Taiwan Credit Card Payment Default Identification

Capstone Project Report for HarvardX Professional Data Science Certificate

Klaus Puchner

June 10th, 2019

Contents

1 Executive Summary

In this project we will use this data and format/tidy the data in a first step for an exploratory data analysis (EDA). After the EDA, machine leaning algorithms for this classification problem will be used to predict whether a customer payment default occurs or not. For this project, the "default of creadit card clients data set" from the **UCI Machine Learning Repository** is used.

The dataset chosen for this project contains observations about a credit card company's customer payment defaults in Taiwan. From the perspective of risk management, the result of predictive accuracy of the estimated probability of default will be more valuable than the binary result of classification - credible or not credible clients. The data is an Excel sheet with 30,000 observations on the following 24 variables:

- **Personal data**
  - **limit_bal:** Amount of the given credit (in NT$)
  - **sex:** Gender (1 = male; 2 = female)
  - **education:** Education (1 = graduate school; 2 = university; 3 = high school; 4 = others)
  - **marriage:** Martial status (1 = married; 2 = single; 3 = others)
  - **age** Age (years)
- **Past monthly repayments** (-1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . . ; 8 = payment delay for eight months; 9 = payment delay for nine months and above)
  - **pay_0:** September
  - **pay_2:** August
  - **pay_3:** July
  - **pay_4:** June
  - **pay_5:** May
  - **pay_6:** April
- **Amount of bill statement** (in NT$)
  - **bill_amt1:** September
  - **bill_amt2:** August
  - **bill_amt3:** July
  - **bill_amt4:** June
  - **bill_amt5:** May
  - **bill_amt6:** April
- **Amount of previous payment** (in NT$)
  - **pay_amt1:** September
  - **pay_amt2:** August
  - **pay_amt3:** July
  - **pay_amt4:** June
  - **pay_amt5:** May
  - **pay_amt6:** April
- **default_payment_next_month** (our target variable: is it a payment default or not)

2 Methods and Analysis

The particular steps are documented in detail by the comments next to the corresponding code. First we introduce the system environment used for this project, then we introduce and load the necessary libraries. After that we proceed with the data wrangling in order to analyze our predictor variables and their respective relation to the target variable. Next we train eight different models and compare them in respect of their predicting performance.

2.1 System Environment

To ease the reproducability of this project, we take a look at the (technical) infrastructure used for this project:

```r
# Show software versions, parameters and packages
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-pc-linux-gnu (64-bit)
```

```
## Running under: Ubuntu 18.10
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.8.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.8.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=de_AT.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=de_AT.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] benchmarkme_1.0.0 doParallel_1.0.14 iterators_1.0.10  foreach_1.4.4
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.1            benchmarkmeData_1.0.1 knitr_1.23
##  [4] magrittr_1.5          tidyselect_0.2.5      lattice_0.20-35
##  [7] R6_2.4.0              rlang_0.3.4           httr_1.4.0
## [10] stringr_1.4.0         dplyr_0.8.1           tools_3.5.1
## [13] grid_3.5.1            xfun_0.7              htmltools_0.3.6
## [16] yaml_2.2.0            digest_0.6.19         assertthat_0.2.1
## [19] tibble_2.1.3          crayon_1.3.4          Matrix_1.2-14
## [22] purrr_0.3.2           codetools_0.2-15      glue_1.3.1
## [25] evaluate_0.14         rmarkdown_1.13        stringi_1.4.3
## [28] compiler_3.5.1        pillar_1.4.1          pkgconfig_2.0.2
```

```r
# Show CPU and memory
cbind(as.data.frame(get_cpu()),
      memory = paste0(as.character(round(get_ram()/1024^3)),"GB"))
```

```
##      vendor_id                                model_name no_of_cores
## 1 GenuineIntel Intel(R) Core(TM) i7-4810MQ CPU @ 2.80GHz           8
##   memory
## 1   31GB
```

2.2 Load Libraries

Throughout this project, more than just basic R functionality is needed. Thus we load and use additional packages.

```r
library(tidyverse) # For using R the tidy(verse) way
library(readxl) # for reading excel files
library(janitor) #For cleaning variable names
library(caret) # For machine learning tasks
library(lubridate) # For time processing
library(caretEnsemble) # For making model ensembles
library(ranger) # For random forest models
```

```r
library(glmnet) # For glm network models
library(naivebayes) # For naivebayes models
library(C50) # For C50 models
library(DMwR) # For SMOTE sampling
```

2.3 Data Preparation

Our starting point is the creation of a new folder, in which the dataset will be downloaded into:

```r
# We create a folder for the dataset...
dir.create("data/")

# ...and we download it
download.file("https://tinyurl.com/y2ewr9e7","data/CreditDataSet.xls")

# Then we read the dataset into a tibble with clean names, correctly format submitted_date
# as date object and only used data from 2017 and 2018
dataset <- read_excel(path = "data/CreditDataSet.xls", col_names = TRUE, trim_ws = TRUE,
                      skip = 1) %>% clean_names() %>% select(-id) %>% as.data.frame()

# Let's see what variables and character types we have so far
dataset %>% glimpse()
```

```
## Observations: 30,000
## Variables: 24
## $ limit_bal                  <dbl> 20000, 120000, 90000, 50000, 50000,...
## $ sex                        <dbl> 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 2, 2,...
## $ education                  <dbl> 2, 2, 2, 2, 2, 1, 1, 2, 3, 3, 3, 1,...
## $ marriage                   <dbl> 1, 2, 2, 1, 1, 2, 2, 2, 1, 2, 2, 2,...
## $ age                        <dbl> 24, 26, 34, 37, 57, 37, 29, 23, 28,...
## $ pay_0                      <dbl> 2, -1, 0, 0, -1, 0, 0, 0, 0, -2, 0,...
## $ pay_2                      <dbl> 2, 2, 0, 0, 0, 0, 0, -1, 0, -2, 0, ...
## $ pay_3                      <dbl> -1, 0, 0, 0, -1, 0, 0, -1, 2, -2, 2...
## $ pay_4                      <dbl> -1, 0, 0, 0, 0, 0, 0, 0, 0, -2, 0, ...
## $ pay_5                      <dbl> -2, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, ...
## $ pay_6                      <dbl> -2, 2, 0, 0, 0, 0, 0, -1, 0, -1, -1...
## $ bill_amt1                  <dbl> 3913, 2682, 29239, 46990, 8617, 644...
## $ bill_amt2                  <dbl> 3102, 1725, 14027, 48233, 5670, 570...
## $ bill_amt3                  <dbl> 689, 2682, 13559, 49291, 35835, 576...
## $ bill_amt4                  <dbl> 0, 3272, 14331, 28314, 20940, 19394...
## $ bill_amt5                  <dbl> 0, 3455, 14948, 28959, 19146, 19619...
## $ bill_amt6                  <dbl> 0, 3261, 15549, 29547, 19131, 20024...
## $ pay_amt1                   <dbl> 0, 0, 1518, 2000, 2000, 2500, 55000...
## $ pay_amt2                   <dbl> 689, 1000, 1500, 2019, 36681, 1815,...
## $ pay_amt3                   <dbl> 0, 1000, 1000, 1200, 10000, 657, 38...
## $ pay_amt4                   <dbl> 0, 1000, 1000, 1100, 9000, 1000, 20...
## $ pay_amt5                   <dbl> 0, 0, 1000, 1069, 689, 1000, 13750,...
## $ pay_amt6                   <dbl> 0, 2000, 5000, 1000, 679, 800, 1377...
## $ default_payment_next_month <dbl> 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
```

```r
# Let's also see if there are missing values within our dataset
# Our dataset seems to have no missing values
dataset %>% anyNA()
```

```
## [1] FALSE
```

Now we have insight which variables, how many observations, which data types our dataset contains. We also already saw that there are some values missing. One by one, we will now tidy the data sets' variables for our exploratory data analysis.

```
############### Variable 1: "limit_bal" ###############

# We take a look at the distribution
dataset$limit_bal %>% summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   10000   50000  140000  167484  240000 1000000
```

```
# Double is a suitable type, so we have nothing to do
dataset$limit_bal %>% typeof()
```

```
## [1] "double"
```

```
############### Variable 2: "sex" ###############

# Are there any strange values? No
dataset$sex %>% unique()
```

```
## [1] 2 1
```

```
# Sex is currently integer, but we want it to be a factor...
dataset$sex %>% typeof()
```

```
## [1] "double"
```

```
# ...so we convert sex
dataset <- dataset %>% mutate(sex = as_factor(ifelse(sex == 1, "male", "female")))

############### Variable 3: "education" ###############

# Are there any strange values? Yes, the values 5, 6 and 0 are not correct...
dataset$education %>% unique()
```

```
## [1] 2 1 3 5 4 6 0
```

```
# ...so we get rid of them (345 observations)
dataset <- dataset %>% filter(education > 0) %>% filter(education < 5)

# Education is currently double, but we want it to be a factor...
dataset$education %>% typeof()
```

```
## [1] "double"
```

```
# ...so we convert education
dataset <- dataset %>% mutate(education = as_factor(case_when(
  education == 1 ~ "graduate.school",
  education == 2 ~ "university",
  education == 3 ~ "high.school",
  TRUE ~ "others")))

############### Variable 4: "marriage" ###############

# Are there any strange values? Yes, the value 0 is not correct...
dataset$marriage %>% unique()
```

```
## [1] 1 2 3 0
```

```r
# ...so we get rid of them (54 observations)
dataset <- dataset %>% filter(marriage != 0)

# Education is currently double, but we want it to be a factor...
dataset$marriage %>% typeof()
```

```
## [1] "double"
```

```r
# ...so we convert education
dataset <- dataset %>% mutate(marriage = as_factor(case_when(
  marriage == 1 ~ "married",
  marriage == 2 ~ "single",
  marriage == 3 ~ "others",
  TRUE ~ "others")))

############### Variable 5: "age" ###############

# We take a look at the distribution
dataset$age %>% summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   21.00   28.00   34.00   35.46   41.00   79.00
```

```r
# Double is a suitable type, so we have nothing to do
dataset$age %>% typeof()
```

```
## [1] "double"
```

```r
############### Variable 6 - 11: "past monthly repayments" ###############

# We want the variables renamed...
dataset$pay_0 %>% typeof()
```

```
## [1] "double"
```

```r
# ...so we rename them to have more self-explaining names
dataset <- dataset %>% rename(pay_delay_sep = pay_0, pay_delay_aug = pay_2,
                              pay_delay_jul = pay_3,pay_delay_jun = pay_4,
                              pay_delay_may = pay_5, pay_delay_apr = pay_6)

############### Variable 12 - 17: "Amount of bill statement" ###############

# We want the variables renamed...
dataset$bill_amt1 %>% typeof()
```

```
## [1] "double"
```

```r
# ...so we rename them to have more self-explaining names
dataset <-dataset %>% rename(bill_sep = bill_amt1, bill_aug = bill_amt2,
                             bill_jul = bill_amt3, bill_jun = bill_amt4,
                             bill_may = bill_amt5, bill_apr = bill_amt6)

############### Variable 18 - 23: "Amount of previous payment" ###############

# We want the variables renamed...
dataset$pay_amt1 %>% typeof()
```

```
## [1] "double"
```

```r
# ...so we rename them to have more self-explaining names
#dataset <-
dataset <- dataset %>% rename(prev_payed_sep = pay_amt1, prev_payed_aug = pay_amt2,
                              prev_payed_jul = pay_amt3, prev_payed_jun = pay_amt4,
                              prev_payed_may = pay_amt5, prev_payed_apr = pay_amt6)

############### Variable 24: "Default payment next month" ###############

# default_payment_next_month is currently double, but we want it to be a factor...
dataset$default_payment_next_month %>% typeof()
```

```
## [1] "double"
```

```r
# ...so we convert it
dataset <- dataset %>% rename(payment_default = default_payment_next_month) %>%
  mutate(payment_default = as_factor(ifelse(payment_default == 1, "default", "non_default")))
```

