Credit card defaults prediction

Andrea

21 June 2020

Contents

Introduction

In this project, using a dataset containing six months information, we want to create a model that predicts credit card default payments. From a risk management perspective, an accurate prediction and assessment of the credit risk is of relevant importance for the bank issuing the credit card. The dataset we are going to use can be downloaded at the following url:

"https://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20of%20credit%20card%20clients.xls".

We have downloaded and manually saved a version of the *.xlsx* file in the "data" subfolder you can find within the github repo link we have provided together with the three files or at the following ulr:

"https://github.com/andrearoetti/Choose-Your-Own/tree/master/data".

Before getting started, we load the necessary packages for our work, after installing them in case they are not installed yet.

```
if (!require(tidyverse)) install.packages('tidyverse')
if (!require(lattice)) install.packages('lattice')
if (!require(caret)) install.packages('caret')
if (!require(readxl)) install.packages('readxl')
if (!require(gam)) install.packages('gam')
library(tidyverse)
library(lattice)
library(caret)
library(readxl)
library(gam)
```

We therefore use here below a relative path in order to read the excel file in R: for the sake of simplicity, we will simply call the dataset **credit_card**.

```
credit_card <- read_xlsx("data/default of credit card clients.xlsx")
```

Data cleaning

We first take a look of the first six rows to get a flavour of how the dataset is made.

```
head(credit_card)
```

```
## # A tibble: 6 x 25
##   ...1  X1     X2    X3    X4    X5    X6    X7    X8    X9    X10   X11   X12
##   <chr> <chr>  <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 ID    LIMIT~ SEX   EDUC~ MARR~ AGE   PAY_0 PAY_2 PAY_3 PAY_4 PAY_5 PAY_6 BILL~
## 2 1     20000  2     2     1     24    2     2     -1    -1    -2    -2    3913
## 3 2     120000 2     2     2     26    -1    2     0     0     0     2     2682
## 4 3     90000  2     2     2     34    0     0     0     0     0     0     29239
## 5 4     50000  2     2     1     37    0     0     0     0     0     0     46990
## 6 5     50000  1     2     1     57    -1    0     -1    0     0     0     8617
## # ... with 12 more variables: X13 <chr>, X14 <chr>, X15 <chr>, X16 <chr>,
## #   X17 <chr>, X18 <chr>, X19 <chr>, X20 <chr>, X21 <chr>, X22 <chr>,
## #   X23 <chr>, Y <chr>
```

We immediately notice that the first column may be a simple counter of the lines if every ID is different from each other: we confirm this with the code below.

```
nrow(credit_card)
```

```
## [1] 30001
```

```
n_distinct(credit_card$...1)
```

```
## [1] 30001
```

This does not add any value to our data: we are going to delete the first column. In the meanwhile, we acknowledge that our dataset has 30.000 records. Moreover, we also notice that the first line is actually a description of each column: we will save this line under another object called *col_description* and delete it from our dataset as well.

```
credit_card <- credit_card[,-1]
col_description <- credit_card[1,]
credit_card <- credit_card[-1,]
```

We now take the *col_description*, which is a tibble, and we transpose it so that it is easier to be shown and read.

```
class(col_description)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
t(col_description)
```

```
##     [,1]
## X1  "LIMIT_BAL"
## X2  "SEX"
## X3  "EDUCATION"
## X4  "MARRIAGE"
## X5  "AGE"
## X6  "PAY_0"
## X7  "PAY_2"
## X8  "PAY_3"
## X9  "PAY_4"
## X10 "PAY_5"
## X11 "PAY_6"
## X12 "BILL_AMT1"
## X13 "BILL_AMT2"
## X14 "BILL_AMT3"
## X15 "BILL_AMT4"
## X16 "BILL_AMT5"
## X17 "BILL_AMT6"
## X18 "PAY_AMT1"
## X19 "PAY_AMT2"
## X20 "PAY_AMT3"
## X21 "PAY_AMT4"
## X22 "PAY_AMT5"
## X23 "PAY_AMT6"
## Y   "default payment next month"
```

We now take the opportunity to describe the type of information we have in our dataset. First of all, since our data is taken from Taiwan, every money amount is in New Taiwan Dollars, also abbreviated as NT

Dollars or TWD. On average in the last year, the TWD has exchanged against USD with a rate of about 0.032, or about 30 TWD for 1 USD.

Columns description

- The first column (**X1** or "LIMIT_BAL") represent the maximum limit of each credit card.
- The second column (**X2** or "SEX") represents the gender: 1 = male and 2 = female.
- The Third column (**X3** or "EDUCATION") represents the education level: 1 = graduate school, 2 = university, 3 = high school, 4 = others, 5, 6 = unknown.
- The fourth column (**X4** or "MARRIAGE") represents the marital status: 1 = married, 2 = single, 3 = others.
- The fifth column (**X5** or "AGE") represents the age in years.
- From the column **X6** ("PAY_0") to the column **X11** ("PAY_6") we have repayment status from September 2005 back to April 2005 respectively ("PAY_0" is September, "PAY_2" is August, all the way to "PAY_6" that is April): -1 = pay duly, 1 = payment delay for one month, 2 = payment delay for two months, ... 8 = payment delay for 8 months, 9 = payment delay for 9 months or more.
- From the column **X12** ("BILL_AMT1") to the column **X17** ("BILL_AMT6"), with the same logic we have the amount of bill statements from September 2005, in "BILL_AMT1", all the way to April 2005 ("BILL_AMT6").
- From the column **X18** ("PAY_AMT1") to the column **X23** ("PAY_AMT6"), again with the same logic, we have the amounts of previous payment from September 2005, in "PAY_AMT1", all the way to April 2005 ("PAY_AMT6").
- Finally, in the last column (**Y** or 'default payment next month') we have the outcome 1 in case of default, 0 otherwise.

