$.

$$\hat{Pr}(Y = j|X = x) = \frac{\hat{Pr}(Y = j, X = x)}{\hat{Pr}(X = x)}$$

- Discriminative model estimates only the conditional distribution $(Y \mid X)$, that is, directly the posterior class probabilites $\hat{Pr}(Y = j|X = x)$
- Geometric model directly estimates a function that predicts the class 0, 1 of y from x.

We will present six models : Random Forest, XGBoost, SVM, Neural Network, Ensembles and LDA (that can be found in Appendix 2 : LDA).

All accuracies of all models are displayed in the Section 4, Table 1.

3.1 Cross-validation

For all of our models, we use a 10-folds cross-validation which creates new test samples with the same training set. It allows to test the skill of the model on new data and to avoid overfitting. A model will perform often better with data on which it was trained than unkown data. It is also a way to get the best tuning parameters that guarantee the highest acuracy, for the models that need such parameters.

The procedure to create it is as follow :

- Separate dataset in a certain amount of folds (we choose 10 folds).
- Retrieve the first fold which will be used as test set.
- Fit your model on the other folds with the tuning parameter chosen.
- Evaluate your model on the excluded fold with the chosen metric.
- Repeat the procedure but now with the second fold as test set.
- Once all the folds have been used as test set, make the average of all the metrics. This will be your average metric for the given tuning parameter. (You can also calculate the variation to used the "one standard deviation rule").
- Repeat all the procedure but with another tuning parameter.
- Choose the tuning parameter which has the best metric.

3.2 Random Forest

3.2.1 Model Presentation

As an example of discriminative model, we choose to present Random Forest.

Random Forest is an extension of the tree-based method, so we will first describe how this latter works. The tree-based method stratify feature space recursively into simple regions, namely rectangle $R_1, \ldots, R_n$.

To make predictions for a given observation, we use the mean or the most common class of the training observations in the region to which it belongs (here since our problem is a classification one, we will use the most common class).

Figure 5 shows an example of a classification tree.

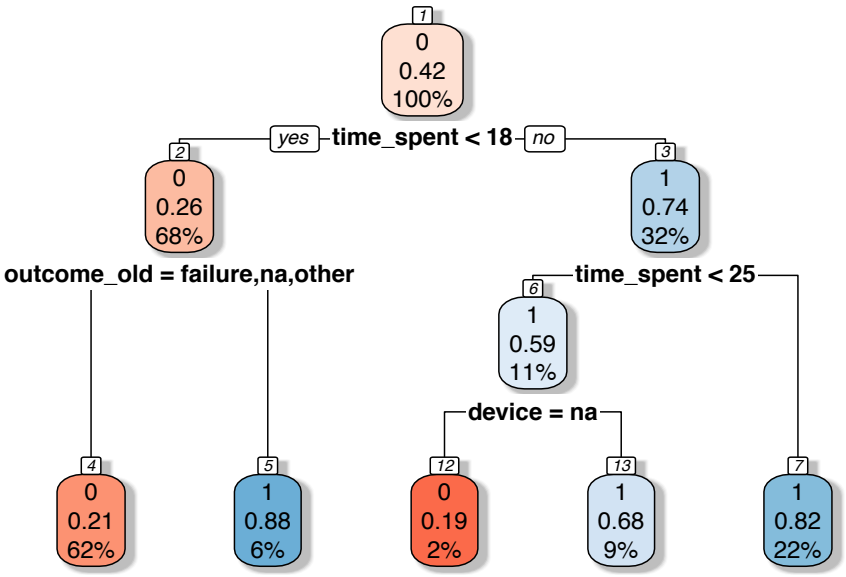

Figure 5: Classification tree with five final nodes

Terminology:

- Each split is called a node.
- A terminal node is called a leaf.
- The interior nodes lead to branches.
- The label $(X_j < t)$ indicates the left-hand branch from that split and the right-hand split corresponds to $(X_j \geq t)$ with X_j being the j^{th} rectangle and t being the cutpoint.
- The number in each leaf is the most common class of the responses for the observations that fall there.

Growing the tree :

Since it's computationally impossible to search for all the possible partitions, we use instead a top-down, greedy algorithm that is also known as recursive binary splitting. It is top-down because it begins at the top of the tree and then successively splits the predictor space (each split is indicated by two new branches further down on the tree). It is greedy because at each step of the tree-building process, the best split is made at that particular step. To determine the best split we take the one which minimizes the Gini index or the deviance/cross-entropy.

$$GiniIndex : G = \sum_{j=1}^{k} \hat{p}_{jg}(1 - \hat{p}_{jg})$$

$$Deviance/cross - entropy : D = -\sum_{j=1}^{k} \hat{p}_{jg} log(\hat{p}_{jg})$$

With $\hat{p}_{jg}$ being the probability that the class g is in the j^{th} rectangle/region.

For a Random Forest we use boostrap data sets (samples with replacement) from which we grow a full tree for each one of them. Also it is important to notice that at each split, a number of variables (tuning parameter) is chosen and between them, the best one is selected. It makes the trees uncorrelated from each other which allows the variance to decrease. On RStudio this tuning parameter is called $mtry$.