2.4 Data Statistics

After cleaning and tidying our data we now have a look at the overall dataset statistics.

```r
# We take a look at the dataset structure
dataset %>% str()
```

```
## 'data.frame':    29601 obs. of  24 variables:
##  $ limit_bal      : num  20000 120000 90000 50000 50000 50000 500000 100000 140000 20000 ...
##  $ sex            : Factor w/ 2 levels "female","male": 1 1 1 1 2 2 2 1 1 2 ...
##  $ education      : Factor w/ 4 levels "university","graduate.school",..: 1 1 1 1 1 2 2 1 3 3 ...
##  $ marriage       : Factor w/ 3 levels "married","single",..: 1 2 2 1 1 2 2 2 1 2 ...
##  $ age            : num  24 26 34 37 57 37 29 23 28 35 ...
##  $ pay_delay_sep  : num  2 -1 0 0 -1 0 0 0 0 -2 ...
##  $ pay_delay_aug  : num  2 2 0 0 0 0 0 -1 0 -2 ...
##  $ pay_delay_jul  : num  -1 0 0 0 -1 0 0 -1 2 -2 ...
##  $ pay_delay_jun  : num  -1 0 0 0 0 0 0 0 0 -2 ...
##  $ pay_delay_may  : num  -2 0 0 0 0 0 0 0 0 -1 ...
##  $ pay_delay_apr  : num  -2 2 0 0 0 0 0 -1 0 -1 ...
##  $ bill_sep       : num  3913 2682 29239 46990 8617 ...
##  $ bill_aug       : num  3102 1725 14027 48233 5670 ...
##  $ bill_jul       : num  689 2682 13559 49291 35835 ...
##  $ bill_jun       : num  0 3272 14331 28314 20940 ...
##  $ bill_may       : num  0 3455 14948 28959 19146 ...
##  $ bill_apr       : num  0 3261 15549 29547 19131 ...
##  $ prev_payed_sep : num  0 0 1518 2000 2000 ...
##  $ prev_payed_aug : num  689 1000 1500 2019 36681 ...
##  $ prev_payed_jul : num  0 1000 1000 1200 10000 657 38000 0 432 0 ...
##  $ prev_payed_jun : num  0 1000 1000 1100 9000 ...
##  $ prev_payed_may : num  0 0 1000 1069 689 ...
##  $ prev_payed_apr : num  0 2000 5000 1000 679 ...
##  $ payment_default: Factor w/ 2 levels "default","non_default": 1 1 2 2 2 2 2 2 2 2 ...
```

```r
# We take a look at the statistics of payment defaults
dataset %>% filter(payment_default == "default") %>% summary()
```

```
##    limit_bal           sex                  education        marriage
```

```
##  Min.   : 10000   female:3744   university     :3329   married:3192
##  1st Qu.: 50000   male  :2861   graduate.school:2036   single :3329
##  Median : 90000                 high.school    :1233   others :  84
##  Mean   :130125                 others         :   7
##  3rd Qu.:200000
##  Max.   :740000
##       age         pay_delay_sep     pay_delay_aug     pay_delay_jul
##  Min.   :21.00   Min.   :-2.0000   Min.   :-2.0000   Min.   :-2.0000
##  1st Qu.:28.00   1st Qu.: 0.0000   1st Qu.:-1.0000   1st Qu.:-1.0000
##  Median :34.00   Median : 1.0000   Median : 0.0000   Median : 0.0000
##  Mean   :35.71   Mean   : 0.6698   Mean   : 0.4598   Mean   : 0.3634
##  3rd Qu.:42.00   3rd Qu.: 2.0000   3rd Qu.: 2.0000   3rd Qu.: 2.0000
##  Max.   :75.00   Max.   : 8.0000   Max.   : 7.0000   Max.   : 8.0000
##  pay_delay_jun     pay_delay_may     pay_delay_apr        bill_sep
##  Min.   :-2.0000   Min.   :-2.0000   Min.   :-2.0000   Min.   : -6676
##  1st Qu.:-1.0000   1st Qu.:-1.0000   1st Qu.:-1.0000   1st Qu.:  2948
##  Median : 0.0000   Median : 0.0000   Median : 0.0000   Median : 20039
##  Mean   : 0.2551   Mean   : 0.1699   Mean   : 0.1149   Mean   : 48315
##  3rd Qu.: 2.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 59121
##  Max.   : 8.0000   Max.   : 8.0000   Max.   : 8.0000   Max.   :613860
##     bill_aug          bill_jul          bill_jun          bill_may
##  Min.   :-17710   Min.   :-61506   Min.   :-65167   Min.   :-53007
##  1st Qu.:  2650   1st Qu.:  2500   1st Qu.:  2140   1st Qu.:  1509
##  Median : 20237   Median : 19783   Median : 19104   Median : 18462
##  Mean   : 47128   Mean   : 45063   Mean   : 41988   Mean   : 39530
##  3rd Qu.: 57550   3rd Qu.: 54293   3rd Qu.: 50126   3rd Qu.: 47835
##  Max.   :581775   Max.   :578971   Max.   :548020   Max.   :547880
##     bill_apr         prev_payed_sep   prev_payed_aug   prev_payed_jul
##  Min.   :-339603   Min.   :     0   Min.   :     0   Min.   :     0
##  1st Qu.:   1150   1st Qu.:     0   1st Qu.:     0   1st Qu.:     0
##  Median :  18023   Median :  1611   Median :  1529   Median :  1219
##  Mean   :  38266   Mean   :  3366   Mean   :  3383   Mean   :  3352
##  3rd Qu.:  47418   3rd Qu.:  3465   3rd Qu.:  3300   3rd Qu.:  3000
##  Max.   : 514975   Max.   :300000   Max.   :358689   Max.   :508229
##  prev_payed_jun   prev_payed_may   prev_payed_apr      payment_default
##  Min.   :     0   Min.   :     0   Min.   :     0   default    :6605
##  1st Qu.:     0   1st Qu.:     0   1st Qu.:     0   non_default:   0
##  Median :  1000   Median :  1000   Median :  1000
##  Mean   :  3157   Mean   :  3200   Mean   :  3412
##  3rd Qu.:  2938   3rd Qu.:  3000   3rd Qu.:  2971
##  Max.   :432130   Max.   :332000   Max.   :345293
```

```r
# We also take a look at the statistics of the non payment defaults
dataset %>% filter(payment_default == "non_default") %>% summary()
```

```
##    limit_bal            sex                  education         marriage
##  Min.   :  10000   female:14111   university     :10695   married:10285
##  1st Qu.:  70000   male  : 8885   graduate.school: 8545   single :12477
##  Median : 150000                  high.school    : 3640   others :  234
##  Mean   : 178300                  others         :  116
##  3rd Qu.: 250000
##  Max.   :1000000
##       age         pay_delay_sep     pay_delay_aug     pay_delay_jul
##  Min.   :21.00   Min.   :-2.0000   Min.   :-2.0000   Min.   :-2.0000
##  1st Qu.:28.00   1st Qu.:-1.0000   1st Qu.:-1.0000   1st Qu.:-1.0000
```