Data tidying and partitioning

Based on columns definition, we first change the columns class to factors (for sex, education, marriage and the outcome 'default payment next month') or numbers (all the remaining).

```
credit_card[,-c(2,3,4,24)] <- as_tibble(sapply(credit_card[,-c(2,3,4,24)], as.numeric))
credit_card$X2 <- as.factor(credit_card$X2)
credit_card$X3 <- as.factor(credit_card$X3)
credit_card$X4 <- as.factor(credit_card$X4)
credit_card$Y <- as.factor(credit_card$Y)
```

The ultimate goal of this project is to predict, on an unknown dataset, the outcome of column Y given all the Xs. To do this, we split our dataset into two parts: one, called *historic* will be treated as historic series of data at our disposition, where we will do our data analysis and build our model. The second one, called *validation*, will be treated as unknown and only used to evaluate the performance of our final model. The proportion chosen is 80-20% in order to have enough data to build and train our model but also a consistent dataset for evaluating it.
To permit the replica of our work, we set the seed before creating the index.

```
set.seed(1, sample.kind = "Rounding")
validation_index <- createDataPartition(y = credit_card$Y, times = 1,p = 0.2, list = FALSE)
historic <- credit_card[-validation_index,]
validation <- credit_card[validation_index,]
```

Exploratory data analysis

We are now ready to start our data analysis on the *historic* dataset as if it were our historical series of records to use for our model. First, here below a summary of the statistics.

```
summary(historic)
```

```
##        X1          X2         X3         X4              X5
##  Min.   : 10000   1: 9495   0:   12   0:   41   Min.   :21.00
##  1st Qu.: 50000   2:14504   1: 8460   1:10855   1st Qu.:28.00
##  Median :140000             2:11244   2:12844   Median :34.00
##  Mean   :167286             3: 3916   3:  259   Mean   :35.44
##  3rd Qu.:240000             4:  104             3rd Qu.:41.00
##  Max.   :800000             5:  220             Max.   :75.00
##                             6:   43
##        X6                  X7                X8                X9
##  Min.   :-2.00000   Min.   :-2.0000   Min.   :-2.0000   Min.   :-2.0000
##  1st Qu.:-1.00000   1st Qu.:-1.0000   1st Qu.:-1.0000   1st Qu.:-1.0000
##  Median : 0.00000   Median : 0.0000   Median : 0.0000   Median : 0.0000
##  Mean   :-0.01433   Mean   :-0.1333   Mean   :-0.1699   Mean   :-0.2257
##  3rd Qu.: 0.00000   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000
##  Max.   : 8.00000   Max.   : 8.0000   Max.   : 8.0000   Max.   : 8.0000
##
##       X10               X11               X12               X13
##  Min.   :-2.0000   Min.   :-2.0000   Min.   :-165580   Min.   :-69777
##  1st Qu.:-1.0000   1st Qu.:-1.0000   1st Qu.:   3526   1st Qu.:  2948
##  Median : 0.0000   Median : 0.0000   Median :  22136   Median : 20911
##  Mean   :-0.2678   Mean   :-0.2973   Mean   :  51070   Mean   : 49004
##  3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.:  67054   3rd Qu.: 63872
##  Max.   : 8.0000   Max.   : 8.0000   Max.   : 746814   Max.   :646770
##
##       X14               X15               X16               X17
##  Min.   :-157264   Min.   :-170000   Min.   :-81334   Min.   :-339603
##  1st Qu.:   2628   1st Qu.:   2314   1st Qu.:  1744   1st Qu.:   1231
##  Median :  19945   Median :  18926   Median : 17987   Median :  16847
##  Mean   :  46821   Mean   :  42943   Mean   : 39972   Mean   :  38555
##  3rd Qu.:  59846   3rd Qu.:  53826   3rd Qu.: 49893   3rd Qu.:  48901
##  Max.   :1664089   Max.   : 616836   Max.   :587067   Max.   : 699944
##
##       X18               X19                X20               X21
##  Min.   :     0   Min.   :      0.0   Min.   :     0   Min.   :     0
##  1st Qu.:   980   1st Qu.:    835.5   1st Qu.:   390   1st Qu.:   279
##  Median :  2100   Median :   2005.0   Median :  1800   Median :  1500
##  Mean   :  5603   Mean   :   5802.5   Mean   :  5084   Mean   :  4750
##  3rd Qu.:  5003   3rd Qu.:   5000.0   3rd Qu.:  4514   3rd Qu.:  4006
##  Max.   :873552   Max.   :1684259.0   Max.   :889043   Max.   :621000
##
##       X22               X23          Y
##  Min.   :     0   Min.   :     0   0:18691
##  1st Qu.:   256   1st Qu.:   101   1: 5308
##  Median :  1500   Median :  1500
##  Mean   :  4765   Mean   :  5215
##  3rd Qu.:  4034   3rd Qu.:  4000
##  Max.   :417990   Max.   :528666
```

##

We notice that our population is aged, on average, almost 35.5, with a range that goes from 21 to 75. 50% of the cases are between 28 and 41 years old. Moreover, the average credit card limit is TWD 167.286 (around USD 5.500), with a range between TWD 10.000 (something more than USD 300) and 800.000 (more than USD 25.500). Among other statistics, it would be interesting understanding the rate of credit card defaults, the proportion male/female and the proportion of married individuals.

```
mean(historic$Y == "1")
```

```
## [1] 0.2211759
```

```
mean(historic$X2 == "1")
```

```
## [1] 0.3956415
```

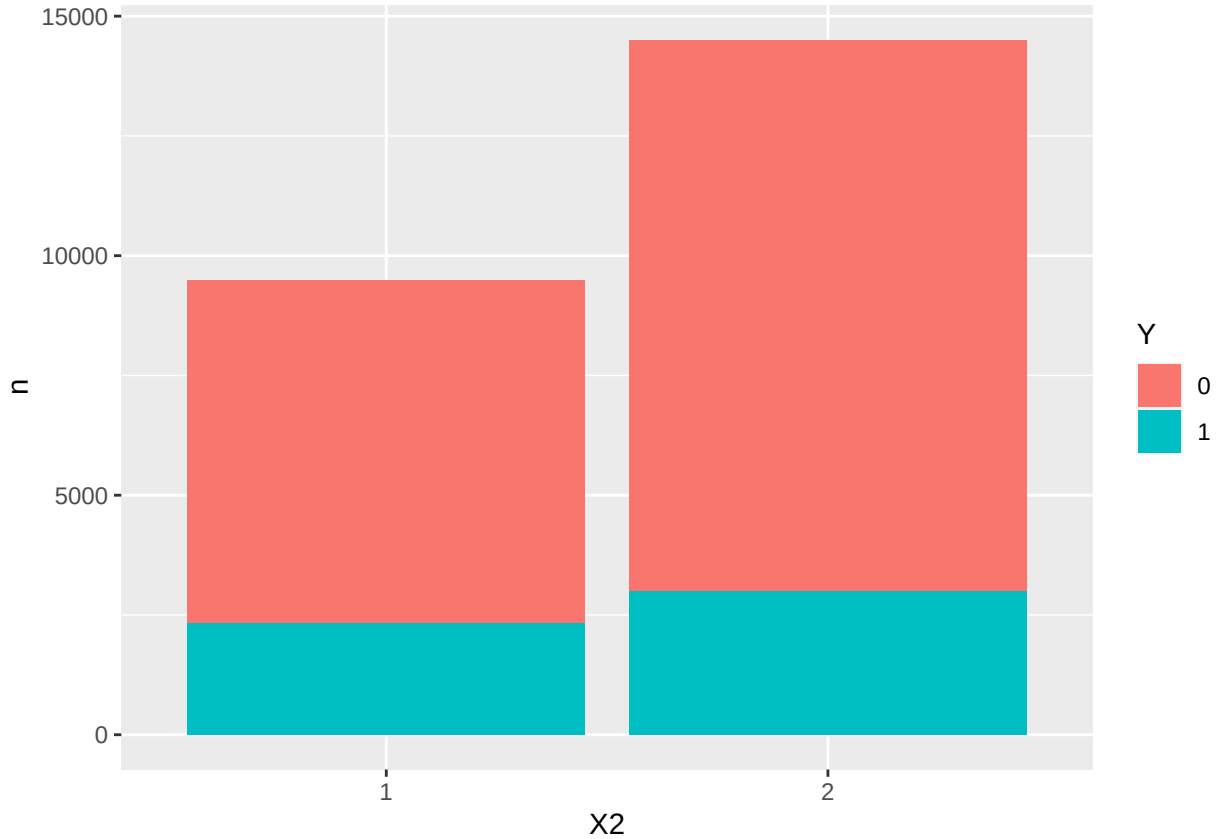
```
mean(historic$X4 == "1")
```

```
## [1] 0.4523105
```