Data manipulation:

No manipulation or transformation has been made on the data. Even the "na" have not been cleaned and transformed in NA.

Tuning parameter selection:

For the Random Forest classification, the tuning parameter $mtry$ has to be determined.

As we can see from Figure 6, $mtry = 1$, performs very badly for the train and CV. Indeed, since we are taking only one parameter randomly, the full grown tree has not anymore a small bias. We can also see that the train accuracy is much higher since we trained our model on the training set, it should be better than an unknown dataset. To avoid overfitting we are going to choose the one which maximizes the CV accuracy.

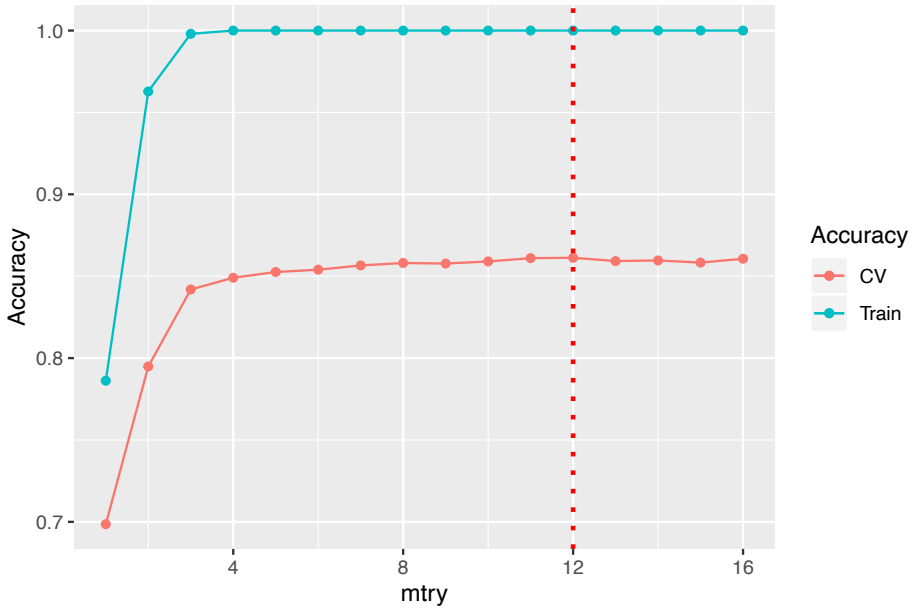

Figure 6: Train and cross-validation accuracy for Random Forest

Evaluation of the model:

The best tuning parameter chosen is $mtry = 12$. It maximizes the CV accuracy at 86.12%. The accuracies can be found in Table 1.

```
rf_default$results$Accuracy[which.max(rf_default$results$Accuracy)]
```

```
## [1] 0.8612488
```

```
rf_default$bestTune
```

```
##    mtry
## 12   12
```

3.2.2 Advantages

- Great predictive performance.
- Almost no tuning needed.
- Easy to fit using existing functions in R and quite fast.

3.2.3 Disadvantage

- By bagging trees we lose the interpretability of a single tree.

3.3 Extreme Gradient Boosting (XGBoost)

3.3.1 Model presentation

Extreme Gradient Boosting (XGBoost) comes from the tree family. In order to explain it, we first define Boosting, Gradient Descent and Gradient Boosting.

Boosting algorithm is an ensemble method that improve prediction by combining models. In the following, we will define it in the case of decision trees. The principle is to grow iteratively trees that will learn from the previous grown trees. The new trees added fix the errors of the previous trees. It is a sequential approach where at each step we improve our performance. The trees are weak, biased with a low variance. By combining the weak learners (poor prediction) until there is no improvement, we should build a stronger model by reducing the bias.

Gradient Descent is an optimization algorithm. The gradient points in the direction of the steepest increase of the function. In order to minimize the function, it moves in the opposite direction of the gradient.

Gradient Boosting uses the same principle as the Boosting algorithm but it uses a Gradient Descent algorithm to minimize the loss when new trees are added.

XGBoost is an evolution of the Gradient Boosting. It was created to have a higher predictive power and to be faster than Gradient Boosting. It is an optimized Gradient Boosting through parallel processing and tree-pruning. Moreover, it includes a variety of regularization which reduces overfitting and improves overall performance.

Figure 7 summarizes what we have just said above.