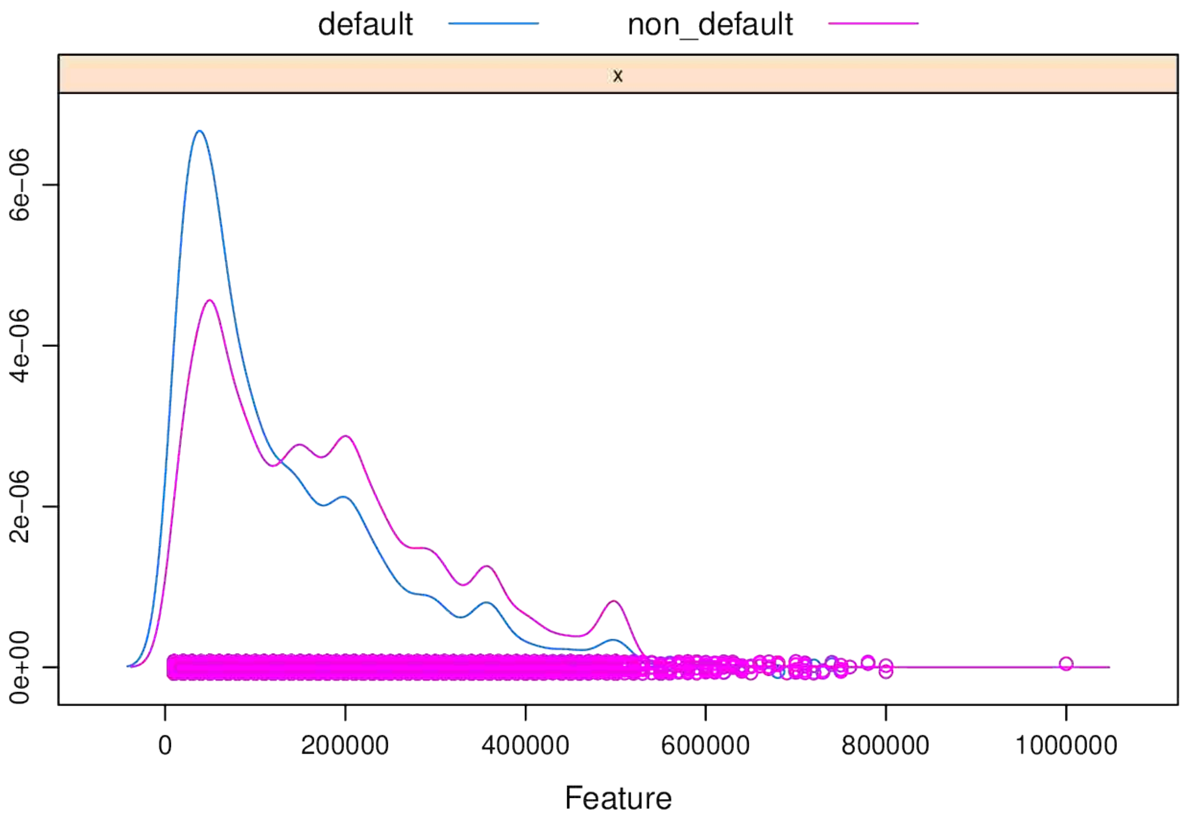
```
##  Median :34.00   Median : 0.0000   Median : 0.0000   Median : 0.0000
##  Mean   :35.39   Mean   :-0.2116   Mean   :-0.3011   Mean   :-0.3147
##  3rd Qu.:41.00   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000
##  Max.   :79.00   Max.   : 8.0000   Max.   : 8.0000   Max.   : 8.0000
##  pay_delay_jun     pay_delay_may     pay_delay_apr        bill_sep
##  Min.   :-2.0000   Min.   :-2.0000   Min.   :-2.0000   Min.   :-165580
##  1st Qu.:-1.0000   1st Qu.:-1.0000   1st Qu.:-1.0000   1st Qu.:   3662
##  Median : 0.0000   Median : 0.0000   Median : 0.0000   Median :  22948
##  Mean   :-0.3543   Mean   :-0.3886   Mean   :-0.4032   Mean   :  51716
##  3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.:  68590
##  Max.   : 8.0000   Max.   : 7.0000   Max.   : 7.0000   Max.   : 964511
##     bill_aug         bill_jul          bill_jun          bill_may
##  Min.   :-69777   Min.   :-157264   Min.   :-170000   Min.   :-81334
##  1st Qu.:  3037   1st Qu.:   2754   1st Qu.:   2362   1st Qu.:  1851
##  Median : 21502   Median :  20148   Median :  18966   Median : 17975
##  Mean   : 49463   Mean   :  47303   Mean   :  43448   Mean   : 40438
##  3rd Qu.: 65235   3rd Qu.:  61444   3rd Qu.:  55716   3rd Qu.: 51015
##  Max.   :983931   Max.   :1664089   Max.   : 891586   Max.   :927171
##     bill_apr        prev_payed_sep    prev_payed_aug    prev_payed_jul
##  Min.   :-209051   Min.   :     0   Min.   :      0   Min.   :     0
##  1st Qu.:   1300   1st Qu.:  1161   1st Qu.:   1005   1st Qu.:   600
##  Median :  16736   Median :  2450   Median :   2236   Median :  2000
##  Mean   :  39029   Mean   :  6306   Mean   :   6616   Mean   :  5729
##  3rd Qu.:  49818   3rd Qu.:  5600   3rd Qu.:   5300   3rd Qu.:  5000
##  Max.   : 961664   Max.   :873552   Max.   :1684259   Max.   :896040
##  prev_payed_jun    prev_payed_may    prev_payed_apr        payment_default
##  Min.   :     0   Min.   :     0   Min.   :     0.0   default    :    0
##  1st Qu.:   390   1st Qu.:   378   1st Qu.:   313.8   non_default:22996
##  Median :  1734   Median :  1770   Median :  1709.0
##  Mean   :  5309   Mean   :  5253   Mean   :  5689.4
##  3rd Qu.:  4627   3rd Qu.:  4632   3rd Qu.:  4555.8
##  Max.   :621000   Max.   :426529   Max.   :528666.0
```

```r
# We set parameters for our following plots: scales and strip
scales <- list(x=list(relation="free"), y=list(relation="free"))
log10scalex <- list(x = list(log = 10))
log10scaley <- list(y = list(log = 10))
strip <- strip.custom(par.strip.text=list(cex=.7))
```

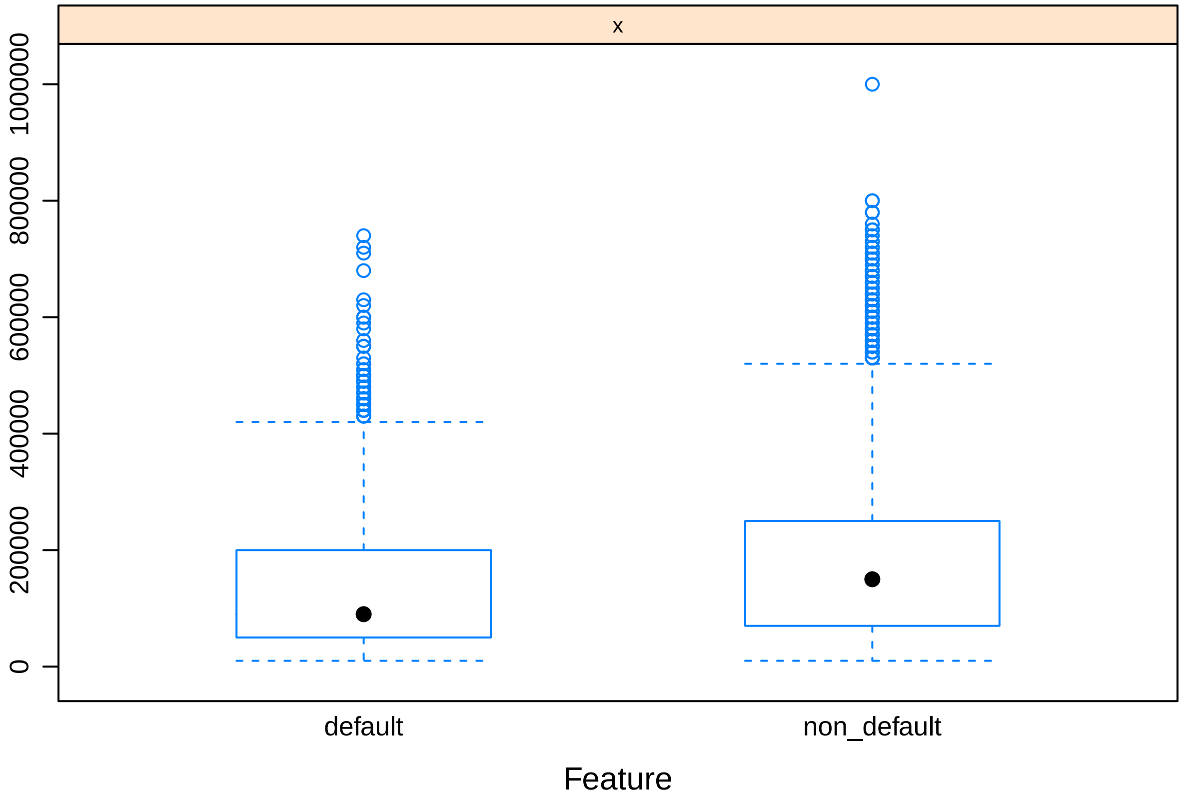
Our target variable consists of two imbalanced classes within "payment_default" with 6605 (~22% default) vs 22996 (~78% non-default) observations. Additionally, our potential predictors in the dataset look quite skewed. We will take care of the imbalance problem later with SMOTE upsampling.

2.4.1 Variable 1: limit_bal

```r
############### Variable 1: "limit_bal" ###############
featurePlot(x = dataset$limit_bal, y = dataset$payment_default, plot = "density",
            strip = strip, scales = scales, auto.key = list(columns = 2))
```




```
featurePlot(x = dataset$limit_bal, y = dataset$payment_default, plot = "box",
            strip = strip, scales = scales)
```