We understand that the 22.1% of the cases has a credit card default payment in the next month, the 39.6% is male and the 45.3% of individuals is married. More than that, before focusing on previous months data, it would be interesting understanding whether and how there is a correlation between the defaults and the other factors as gender, education level, marriage status, age or credit card limit.

Gender

We first check how the default payments are divided by gender and we do it with the following barplot.



It looks like the default payments for male (X2 = 1) are, on percentage, slightly higher than for females (X2

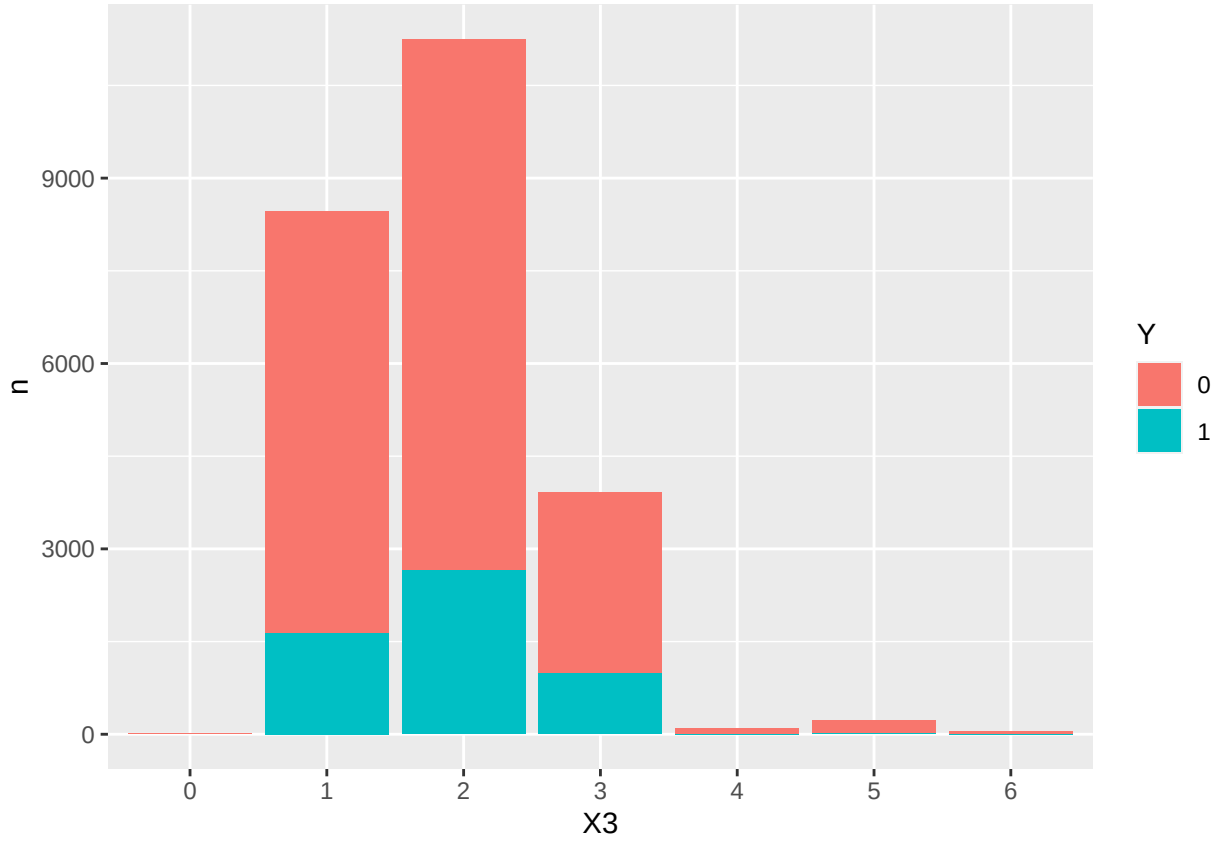
= 2), despite the fact that in absolute numbers they are a bit less. The below table confirms our suspect.

```
historic %>% group_by(X2) %>% summarize(rate = mean(Y == "1"), n = n())
```

```
## # A tibble: 2 x 3
##   X2     rate     n
##   <fct> <dbl> <int>
## 1 1     0.244  9495
## 2 2     0.206 14504
```

Education level

We now make the same exercise on education level, here is the barplot.



This last plot is more difficult to read as the number of individuals in the different levels varies a lot. We therefore make a summary table of the default rates by education.

```
historic %>% group_by(X3) %>% summarize(rate = mean(Y == "1"), n = n())
```