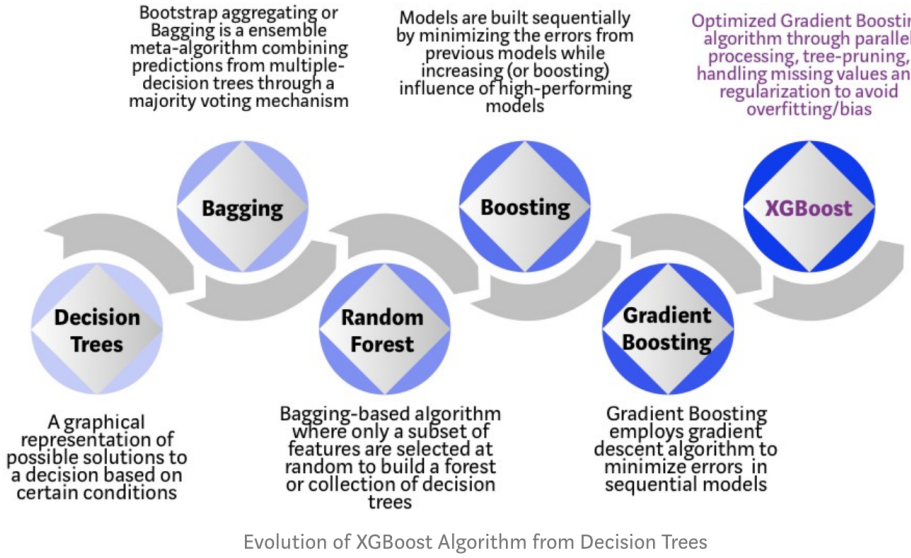

Figure 7: Tree familiy (Source: https://morioh.com/p/32c506939ad5)

Data manipulation:

"na" being present only on categorical variables, they have been considered as an another level and categorical variables have been coded as 0-1 (dummy).

Tuning parameters selection:

For the XGBoost classification, many tuning parameters have to be determined. The complexity of the XGBoost model is on the amount of its parameters. We are going to describe two of them. *Max depth* is the maximum depth of each tree. The higher it is, the closer we are to overfit. *Eta* is a shrinkage parameter which controls the learning rate and so it makes the model more robust. The smaller it is, the slower the model learns. There are many others but here we have used seven parameters where each one of them has to be optimized. For the sake of time, we decided to keep fixed four parameters and try all the combinations of the three left ones at each round. The best combination of tuning parameters that we found:

- nrounds = 400
- max depth = 7
- eta = 0.03
- gamma = 1.5
- colsample bytree = 1
- min child weight = 0
- subsample = 0.75

```
xgbmmod$bestTune
```

```
##    nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 81     400         7 0.03   1.5                1                0      0.75
```

Evaluation of the model:

We can see the evolution of the accuracies in function of *nrounds*. We kept the other tuning parameters constant as they are in the "Tuning parameters selection" part. We can observed with the red dashed line that at *nround* = 400, our model performed the best on the CV accuracy. The accuracies can be found in Table 1.

Figure 8: Train and cross-validation accuracy for XGBoost

3.3.2 Advantages

- As a tree method, there is no distribution assumption.
- Optimized Gradient Boosting which accelerates the process through parallel processing.
- Reduces the overfit with its implementation of regularization.
- It can handle missing values.

3.3.3 Disadvantage

- Lots of tuning parameters which can make the number of possible combinations quite huge.

3.4 Support Vector Machine

3.4.1 Model Presentation

The Support vector machines (SVM) tries to find the hyperplane that separates the classes in the feature space. Thus, it is a geometric model. We used here a SVM with a radial kernel.

$$x_i \in \ R^p : \beta_0 + \beta_1 x_1 + ... + \beta_p x_p = 0$$

Data manipulation:

We created dummies for the categorical variables and we standardized the numerical variables.

Tuning parameters selection:

We fitted the model to the training data and the model was optimized with the tuning parameters C and σ. The cost C is a positive constant and it controls the tolerance of the margin to be violated by the observations. The margin is the distance between the hyperplane and the closest data to it. A high C chosen, will give wide margins and a high missclassification rate. The SV classifier will have a high bias and a small variance. A low C chosen, will give narrow margins and a low missclassification rate. The SV classifier will have a low bias and high variance.
γ controls the curvature of the decision boundary and our second tuning parameter σ is related to it.

$$\gamma = \frac{1}{2\sigma^2}$$

We optimized the tuning parameters in two rounds.
The first round gave us a range of values for the best tuning parameters. We started to fit the model with C =[0.03, 0.1, 0.3, 1, 3, 6] and σ=[0.03, 0.1, 0.3, 1, 3, 6]. Then from the result of the first round, we fitted the model with a fixed σ= 0.03 and C = [0.25, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4].

For a fixed $\sigma = 0.03$, we found the optimized tuning parameter $C = 1.5$.

```
model_svmradial2stand$bestTune
```

```
##   sigma   C
## 6  0.03 1.5
```

Evaluation of the model:

In figure 9, we can observe the accuracies during the optimization of the tuning parameter C. The accuracies can be found in Table 1.