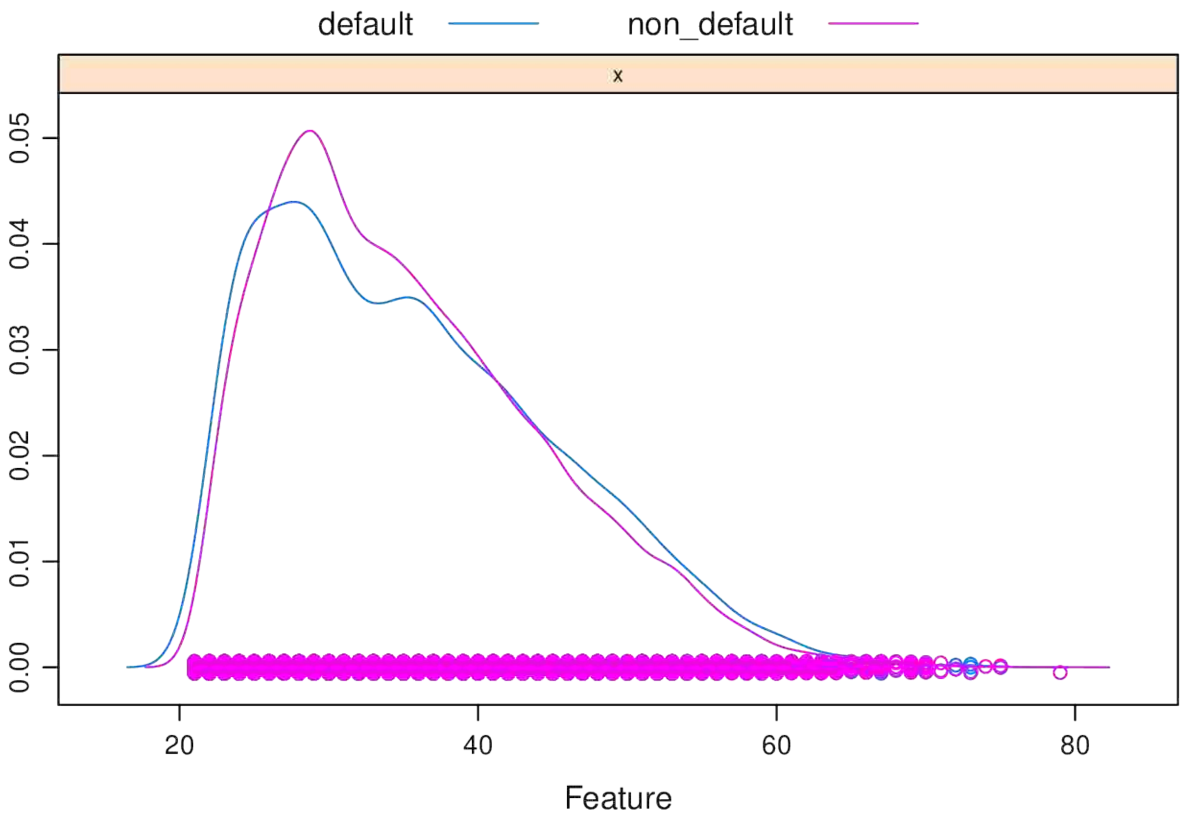

```
dataset$limit_bal %>% as.numeric() %>% summary()
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10000   50000  140000  167550  240000 1000000
```

We can see that defaults happen more often if the limit balance is at around 50,000. We also see that customers with payment defaults tendencially have a lower limit balance (probably because they were rated worse beforehand).

2.4.2 Variable 2: sex

```
############### Variable 2: "sex" ###############
featurePlot(x = as.numeric(dataset$sex), y = dataset$payment_default, plot = "density",
            strip = strip, scales = scales, auto.key = list(columns = 2))
```



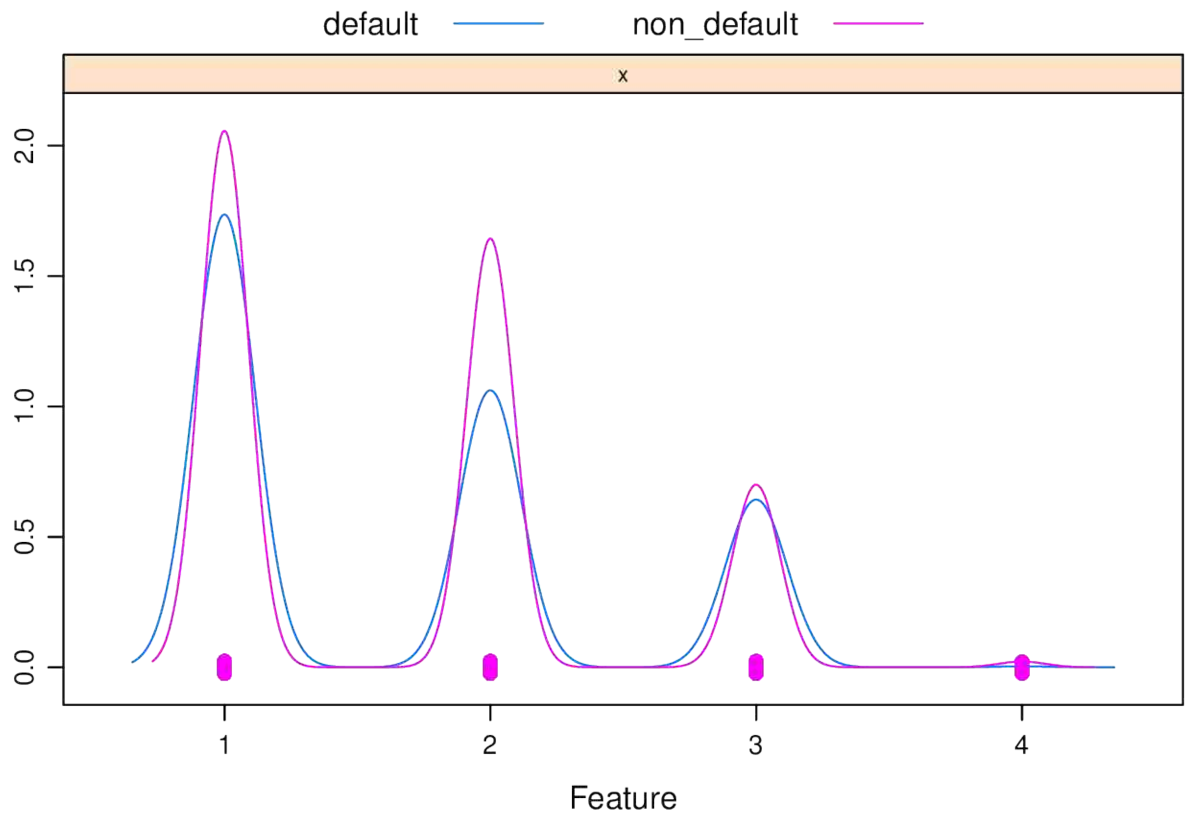

```r
dataset$sex %>% summary()
```

```
## female   male
##  17855  11746
```

We assume that females (1) tend to have a higher occurence of both, “default” as well as “non-default”. Males (2) show a lower overall occurence of being “default” or “non-default”, but both outcomes approximate to each other. But this might also be a bit misleading, since we have 17855 females and 11746 males in our dataset, a 60/40 splilt.

2.4.3 Variable 3: education

```r
############### Variable 3: "education" ###############
featurePlot(x = as.numeric(dataset$education), y = dataset$payment_default, plot = "density",
            strip = strip, scales = scales, auto.key = list(columns = 2))
```



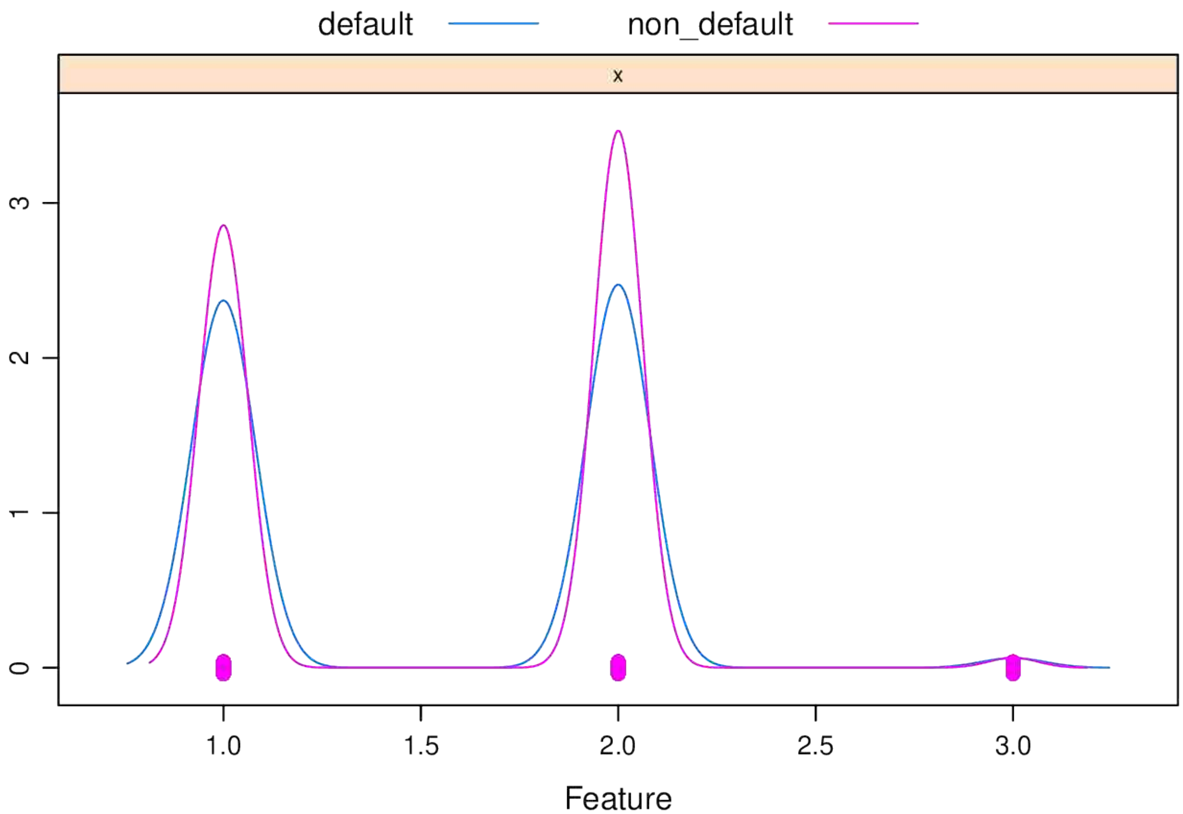

```
dataset$education %>% summary()
```

```
##      university graduate.school     high.school          others
##           14024           10581            4873             123
```

Education seems to have an impact on defaults and non-defaults. Persons that only finished graduate school (1) have the highest rate of defaults, which is significantly reduced if the person holds a high school (3) diploma. Interestingly university graduates (2) also have a lower default rate than graduate school graduates, but a higher default rate than high school graduates (3). What we can also learn from our dataset is that there university graduates (2) seem to be less of a default risk compared to the other groups. Since we don't know what exactly "others" includes we will not make any assumptions here.