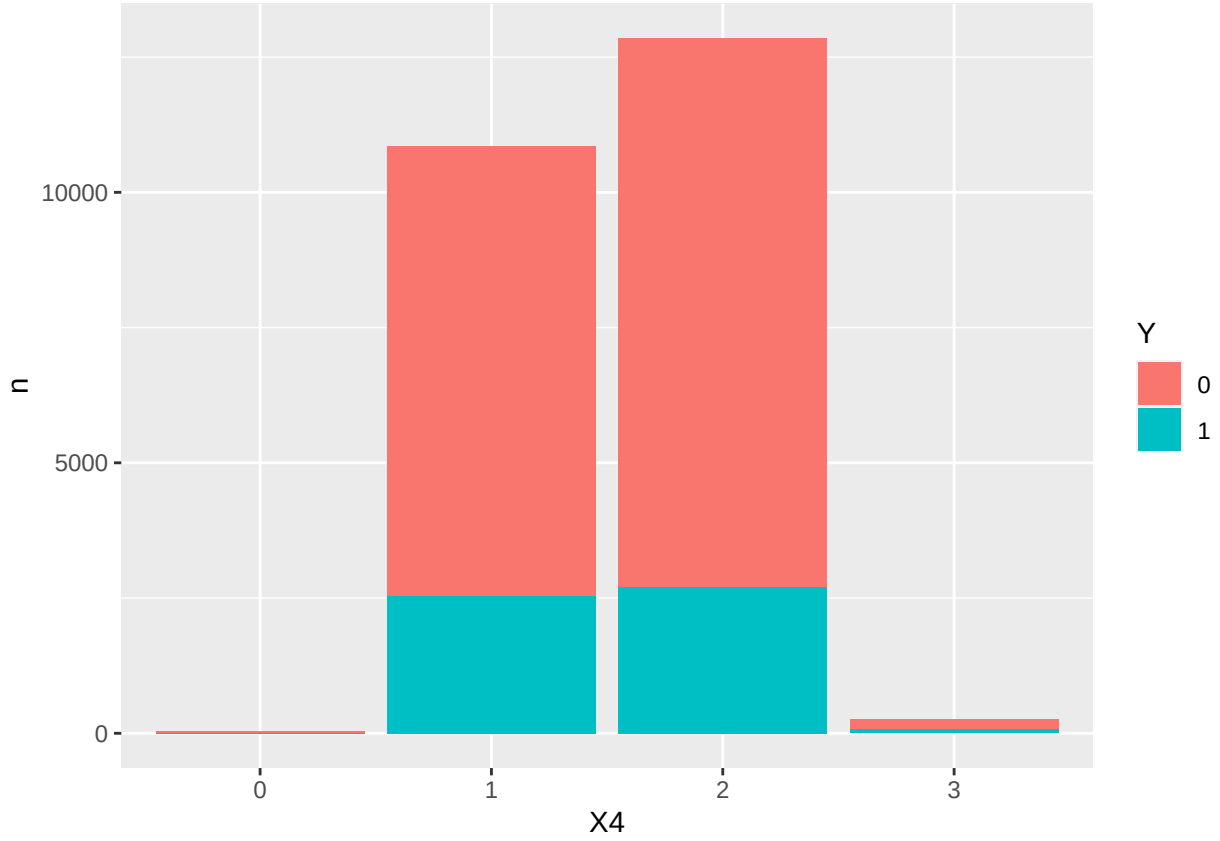
```
## # A tibble: 7 x 3
##   X3      rate     n
##   <fct>  <dbl> <int>
## 1 0     0          12
## 2 1     0.194   8460
## 3 2     0.236  11244
## 4 3     0.252   3916
## 5 4     0.0673   104
```

```
## 6 5     0.0682   220
## 7 6     0.163     43
```

From this table it appears more clearly that actually, among the three main categories (graduate school = 1, university = 2, high school = 3), the education level has an impact on defaults rate: the higher the degree, the lower the rate. No relevant paths can be deducted from the other categories, since the number of individuals is not significant.

Marital status

As done previously, we try to understand if also the marriage status may impact the default payments rate.



Similarly to the gender situation, it appears that in case an individual is married (X4 = 1), the default rate is slightly higher than in the case she/he is single (X4 = 2), despite the fact in our dataset we have overall more cases of defaults pertaining to singles. We confirm our guess with the table below.

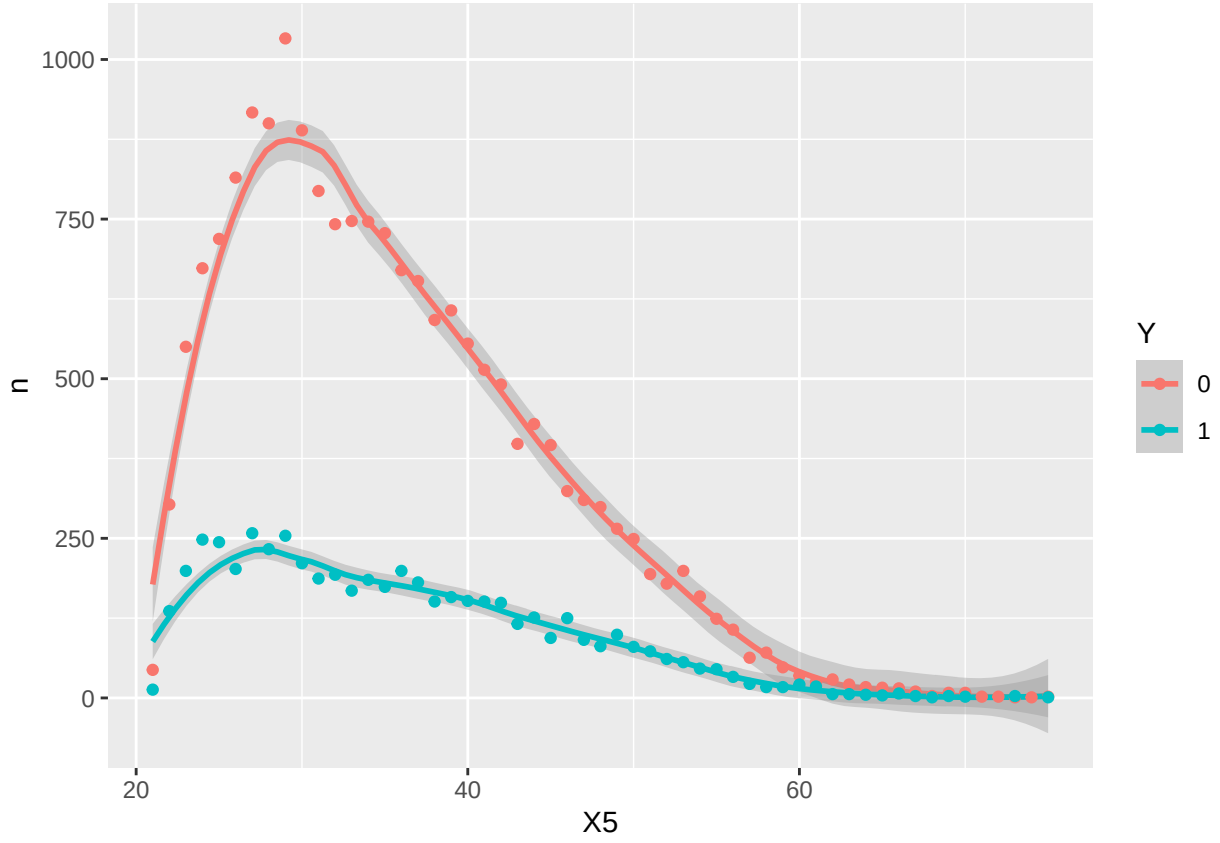
```
historic %>% group_by(X4) %>% summarize(rate = mean(Y == "1"), n = n())
```

```
## # A tibble: 4 x 3
##   X4      rate     n
##   <fct>  <dbl> <int>
## 1 0     0.0976    41
## 2 1     0.233  10855
## 3 2     0.210  12844
## 4 3     0.270    259
```

Other marriage status do not appear to be statistically relevant due to the low number of cases.