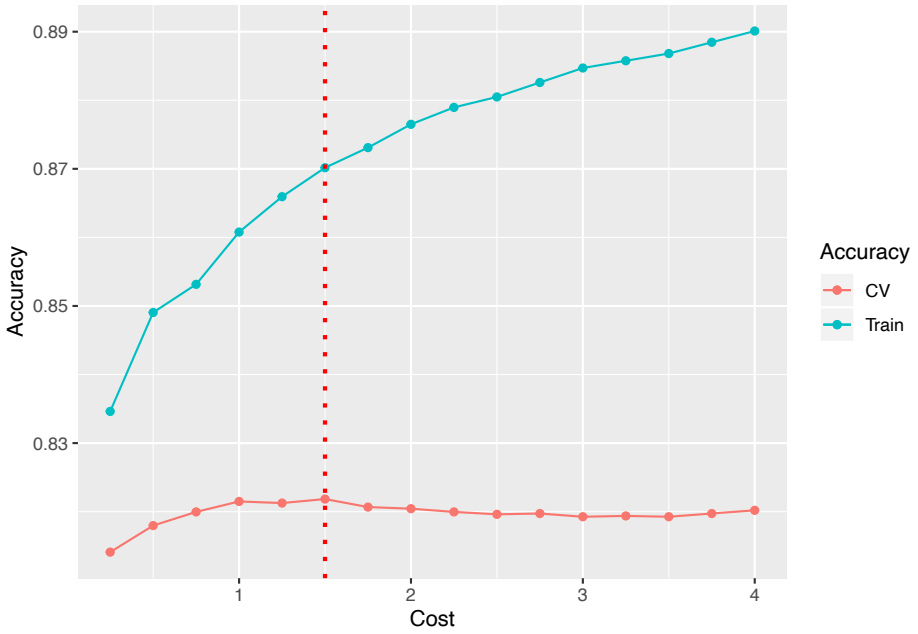

Figure 9: Train and cross-validation for SVM

3.4.2 Advantages

- SVM is a very powerful and flexible tool for classification.
- There is no distribution assumption.

3.4.3 Disadvantages

- SVM does not provide estimates for the probabilities of the class predictions.
- The predictors have to be standardized.

3.5 Neural Network

3.5.1 Model presentation

Neural Network is a discriminative model and non-linear parametric model. It is consisting of neurons and layers. The input layer has as many neurons as our number of variables. The output layer has, in our case, two neurons corresponding to our predicted classes. Between these two layers, we have hidden layers with a certain number of neurons. The neuron is connected with the neurons of the previous and next layers. It receives an input, applies an activation function and generates a signal to the next neurons. Each connection has a weight which controls the signal between the two neurons.

Data manipulation:

The values "na" are present only on categorical variables and they have been considered as another level. Additionally, the categorical variables have been coded as 0-1 (dummy). Furthermore, we normalized numerical variables which fall out in the range [-1, 1]. Lastly, the y response was dummised as well since Neural Network requires two final outputs.

Tuning parameters selection:

Neural Network has many tuning parameters which make hard to find the appropriate structure. The *number of hidden layer* and the *length of each hidden layers* control for the flexibility of the model. The greater they are, the more flexible is the model which could represent any continuous function on R^p. However it is a higher risk for overfitting.
The *size of the mini batch* and the number of *epochs* (one epoch is when the model pass through all the training data to compute the Gradient Descent once) control the learning rate of the model. The smaller is the size of the mini batch, the faster will be the computation time but it will not be optimal. The more epochs we have, the more the model will learn and risk to overfit.
Dropout is a regularization method where at each update step in the Gradient Descent, the network structure is changed. More precisely, each non-output neuron is independently dropped out with some probability which has to be determined. Similarly to Lasso and Ridge regression, there are *parameter penalties* which penalize the size of the weights and biases. *Parameter sharing* forces sets of parameters to have the same value to avoid overfitting. *Early stopping* allows to return the parameter value at the step where a minimum of the validation error is reached, and then stops the descent. The other parameters like the *activation function*, the *loss function* and the *optimizer* have to be determined.

To justify our choice for some parameters of the main structure of our model, we decided to start with a 1^{st} *hidden layer* with more neurons that the *input layer* to allow the model to capture as many patterns as possible. The value in *Dropout* should be lower on the 1^{st} *hidden layer* since we want to drop the less amount of the 1^{st} weights because they capture all the patterns of the *input layers*.

For the sake of time and place on the report, we decided to not show the value of the weights and biases. Furthermore, the validation split was chosen at 10%, the activation function and other parameters were kept constant and tried manually. Our best model was the following one:

- Structure:

  Input layer: 37 neurons.

  1^{st} hidden layer: 120 neurons.

  2^{nd} hidden layer: 80 neurons.

  3^{rd} hidden layer: 60 neurons.

  Output layer: 2 neurons.

- Activation function: *ReLU* was used for the three hidden layers and $softmax$ for the output layer.
- Loss function: *binary crossentropy*.
- Optimizer: *adam*.
- Batch size: 100.
- Early stopping: applied on *val accuracy* with *patience* = 20.
- Dropout: applied on the first hidden layer at 0.25 and 0.40 on the two others.
- Parameter penalty: The three hidden layers were regularized with *Ridge* (*L2*)* with the *penalty parameter* = 0.001.

Evaluation of the model:

From the CV accuracy on Figure 10, we can see that the best model is reached after 52 epochs. Since we used *Early stopping* with *patience* = 20 on the CV accuracy, and none of the 20 following models performed better than the one at *epoch* = 52, the process has stopped at *epoch* = 72. Indeed if the model continued to learn, we would have observed the train accuracy keeping increasing while the CV one stabilizing or even worst, decreasing (overfit). The accuracies can be found in Table 1.