2.4.4 Variable 4: marriage

```
############### Variable 4: "marriage" ###############
featurePlot(x = as.numeric(dataset$marriage), y = dataset$payment_default, plot = "density",
            strip = strip, scales = scales, auto.key = list(columns = 2))
```



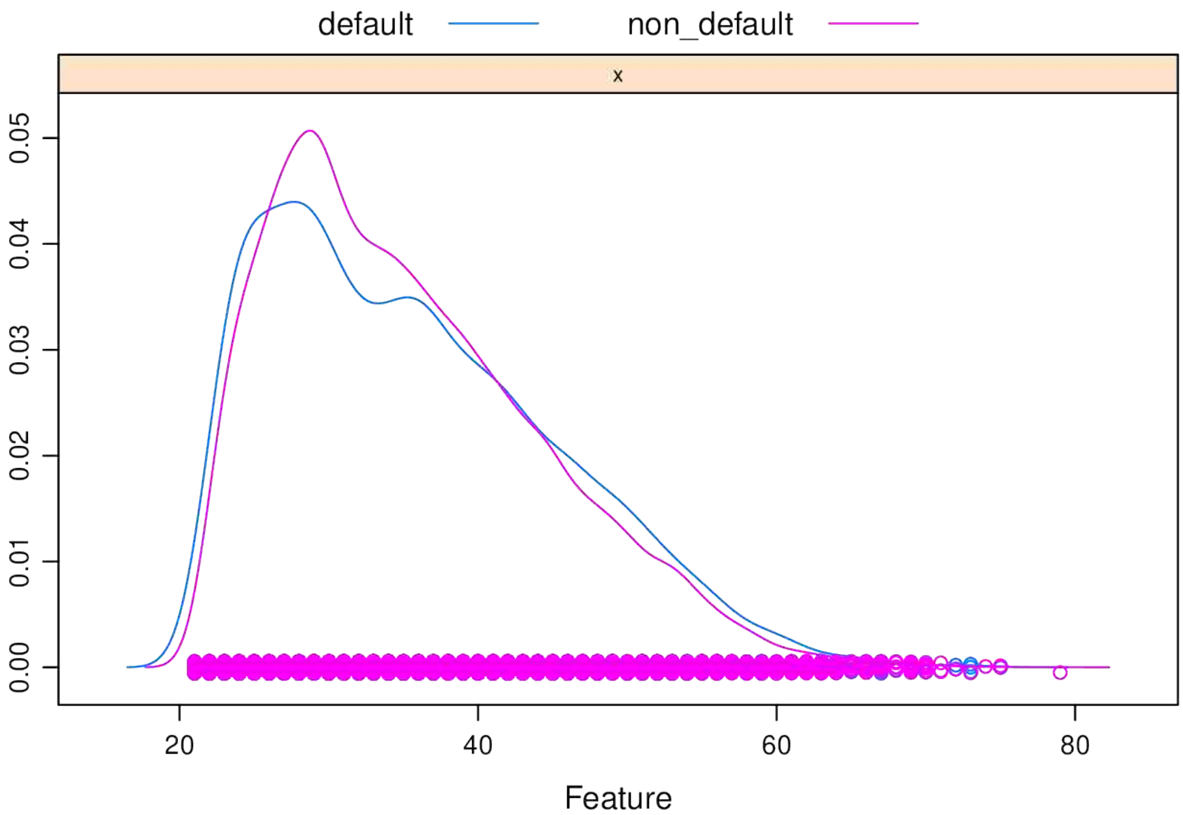

```
dataset$marriage %>% summary()
```

```
## married  single  others
##   13477   15806     318
```

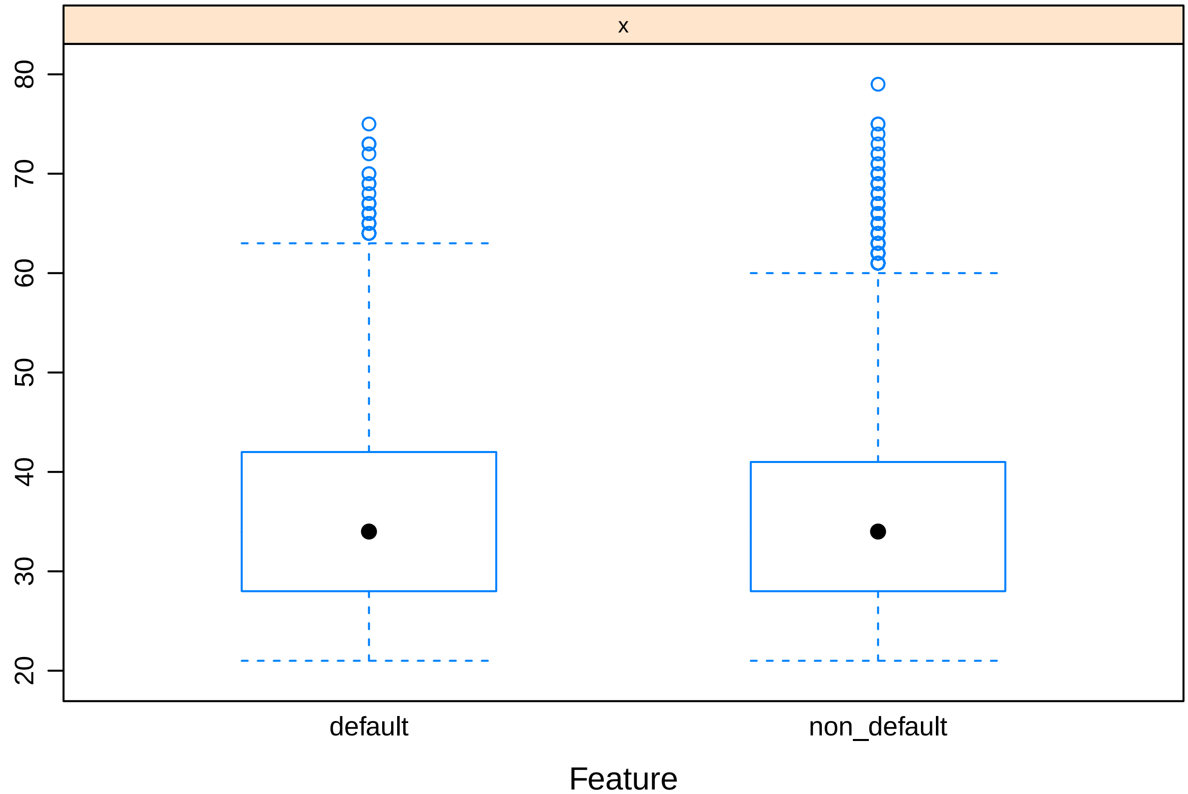
The data in our dataset shows that, nevertheless if being married (1) or not (2), defaults are approximately on the same level. In fact singles (2) seem to have a higher chance of being non-default. Again, we can hardly make any assumptions on "others" (3), so we will just leave it there. Around 45% are married, 54% are single and around 1% are others.

2.4.5 Variable 5: age

```
############### Variable 5: "age" ###############
featurePlot(x = dataset$age, y = dataset$payment_default, plot = "density", strip = strip,
            scales = scales, auto.key = list(columns = 2))
```




```r
featurePlot(x = dataset$age, y = dataset$payment_default, plot = "box", strip = strip,
            scales = scales)
```



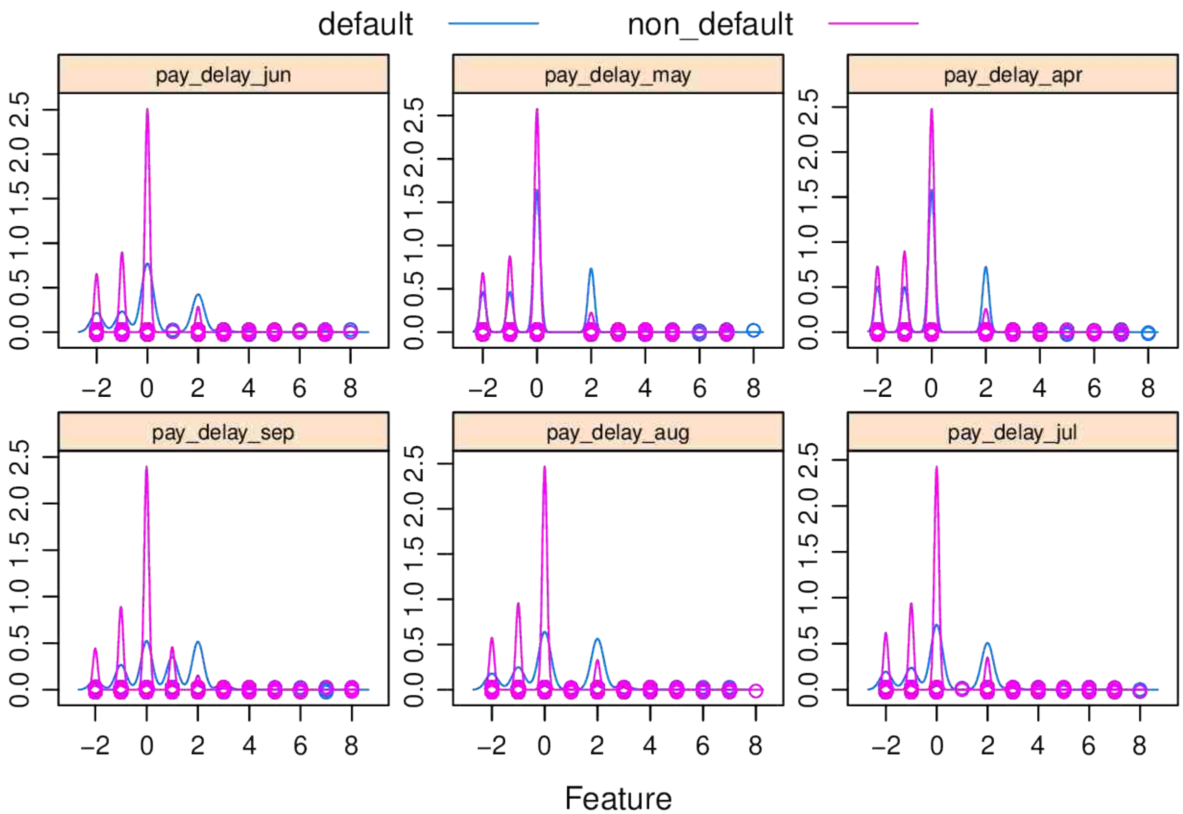

```
dataset$age %>% summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   21.00   28.00   34.00   35.46   41.00   79.00
```