Age

To understand whether age has an impact on default rate or not we check if the age distribution functions for the two possible outcomes we want to predict (Y = 1 or 0) have shapes centred at different ages or follow different paths. We make use of a smooth line since age is not a factor as gender, marital status or education level. Here below the graph.



From this plot it looks like the two curves follow a similar proportional path with no sensible evidences of age disparity (beside a different number of cases by age, of course). We summarise in the table below the average age for default and non-default cases.

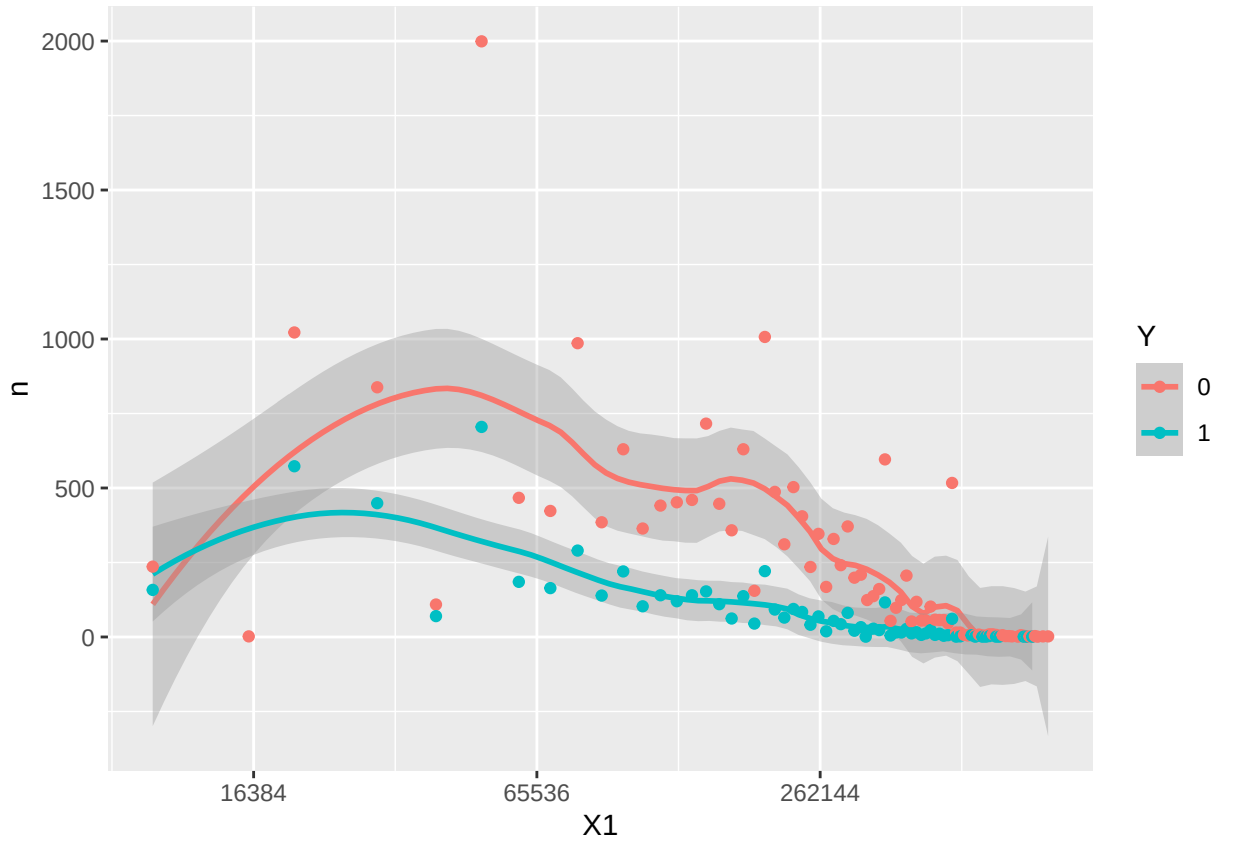
```
historic %>% group_by(Y) %>% summarize(rate = mean(X5), n = n())
```

```
## # A tibble: 2 x 3
##   Y      rate     n
##   <fct> <dbl> <int>
## 1 0      35.4 18691
## 2 1      35.6  5308
```

In our kind of Bayesan analysis it appears that, given the default or not, does not really help in understanding the age of the individual on average, like saying that a younger age does not mean a higher risk and vice-versa. We will therefore not use this predictor any further.

Credit card limit

Our last check, in this chapter, is on the relation between credit card limit and default payments and here we go with the plot. We have used, as for the age, a smooth line, with the x-axis (card limit) rescaled through *log2* transformation to see top peaks less sparse.



This time it looks like for smaller limits we have higher percentages of defaults. We try to confirm this with an ex-post check on average credit card limits for defaults and non-defaults cases.

```
historic %>% group_by(Y) %>% summarize(rate = mean(X1), n = n())
```

```
## # A tibble: 2 x 3
##   Y        rate     n
##   <fct>   <dbl> <int>
## 1 0     177860. 18691
## 2 1     130052.  5308
```

The table confirms our intuition, since the average limit for default cases is lower. This indicates that, perhaps, who asks for an increase in her/his limit, has actually an even higher economic power than what conceded.

Model

We are now ready to build our model. We will exploit the easiness of use of the 'caret' package in order to apply different models to our predictors and then we will choose the best performing ones. Since every model will calculate its accuracy by performing cross-validation on bootstrap samples, we do not need to further split our historical dataset into train and test sets. We will tune, instead, the chosen models in case they have tuning parameters.
Since our categorical outcome (0 or 1 for credit card default payment, column Y) is unbalanced and contains a majority of non-defaults (Y = 0), the benchmark result required to be overperformed is the accuracy rate given by all predictions equal to zero.

```
zeros <- sample(0, nrow(historic), replace = TRUE, prob = 1)
benchmark <- mean(zeros == historic$Y)
benchmark
```

```
## [1] 0.7788241
```

In this sense, the 'Kappa' parameter of every model will be fundamental, being a parameter normalized on the expected accuracy, namely the random chance. It measure how many prediction we are getting correct on top of randomness: of course, the higher the better.
In the previous chapter we have analysed some possible and curious relations between personal information such as gender or marital status and the probability of default payments. From now on, to make it simple, we will call the information we have previously decided to include in our model (gender, education level, marital status and also credit card limit, with the only exclusion of the age) personal information.
We notice that the remaining columns of the datasets, that we have not analysed yet, may be clustered into three groups: monthly repayment status (from X6 to X11), monthly bills (from X12 to X17) and monthly payments (from X18 to X23). To summarise the information contained there, and also to reduce the number of predictors (hopefully, in a meaningful way), we add three columns to our table **historic**, called *sum_delays*, *billed_tot* and *paid_tot* calculated, respectively, as the sum of the columns from X6 to X11, from X12 to X17 and from X18 to X23.

```
historic <- historic %>% mutate(sum_delays = X6 + X7 + X8 + X9 + X10 + X11,
                                billed_tot = X12 + X13 + X14 + X15 + X16 + X17,
                                paid_tot = X18 + X19 + X20 + X21 + X22 + X23)
```