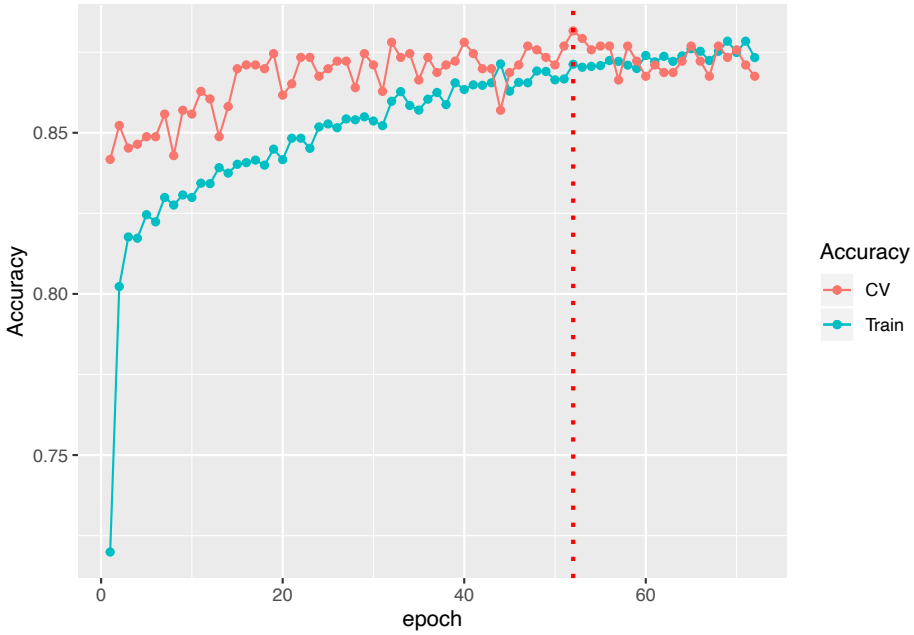

Figure 10: Train and cross-validation accuracy for Neural Network

3.5.2 Advantages

- Very flexible model.

3.5.3 Disadvantages

- High risk of overfitting.
- Difficult to interpret the coefficient.
- Difficult to find the good structure, need lots of experience.
- Computationally costly.

3.6 Ensembles

3.6.1 Model presentation

The Ensembles is an aggregation of different prediction models that could have better predictions than a model by itself. In a case of a regression, it computes the average of the predictions and for classification, it selects the most common class.
The formula to compute the probability of observation i to belong in class j is the following one:

$$\hat{p}_{ij} = \frac{1}{m} \sum_{k=1}^{m} \mathbf{I}\{y_{ik} = j\}$$

With

- i the i^{th} observation.
- j the j^{th} class.
- k the k^{th} model.
- m the number of models.
- y_{ik} the class of observation i according to model k.
- $\hat{p}_{ij}$ the probability that observation i belongs to class j.

Additionally, we can give more importance to some models by weighting them. The formula is modified as follow:

$$\hat{p}_{ij} = \frac{1}{m} \sum_{k=1}^{m} w_k * \mathbf{I}\{y_{ik} = j\}$$

With w_k the weight given to the model k.

Hence, $\hat{y}_i$ is the predicted class j with the highest probability $\hat{p}_{ij}$ at observation i where:

$$\hat{y}_i = \begin{cases} 1, & \text{if } \hat{p}_{i1} >= \hat{p}_{i0}. \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

For our Ensemble, we combined the three first XGBoost and the three first Neural Network with the highest accuracy on the Public leaderboard. The model which had the best accuracy on Kaggle (XGBoost) has a weight = 2 and the others have a weight = 1.

The accuracy can be found in Table 1.

3.6.2 Advantage

- Could reduce the prediction error variance.

3.6.3 Disadvantages

- Could overfit depending of the weights chosen.
- "Blackbox" method, the relationship between the predictors and the outcome variable is not transparent.

4 Results

4.1 Accuracies comparison

Table 1 highlights the differences between the accuracy of the train, CV, Public and Private Leaderboard.

Table 1: Accuracy table

	Train	CV	Public Leaderboard	Private Leaderboard
LDA	0.80835	0.80706	0.80688	0.81869
Random Forest	1	0.86125	0.86239	0.85958
XGBoost	0.88834	0.86969	0.87646	0.86327
SVM	0.84928	0.82184	0.81939	0.81903
Neural Network	0.87332	0.86753	0.86317	0.83545
Ensemble	-	-	0.88037	0.85991

4.2 Discussion

In the following, we analyzed our results for the models presented previously. The first part was written before the end of the competition and the second one after the end. In the first part, we compared our models based on the value of their train, CV and Public Leaderboard accuracies and on their variable importance plot. We ended the first part of our analysis in discussing briefly on our tries in feature engineering. In the second part, we analyzed the accuracies on the Private Leaderboard and evaluated our choice for the final model in the competition.

4.2.1 Before the end of the competition

Accuracies:

In Table 1, we can find the train, CV and Public leaderboard accuracies of all the models. They were ranked by the order of presentation of the models. Generally, the train accuracy of our models is higher than the CV accuracy. It is not a surprise as cross-validation is used to avoid overfitting the data. We can highlight the value of the train accuracy of the Random Forest is at the maximum, it is equal to 1. Since the trees are fully grown (until the final node contains only one observation), the predictions are exactly the value of the observation itself. We can see it overfits the data as the CV accuracy is much lower.