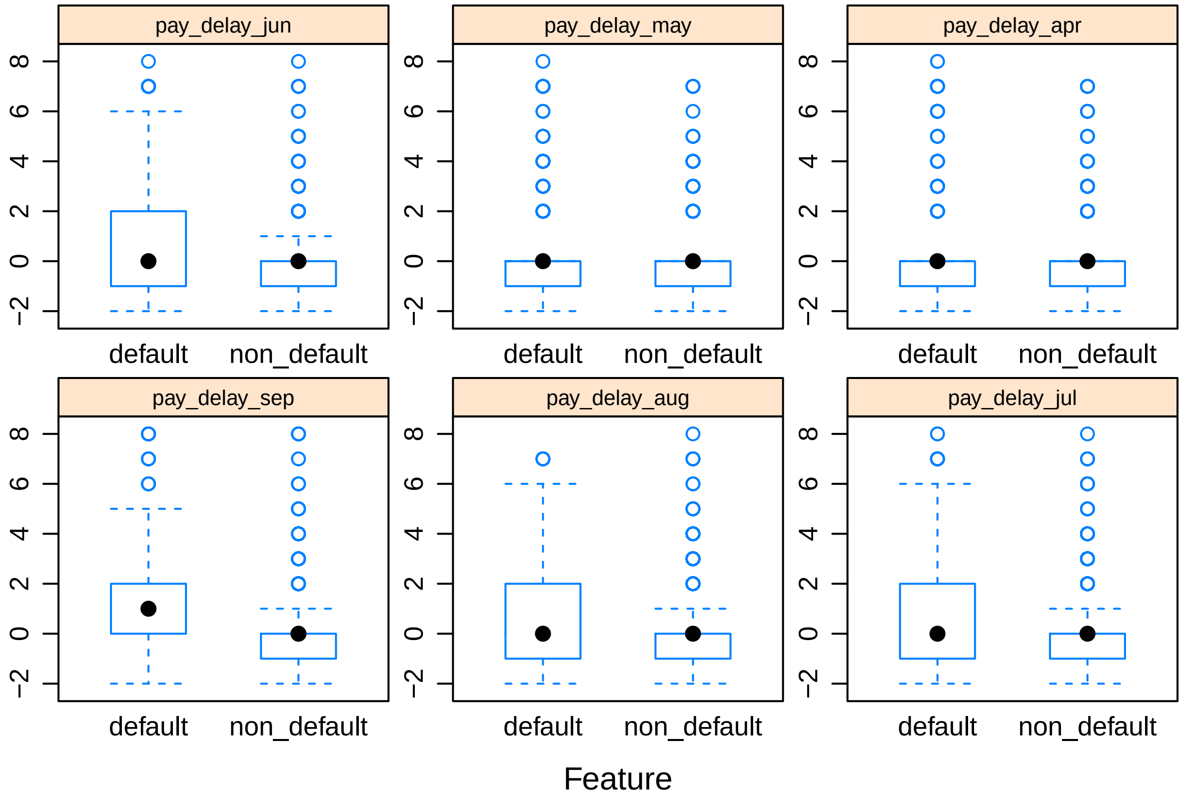
When looking at the age and (non-)default relationship we learn that defaults and non-defaults are approximately homogenous over all ages. The only significant exception is between around 30 and 40, where we can find a larger gap between non-defaults and defaults.

2.4.6 Variable 6-11: past monthly payments

```
############### Variable 6 - 11: "past monthly repayments" ###############
featurePlot(x = dataset[,6:11], y = dataset$payment_default, plot = "density", strip = strip,
            scales = scales, auto.key = list(columns = 2))
```




```r
featurePlot(x = dataset[,6:11], y = dataset$payment_default, plot = "box", strip = strip,
            scales = scales)
```



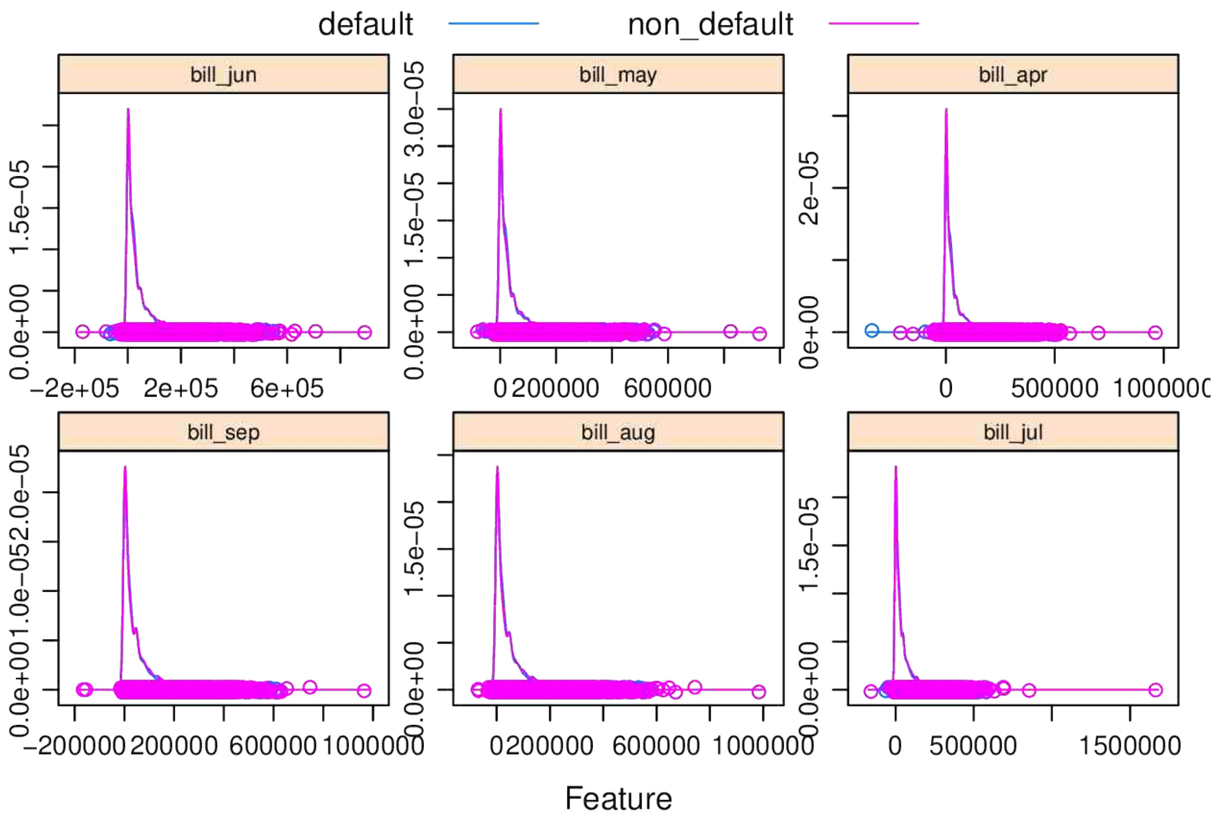

```
dataset[,6:11] %>% summary()
```

```
##  pay_delay_sep       pay_delay_aug      pay_delay_jul      pay_delay_jun
##  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   :-0.01493   Mean   :-0.1313   Mean   :-0.1634   Mean   :-0.2183
##  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
##  pay_delay_may     pay_delay_apr
##  Min.   :-2.000   Min.   :-2.0000
##  1st Qu.:-1.000   1st Qu.:-1.0000
##  Median : 0.000   Median : 0.0000
##  Mean   :-0.264   Mean   :-0.2876
##  3rd Qu.: 0.000   3rd Qu.: 0.0000
##  Max.   : 8.000   Max.   : 8.0000
```

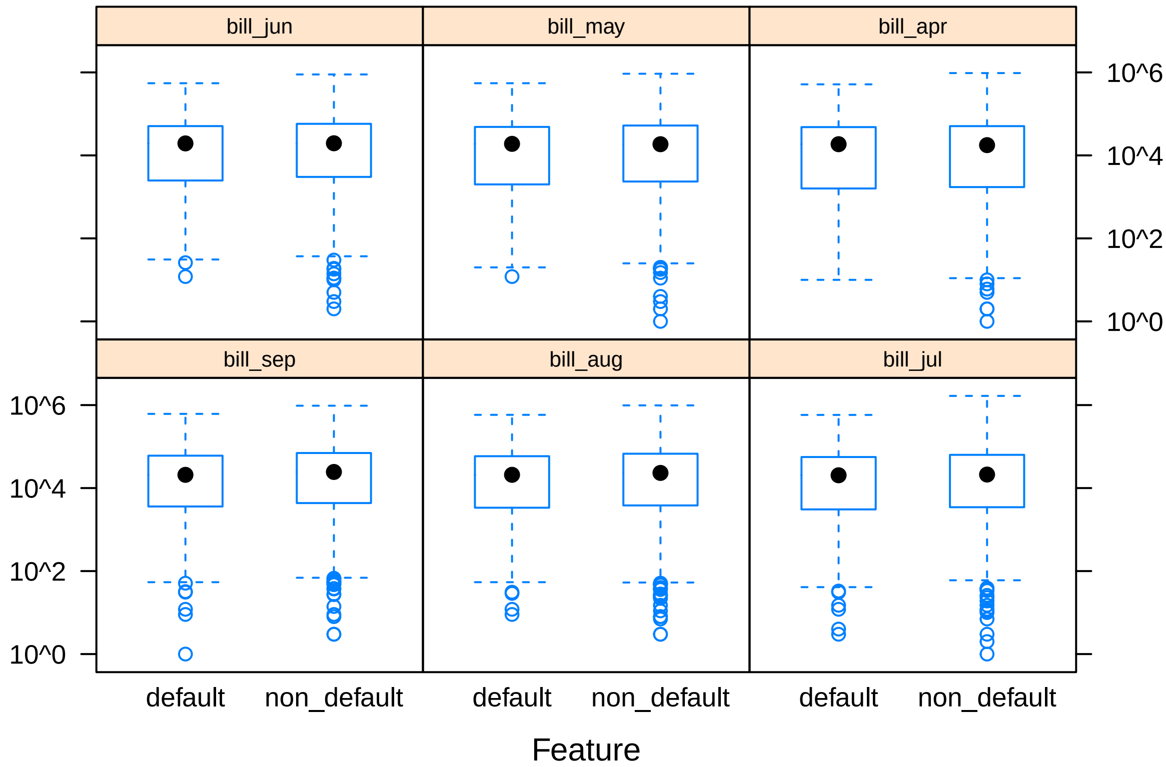
The patterns in all of the six months are approximately similar. We also do not find a normal distribution here (as one might have previously assumed).