LDA and GamLoess

We are now ready to test the accuracy of a bunch of models on our predictors. We exclude from the list of models, among the others, the K-nearest neighbors ("knn"), Quadratic Discriminant Analysis ("qda"), that have shown not to work on this kind of data, Adaboost, for computational time reasons, and, for the time being, Random Forest ("rf"). We train the models on personal information, *sum_delays*, *billed_tot* and *paid_tot*, then we show how do they fit.

```
models <- c("glm", "lda", "naive_bayes", "svmLinear", "gamLoess")
fits <- lapply(models, function(model){
               train(Y ~ X1 + X2 + X3 + X4 + sum_delays + billed_tot + paid_tot,
               method = model, data = historic)})
fits
```

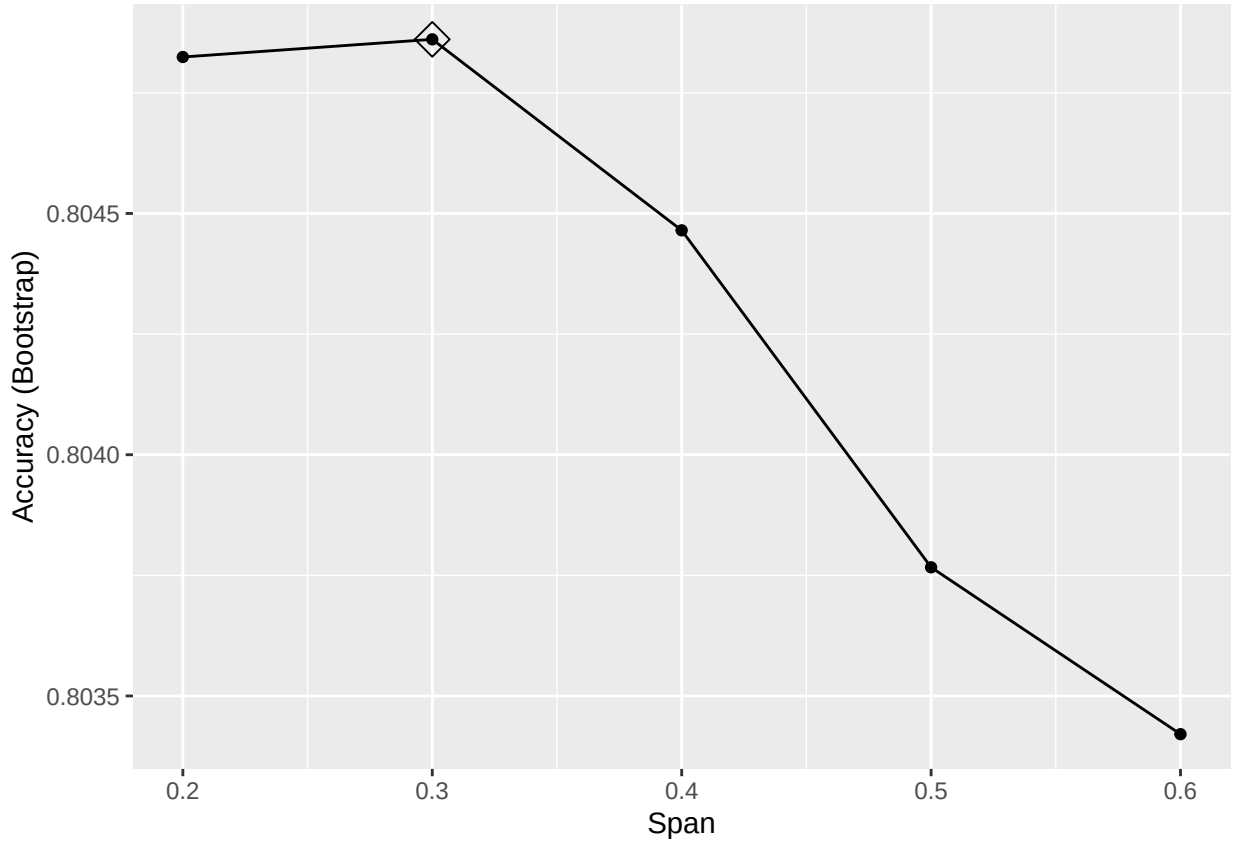
```
## [[1]]
## Generalized Linear Model
##
## 23999 samples
##     7 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 23999, 23999, 23999, 23999, 23999, 23999, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7978622  0.1902544
##
##
```

```
## [[2]]
## Linear Discriminant Analysis
##
## 23999 samples
##     7 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 23999, 23999, 23999, 23999, 23999, 23999, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8001971  0.2037555
##
##
## [[3]]
## Naive Bayes
##
## 23999 samples
##     7 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 23999, 23999, 23999, 23999, 23999, 23999, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.4638491  0.1074483
##    TRUE      0.7924961  0.1782719
##
## Tuning parameter 'laplace' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = TRUE
##  and adjust = 1.
##
## [[4]]
## Support Vector Machines with Linear Kernel
##
## 23999 samples
##     7 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 23999, 23999, 23999, 23999, 23999, 23999, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.779431  0
##
```

```
## Tuning parameter 'C' was held constant at a value of 1
##
## [[5]]
## Generalized Additive Model using LOESS
##
## 23999 samples
##     7 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 23999, 23999, 23999, 23999, 23999, 23999, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8041367  0.2797089
##
## Tuning parameter 'span' was held constant at a value of 0.5
## Tuning
##  parameter 'degree' was held constant at a value of 1
```

We see that the best models in terms of accuracy and kappa are Linear Discriminant Analysis ("lda") and Generalized Additive Model using LOESS ("gamLoess"): both of them overperform our benchmark accuracy. While the former has no tuning parameters because already construct to maximize the accuracy, the latter can be further improved. We therefore set a range for the parameter 'span' from 0.2 to 0.6 (default is 0.5), while we keep the degree at 1 (default value). We plot the result.

```
gridloess <- expand.grid(span = seq(0.2, 0.6, len = 5), degree = 1)
train_gamloess <- train(Y ~ X1 + X2 + X3 + X4 + sum_delays + billed_tot + paid_tot,
                        method = "gamLoess",
                        data = historic,
                        tuneGrid = gridloess)
ggplot(train_gamloess, highlight = TRUE)
```



We also separately train the LDA model.

```
train_lda <- train(Y ~ X1 + X2 + X3 + X4 + sum_delays + billed_tot + paid_tot,
                   method = "lda",
                   data = historic)
```

Random Forest and naive Bayes

While working on the dataset we have discovered that the Random Forest ("rf") and naive Bayes ("naive_bayes") models work pretty well if applied only on the predictor *sum_delays*. We show it here below.