The model that had the best performance for the CV accuracy is the XGBoost, followed by the Neural Network. We expected this result as these two prediction algorithms are known to be highly performant. XGBoost is popular in the Kaggle competitions and should work well with structured data as ours. Whereas for the Neural Network, it should work well with unstructured data (pixels in an image). The Ensemble and both of them are also first between our models in the column Public leaderboard. The accuracy in the Public leaderboard is higher than the CV accuracy for the XGBoost contrary to the Neural Network. XGBoost gave better predictions than the Neural Network on the 30% of the test data but maybe it will not be as good after including the 70% remaining ones. For the Neural Network, the Public Leaderboard accuracy is much smaller than the CV accuracy. Even if many regularizations have been introduced in the model, it is possible that our model still overfits.

The Ensemble had the best accuracy on the Public leaderboard. We don't have a CV accuracy to compare it and it is possible that its accuracy decreases including the other 70% observations.

We still chose the Ensemble as our final model, because the Public leaderboard score was significantly better than any other.

Variable importance:

From the variable importance plot of our models (Figure 11), we can see that "time_spent" and "month" are the first variables for Random Forest, XGBoost and SVM. Additionally, Random forest and XGBoost have in common for the first six variables, "X4" and "age". It could mean that these variables are important in determining the outcome but it could also be due to that these two methods are in the tree family. Indeed, the order in SVM is quite different.

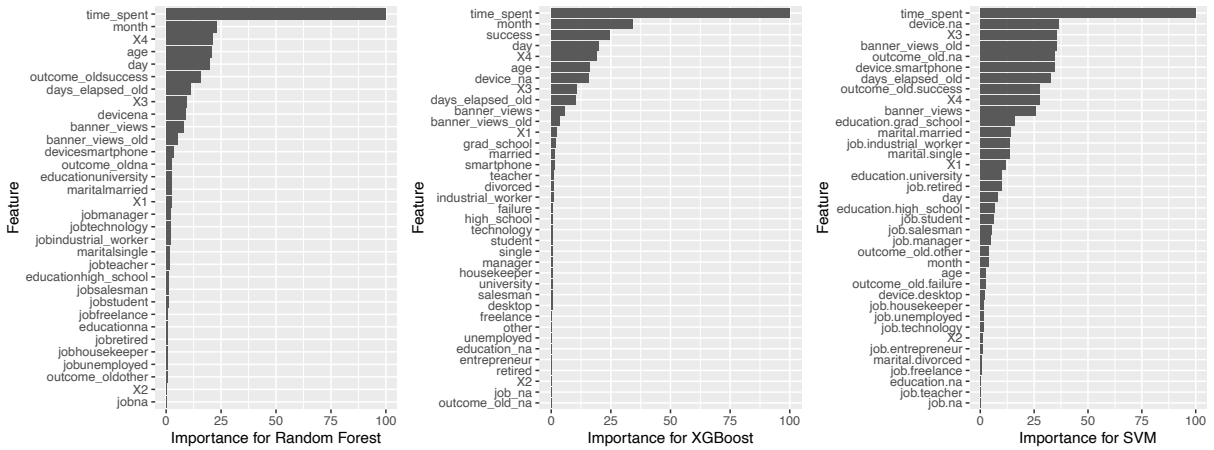

Figure 11: Variable importance plot for Random Forest, XGBoost and SVM

Features engineering:

In the beginning of the competition, we started to use simple prediction models such as logistic regression, LDA, etc. on our raw data. Then, we applied feature engineering to try to improve their performance (log transformation, scaling...) however the result was not satisfying. So, we decided to use raw data on the more complex models and to manipulate our data only when it was necessary. For example, we needed to transform the categorical data and the output in dummies for the Neural Network.

4.2.2 After the end of the competition

All the accuracies in the Private Leaderboard were lower than in the Public Leaderboard except for the model LDA. The final score obtained on the Private leaderboard ranked us as the second team, which is the same for the Public leaderboard. However, we should have chosen the model that maximize the CV accuracy, as it is a more robust result. Doing so, our best model is the XGBoost. Fortunately, our rank on the Private leaderboard would not have changed choosing the XGBoost model. It is still a takeaway for future competitions.

5 Conclusion

To conclude, we can see how difficult it is to find to the best model. Some models are more complex to tune than others and require more computational time to find the best combination of parameters. From the results, we see that the best models were those coming from the XGBoost and the Ensemble which were not discussed during the class. We are convinced that with more research on these models, especially the Ensemble, where different versions of it exist (e.g. Stacking), we would be able to capture more insights from the data and improve the accuracy on the Public/Private leaderboard.