2.4.7 Variable 12-17: bill statement amount

```
############### Variable 12 - 17: "Amount of bill statement" ###############
featurePlot(x = dataset[,12:17], y = dataset$payment_default, plot = "density",
            strip = strip, scales = scales, auto.key = list(columns = 2))
```




```r
featurePlot(x = dataset[,12:17], y = dataset$payment_default, plot = "box", strip = strip,
            scales = log10scaley)
```



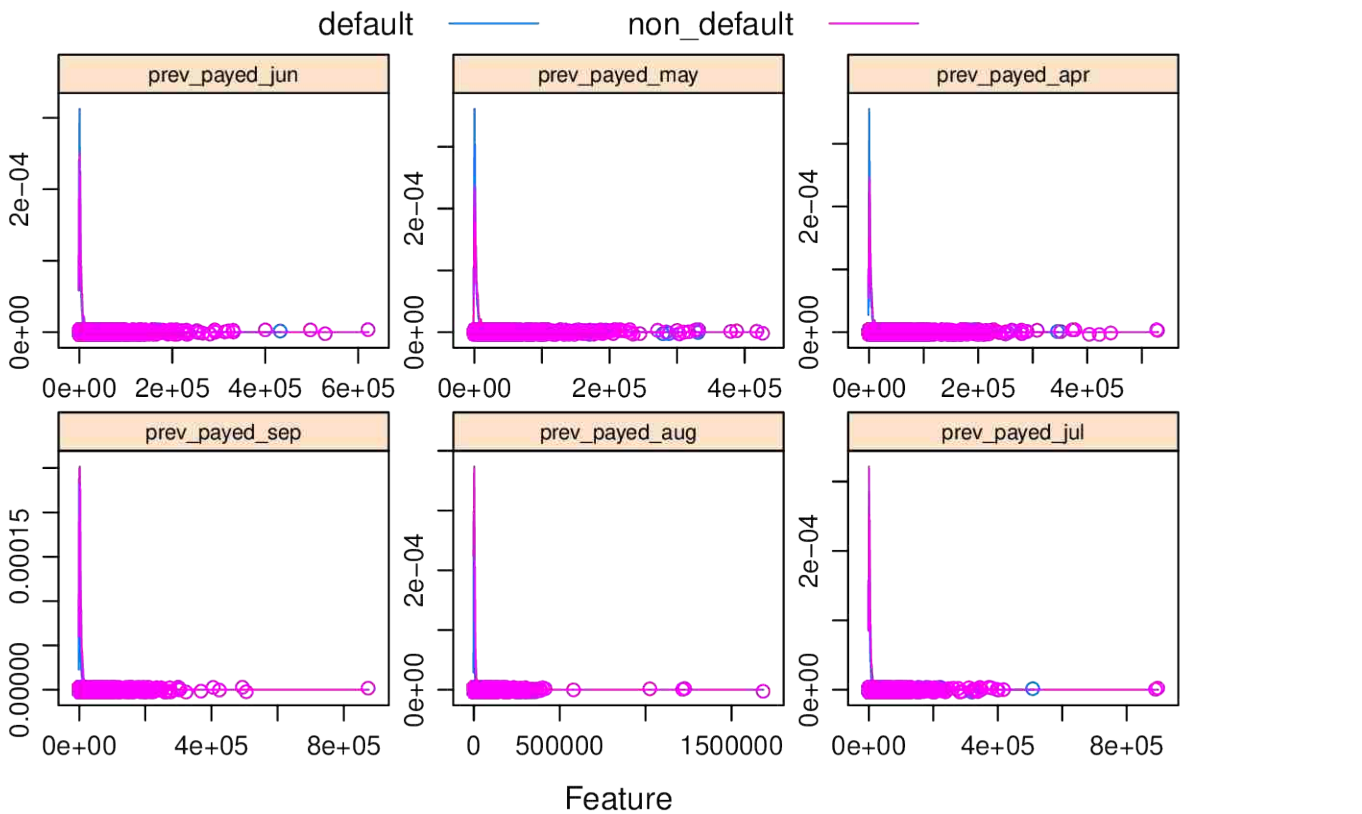

```
dataset[,12:17] %>% summary()
```

```
##     bill_sep          bill_aug          bill_jul          bill_jun     
##  Min.   :-165580   Min.   :-69777   Min.   :-157264   Min.   :-170000  
##  1st Qu.:   3528   1st Qu.:  2970   1st Qu.:   2652   1st Qu.:   2329  
##  Median :  22259   Median : 21050   Median :  20035   Median :  19005  
##  Mean   :  50957   Mean   : 48942   Mean   :  46803   Mean   :  43123  
##  3rd Qu.:  66623   3rd Qu.: 63497   3rd Qu.:  59830   3rd Qu.:  54271  
##  Max.   : 964511   Max.   :983931   Max.   :1664089   Max.   : 891586  
##     bill_may         bill_apr      
##  Min.   :-81334   Min.   :-339603  
##  1st Qu.:  1780   1st Qu.:   1278  
##  Median : 18091   Median :  17118  
##  Mean   : 40236   Mean   :  38858  
##  3rd Qu.: 50072   3rd Qu.:  49121  
##  Max.   :927171   Max.   : 961664
```

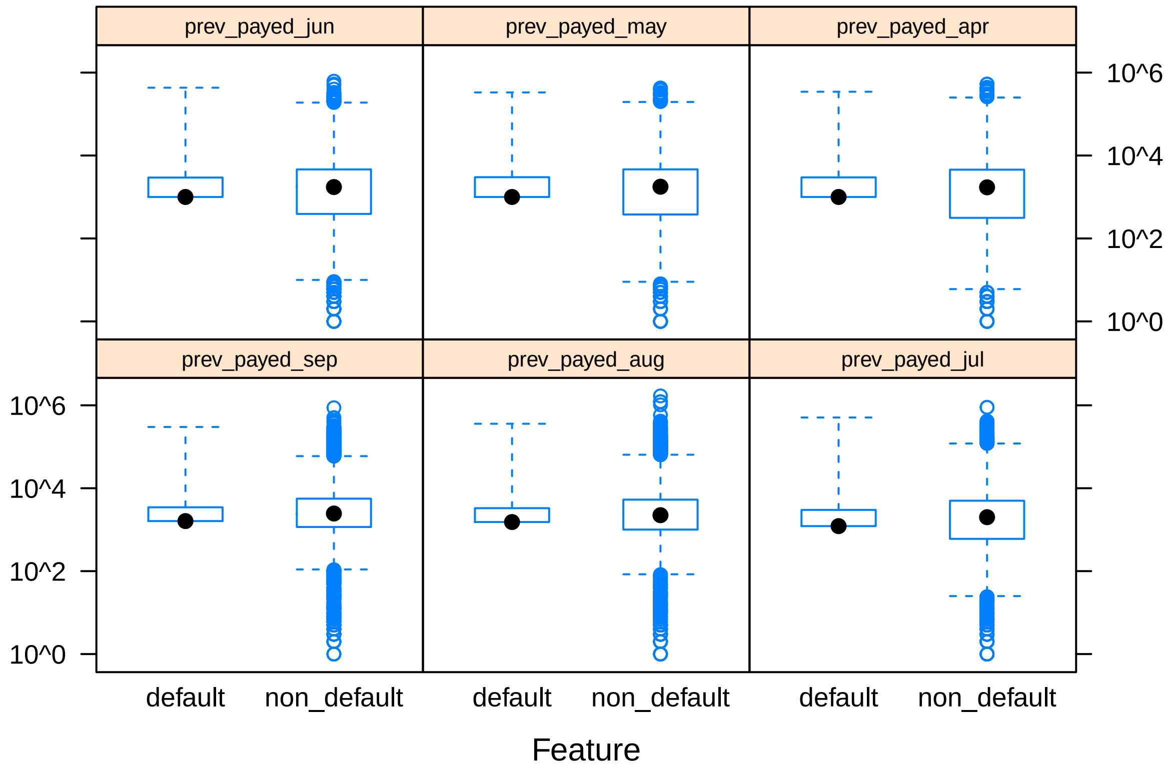
Bill amount does not provide that much new insights since defaults and non-defaults are approximately equal (except a few outliers).