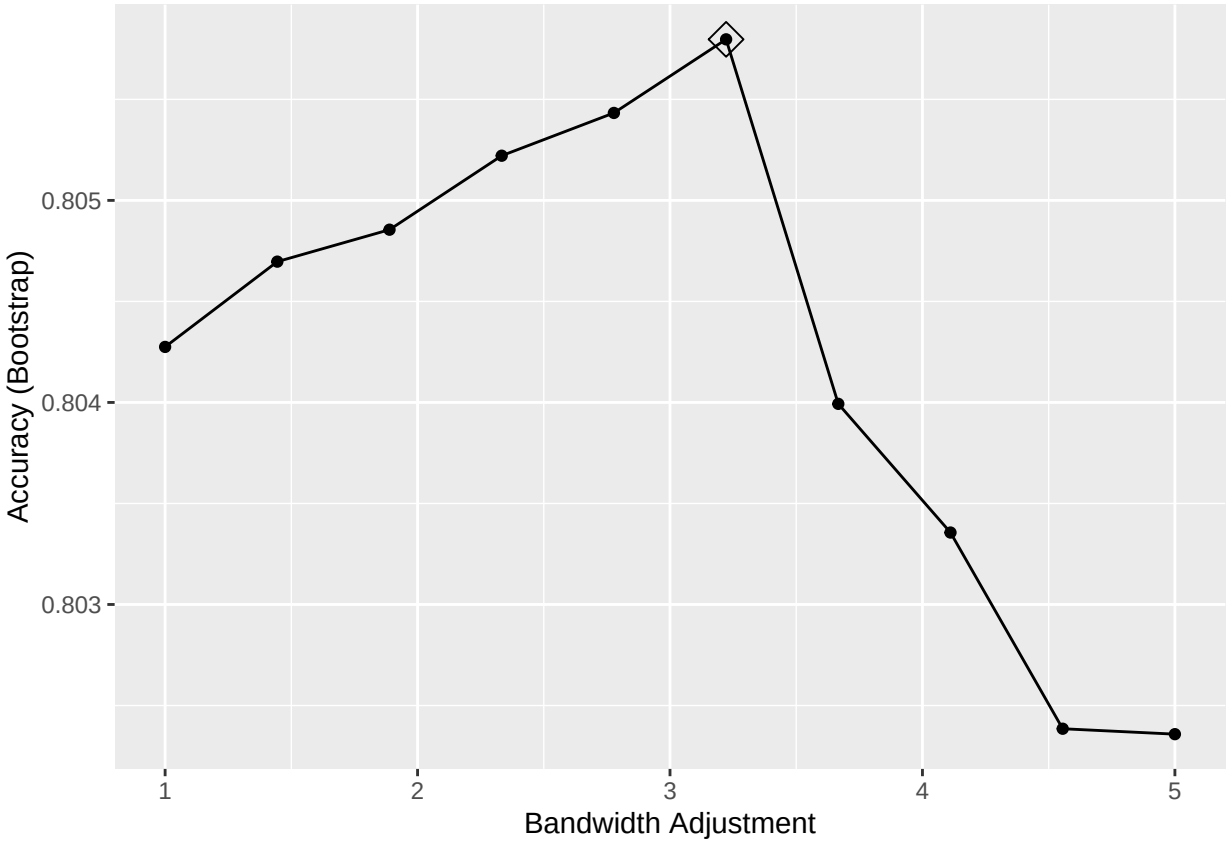
```
models2 <- c("naive_bayes", "rf")
fits2 <- lapply(models2, function(model){
                train(Y ~ sum_delays,
                      method = model,
                      data = historic)})
fits2
```

```
## [[1]]
## Naive Bayes
##
## 23999 samples
##     1 predictor
##     2 classes: '0', '1'
##
## No pre-processing
```

```
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 23999, 23999, 23999, 23999, 23999, 23999, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.8003236  0.2279685
##    TRUE      0.8041568  0.3082952
##
## Tuning parameter 'laplace' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = TRUE
##  and adjust = 1.
##
## [[2]]
## Random Forest
##
## 23999 samples
##     1 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 23999, 23999, 23999, 23999, 23999, 23999, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8052319  0.2981814
##
## Tuning parameter 'mtry' was held constant at a value of 2
```

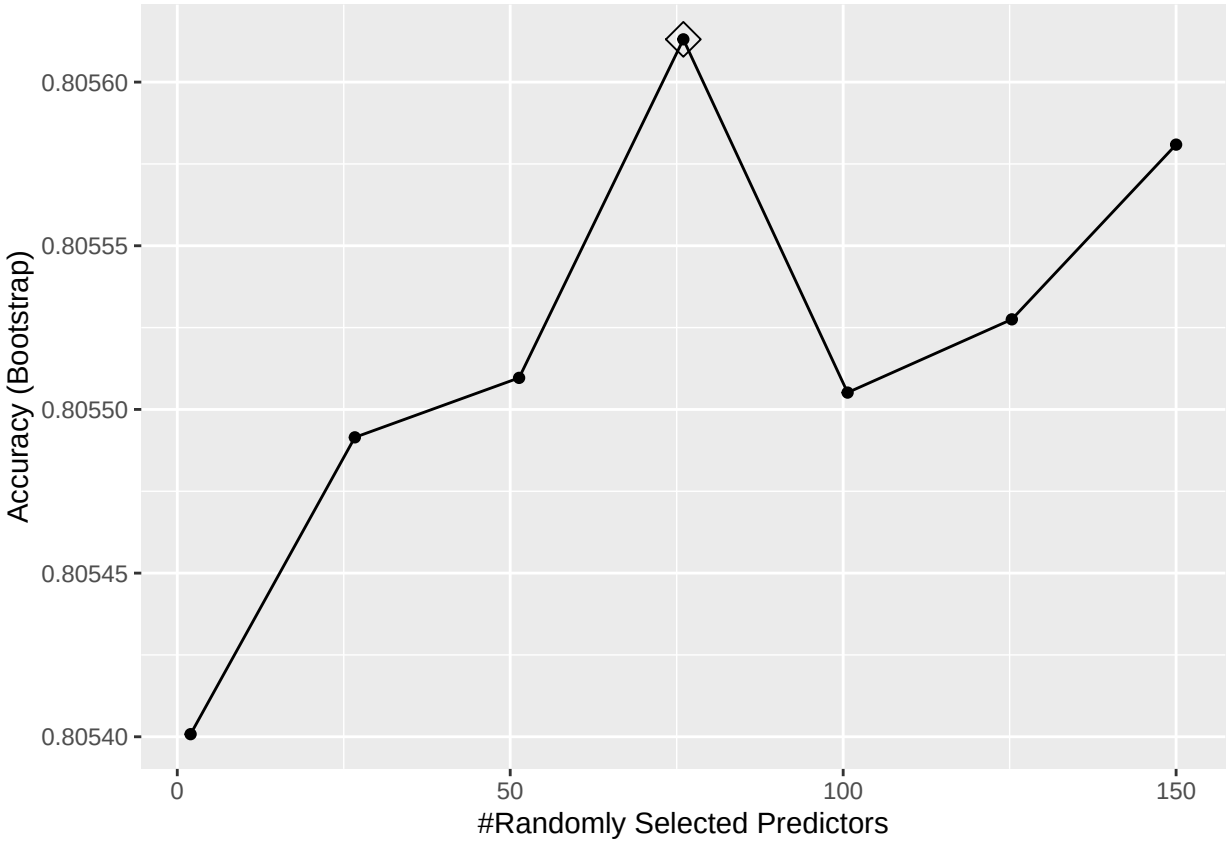
Both the models contain tuning parameters and we train them to master their performances. In the naive Bayes model we keep the parameter 'laplace' equal to 0 and 'usekernel' TRUE, while we make 'adjust' vary from 1 to 5 (10 values). We plot the result.

```
gridbayes <- expand.grid(laplace = 0,
                         usekernel = TRUE,
                         adjust = seq(1, 5, len = 10))
train_bayes <- train(Y ~ sum_delays,
                     method = "naive_bayes",
                     data = historic,
                     tuneGrid = gridbayes)
ggplot(train_bayes, highlight = TRUE)
```