6 Appendix

6.1 Appendix 1 : Exploratory analysis for the test set

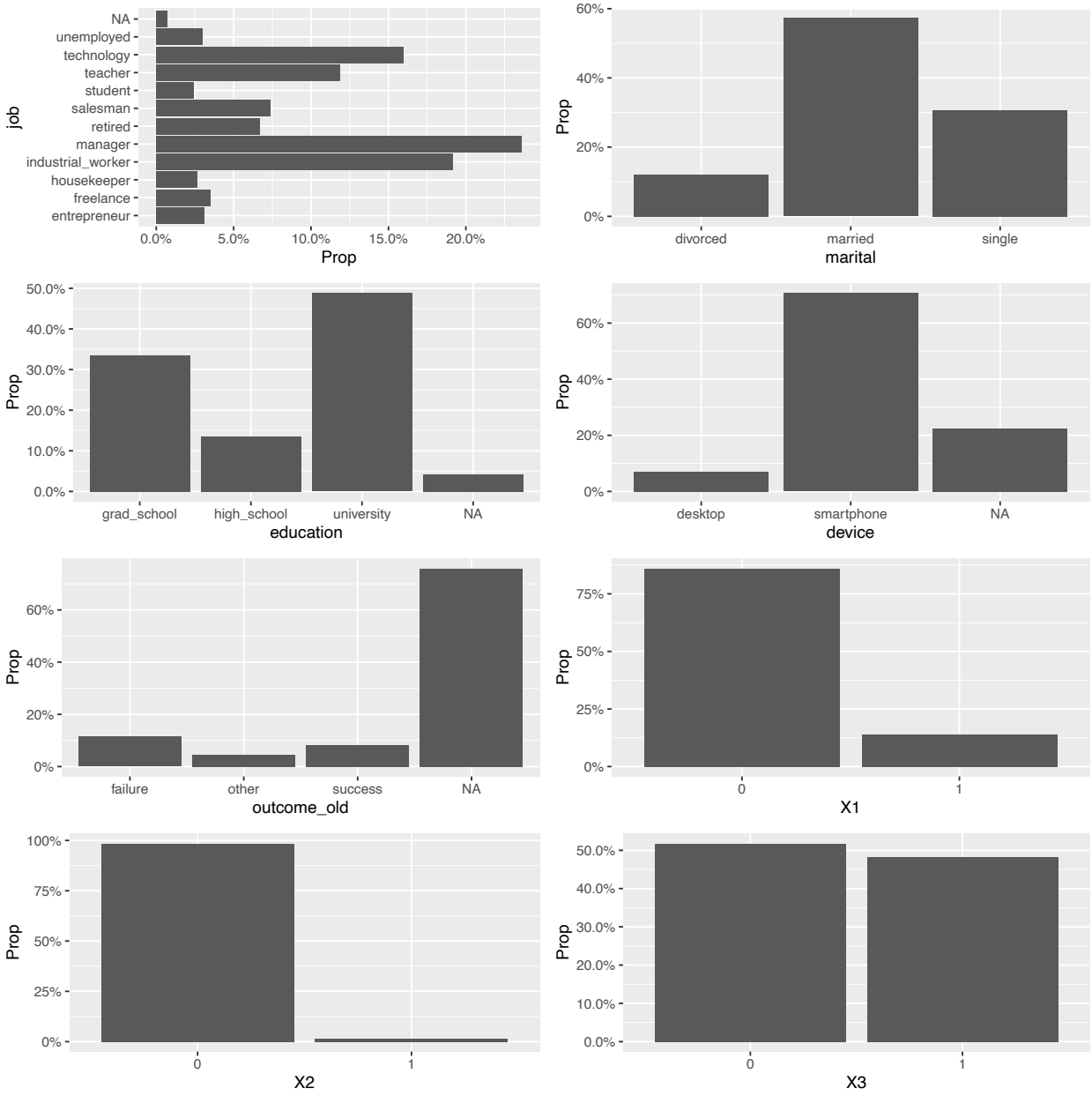

Figure 12: Barplots of the categorical variables with the proportion of target variable for the test set

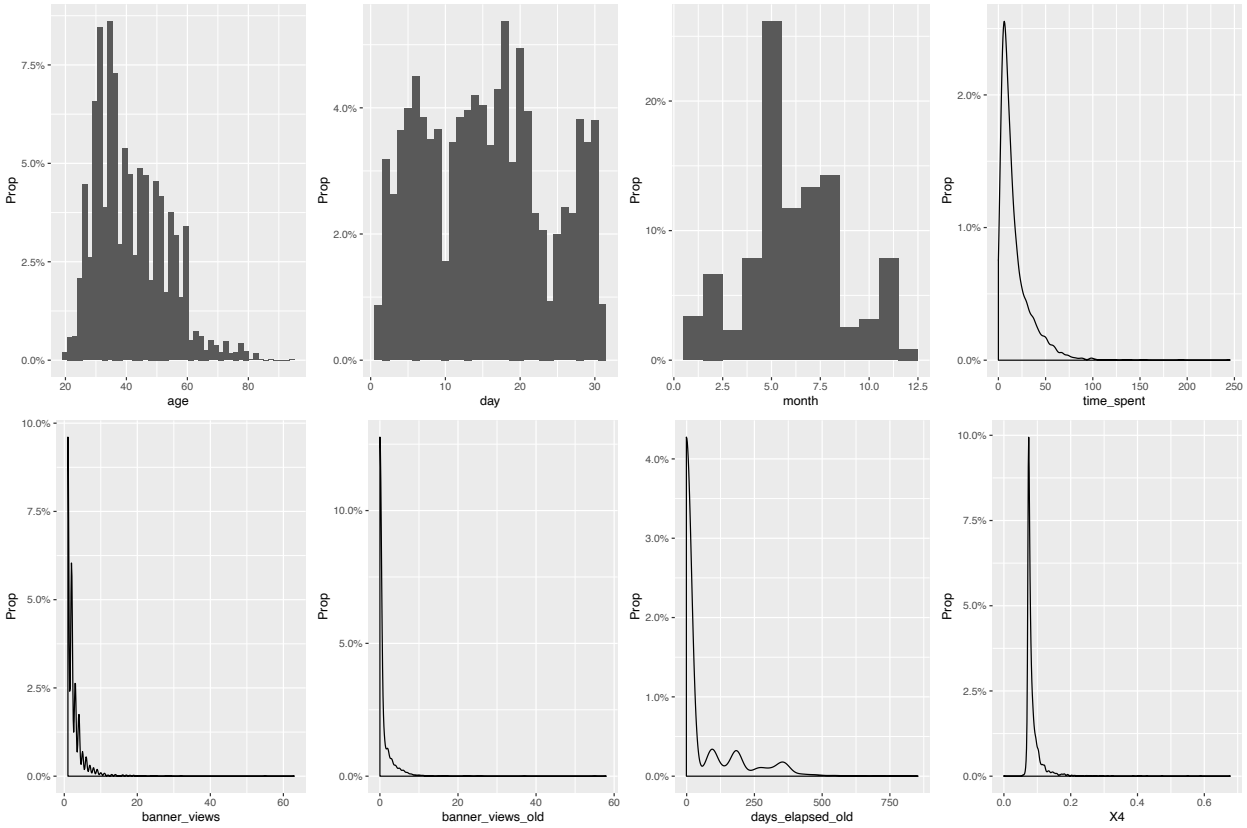

Figure 13: Density plots of the numerical variables with the proportion of target variable for the test set

Findings :

All the plots of the variables show their distribution are quite similar to the train set. Thus, we decided to use the raw data of the test set as such.

6.2 Appendix 2 : LDA

Model Presentation:

The Linear Discriminant Analysis (LDA) is a generative model. It gives the linear combination of all the variables with the biggest separation between the two classes.

Data manipulation:

We processed the data by standardizing the numerical variables.

We fitted the model to the training data and we obtained the estimate parameters of LDA. We highlighted one of the LDA returns, *Scaling*, which gives the coefficients of the linear discriminant. The discriminant function is a linear combination of all the coefficients c_j and our variables X_j.

$$LD1 = \sum_{j=1}^{p} c_j X_j$$

For the sake of clarity, we only show the result of the two first coefficients of the discriminant function.

```
##                         LD1
## X3              -0.5645104
## outcome_old.na -0.4615266
```

$$LD1 = c_{X3} \cdot X_{X3} + c_{outcomeold.na} \cdot X_{outcomeold.na} + ... = -0.5645 \cdot X_{X3} - 0.4615 \cdot X_{outcomeold.na} + ...$$

Then we predicted our model on the training set.

Evaluation of the model:

We can see the result of the LDA with the accuracy that can be found in Table 1.

Variable Importance:

In figure 14, we can observe a plot ranking the variables from the most important (top) to the less important (down).

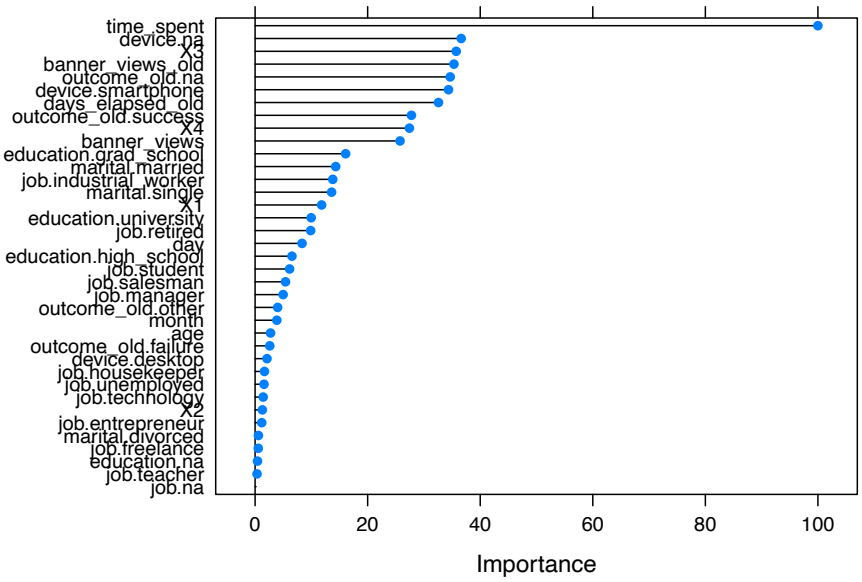

Figure 14: Variable importance plot for LDA

In the following, we mention some advantages and disadvantages of this model.

Advantages:

- LDA is easy and fast to compute.
- LDA is a stable estimation method.

Disadvantage:

- LDA assumes that the covariances matrices are the same for each class.