We do the same exercise on the Random Forest model. We let the 'mtry' parameter vary from 2 (default) to 150, with 7 values. We also use the argument 'nodesize' = 1 to allow small sizes to our nodes, in case we have few extreme istances.

```
gridrf <- data.frame(mtry = seq(2, 150, len = 7))
train_rf <- train(Y ~ sum_delays,
                  method = "rf",
                  data = historic,
                  nodesize = 1,
                  tuneGrid = gridrf)
ggplot(train_rf, highlight = TRUE)
```



Result

Finally we are ready to test the accuracy of our models on the **validation** set. We then predict the outcome for the credit card default payments with LDA and GamLoess models using, as predictors, the personal information, *sum_delays*, *billed_tot* and *paid_tot*, and naive Bayes and Random Forest using the only predictor *sum_delays*. Before the evaluation, we have to create in the validation set the required predictors.

```
validation <- validation %>% mutate(sum_delays = X6 + X7 + X8 + X9 + X10 + X11,
                                    billed_tot = X12 + X13 + X14 + X15 + X16 + X17,
                                    paid_tot = X18 + X19 + X20 + X21 + X22 + X23)
```

And now we apply the methods, calculating every accuracy.

```
pred_lda <- predict(train_lda, validation)
pred_gamloess <- predict(train_gamloess, validation)
pred_bayes <- predict(train_bayes, validation)
pred_rf <- predict(train_rf, validation)
acc_lda <- confusionMatrix(data = pred_lda,
                           reference = validation$Y)$overall["Accuracy"]
acc_gamloess <- confusionMatrix(data = pred_gamloess,
                                reference = validation$Y)$overall["Accuracy"]
acc_bayes <- confusionMatrix(data = pred_bayes,
                             reference = validation$Y)$overall["Accuracy"]
acc_rf <- confusionMatrix(data = pred_rf,
```

```
                            reference = validation$Y)$overall["Accuracy"]
k_lda <- confusionMatrix(data = pred_lda,
                             reference = validation$Y)$overall["Kappa"]
k_gamloess <- confusionMatrix(data = pred_gamloess,
                                  reference = validation$Y)$overall["Kappa"]
k_bayes <- confusionMatrix(data = pred_bayes,
                               reference = validation$Y)$overall["Kappa"]
k_rf <- confusionMatrix(data = pred_rf,
                            reference = validation$Y)$overall["Kappa"]
```

We end up with the following final table.

```
target <- mean(validation$Y == "0")
accuracies <- tibble(method = c("target", "LDA", "GamLoess", "naive Bayes", "Random Forest"),
                     accuracy = c(target, acc_lda, acc_gamloess, acc_bayes, acc_rf),
                     kappa = c(0, k_lda, k_gamloess, k_bayes, k_rf))
accuracies %>% knitr::kable()
```

method	accuracy	kappa
target	0.7787035	0.0000000
LDA	0.8002000	0.2066889
GamLoess	0.8071988	0.3052016
naive Bayes	0.8086986	0.3230688
Random Forest	0.8071988	0.3093628

Conclusion

Despite the fact that we succeeded in overperforming the target accuracy given by the random chance as well as in achieving non-zero kappas, we have found it very difficult to sensibly improve our model performance. One reason for this is that working on an binary outcome where there is a strong unbalance is more difficult than in the 50-50% case.
Moreover, despite the fact that we have taken into account basically all the predictors, with the only exclusion of the age, we have struggled in finding a useful way to summarize the information on six-months bill statements (columns from X12 to X17) and six-months payments (columns from X18 to X23). We have tried making the differences between billed and paid amounts in order to get the unpaid amount that could have been an indicator of possible defaults, if high; we have tried to rescale this difference and of the sum of total bills on the maximum limit amount in order to check who reached the limit and many other. Unfortunately, despite they look quite rough, the simple sums that we have reported as *billed_tot* and *paid_tot* were the most effective with our methods.
Another remark, as with the last two models, trained only on the predictor *sum_delays*, we have reached a similar and even better performance than with the models trained on all the predictors, *sum_delays* has shown a great importance as a summary of the repayment status in the six-months information at our disposal. The table below confirms it, showing the case of Random Forest model, if we had trained it with all the predictors instead of only *sum_delays*.

```
train_rf2 <- train(Y ~ X1 + X2 + X3 + X4 + sum_delays + billed_tot + paid_tot,
                   method = "rf",
                   data = historic,
                   nodesize = 1)
varImp(train_rf2)
```

```
## rf variable importance
##
##              Overall
## sum_delays 100.00000
## paid_tot    30.58567
## billed_tot  24.15055
## X1          21.51430
## X22          2.28636
## X31          1.47440
## X32          1.38538
## X42          1.27345
## X41          1.27175
## X33          1.16729
## X35          0.51514
## X43          0.29544
## X34          0.02795
## X36          0.00000
```

In conclusion, among the possible improvements to this model that does not leave us completely satisfied, we mention possible ensembles techniques, with the models already tested, and, of course, with a somehow better use of the six-months bills and payments data at our disposal.