2.4.8 Variable 18-23: previous payments

```
############### Variable 18 - 23: "Amount of previous payment" ###############
featurePlot(x = dataset[,18:23], y = dataset$payment_default, plot = "density",
            strip = strip, scales = scales, auto.key = list(columns = 2))
```




```r
featurePlot(x = dataset[,18:23], y = dataset$payment_default, plot = "box", strip = strip,
            scales = log10scaley)
```




```r
dataset[,18:23] %>% summary()
```

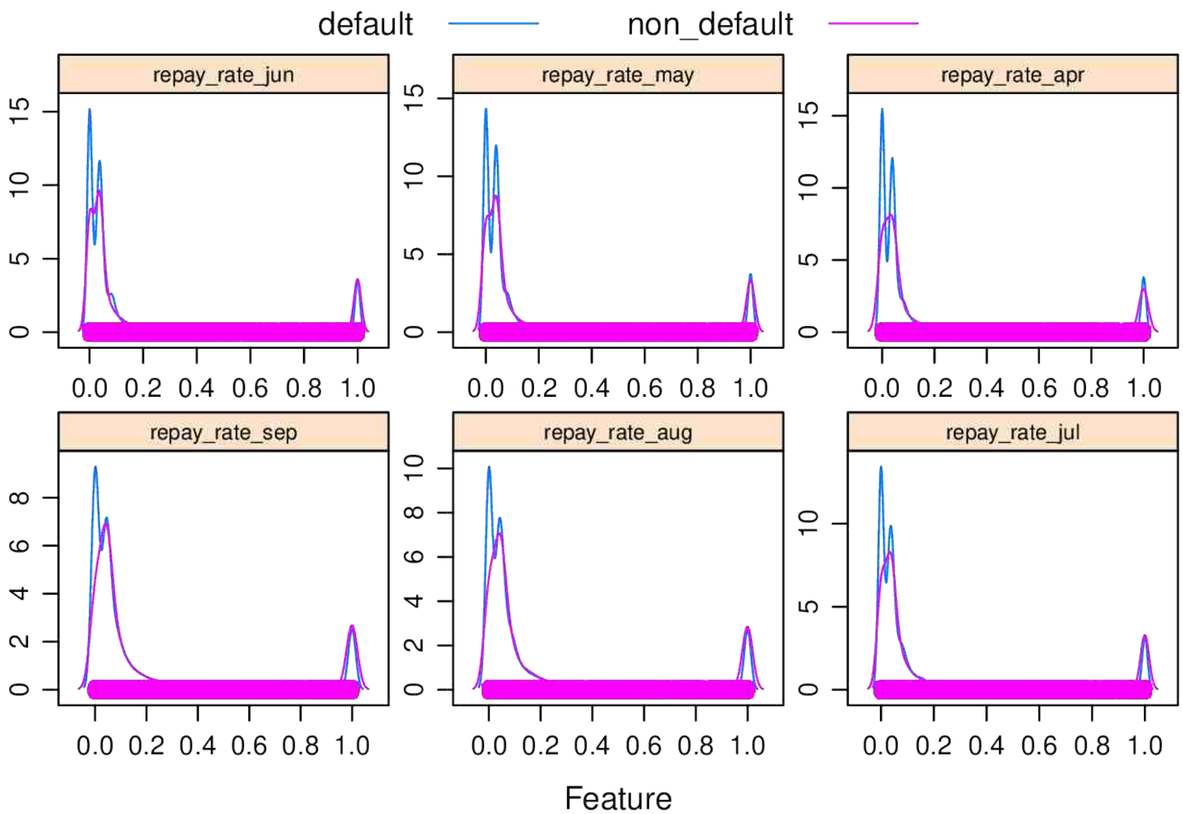
```
##  prev_payed_sep    prev_payed_aug     prev_payed_jul    prev_payed_jun
##  Min.   :     0   Min.   :      0   Min.   :     0   Min.   :     0
##  1st Qu.:  1000   1st Qu.:    825   1st Qu.:   390   1st Qu.:   298
##  Median :  2100   Median :   2007   Median :  1800   Median :  1500
##  Mean   :  5650   Mean   :   5895   Mean   :  5198   Mean   :  4829
##  3rd Qu.:  5005   3rd Qu.:   5000   3rd Qu.:  4500   3rd Qu.:  4014
##  Max.   :873552   Max.   :1684259   Max.   :896040   Max.   :621000
##  prev_payed_may   prev_payed_apr
##  Min.   :     0   Min.   :     0
##  1st Qu.:   259   1st Qu.:   138
##  Median :  1500   Median :  1500
##  Mean   :  4795   Mean   :  5181
##  3rd Qu.:  4042   3rd Qu.:  4000
##  Max.   :426529   Max.   :528666
```

Like bill amount, previously payed does not provide that much new insights either. Instead of those two variables it might be interesting to introduce a new variable that shows if the billed amount was fully, partially or not paid (to represent the payment habit instead of a variety of financial figures without context).

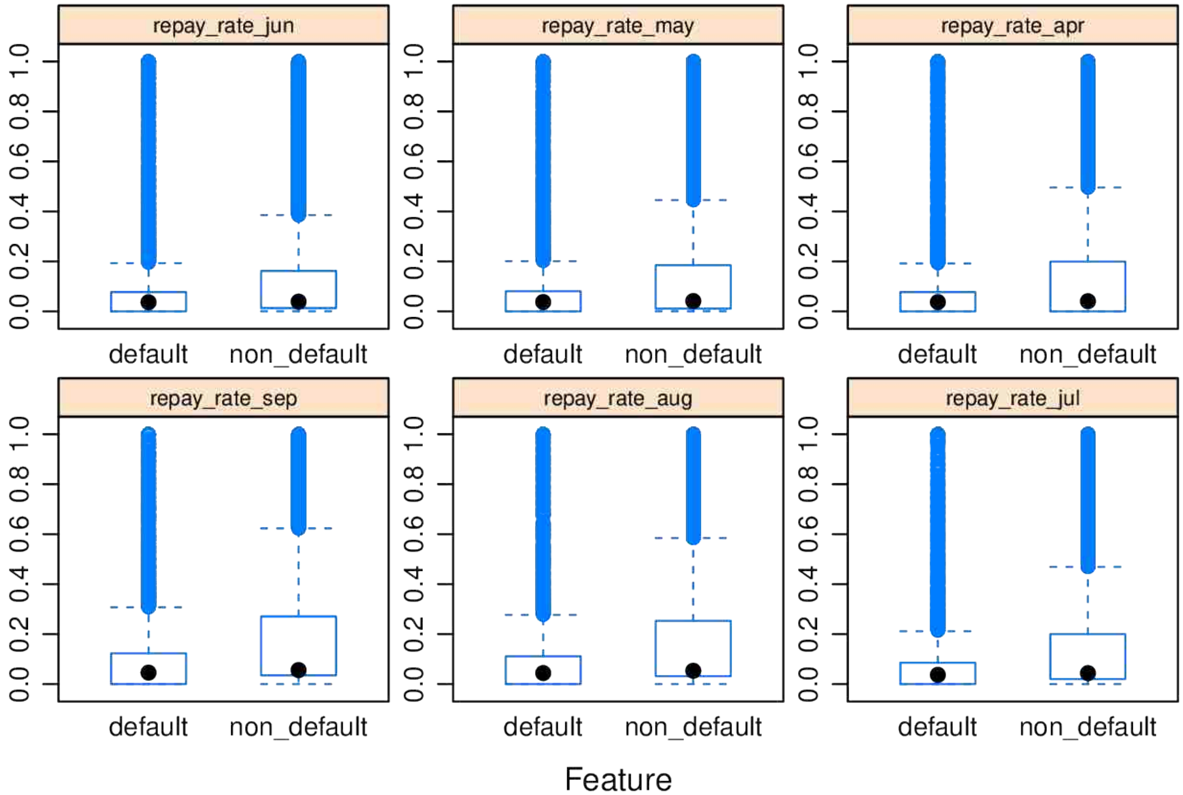
2.4.9 Variable 25-30: repay rate

```r
# We make sure we do not get NaN and Inf by taking care of zeros in our columns,
# set negative payments to zero (since we define this as "nothing was payed"),
# set positive values above one back to one (we also define overpaid as paid) and
```

```r
# calculate the ratio of partially paid bills (proportion of billed to payd)
dataset <- dataset %>%
  mutate(repay_rate_sep = ifelse(bill_sep == 0, 0,
                                 ifelse(prev_payed_sep == 0, 0,
                                        ifelse(prev_payed_sep/bill_sep > 1, 1,
                                               ifelse(prev_payed_sep/bill_sep <= 0, 0,
                                                      prev_payed_sep/bill_sep)))),
         repay_rate_aug = ifelse(bill_aug == 0, 0,
                                 ifelse(prev_payed_aug == 0, 0,
                                        ifelse(prev_payed_aug/bill_aug > 1, 1,
                                               ifelse(prev_payed_aug/bill_aug <= 0, 0,
                                                      prev_payed_aug/bill_aug)))),
         repay_rate_jul = ifelse(bill_jul == 0, 0,
                                 ifelse(prev_payed_jul == 0, 0,
                                        ifelse(prev_payed_jul/bill_jul > 1, 1,
                                               ifelse(prev_payed_jul/bill_jul <= 0, 0,
                                                      prev_payed_jul/bill_jul)))),
         repay_rate_jun = ifelse(bill_jun == 0, 0,
                                 ifelse(prev_payed_jun == 0, 0,
                                        ifelse(prev_payed_jun/bill_jun > 1, 1,
                                               ifelse(prev_payed_jun/bill_jun <= 0, 0,
                                                      prev_payed_jun/bill_jun)))),
         repay_rate_may = ifelse(bill_may == 0, 0,
                                 ifelse(prev_payed_may == 0, 0,
                                        ifelse(prev_payed_may/bill_may > 1, 1,
                                               ifelse(prev_payed_may/bill_may <= 0, 0,
                                                      prev_payed_may/bill_may)))),
         repay_rate_apr = ifelse(bill_apr == 0, 0,
                                 ifelse(prev_payed_apr == 0, 0,
                                        ifelse(prev_payed_apr/bill_apr > 1, 1,
                                               ifelse(prev_payed_apr/bill_apr <= 0, 0,
                                                      prev_payed_apr/bill_apr)))))

# Then we plot our new variables
featurePlot(x = dataset[,25:30], y = dataset$payment_default, plot = "density",
            strip = strip, scales = scales, auto.key = list(columns = 2))
```




```r
featurePlot(x = dataset[,25:30], y = dataset$payment_default, plot = "box",
            strip = strip, scales = scales)
```




```r
dataset[,25:30] %>% summary()
```

```
## repay_rate_sep    repay_rate_aug    repay_rate_jul    repay_rate_jun
## Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.0000000
## 1st Qu.:0.03165   1st Qu.:0.02651   1st Qu.:0.01201   1st Qu.:0.0000677
## Median :0.05278   Median :0.05103   Median :0.04179   Median :0.0380232
## Mean   :0.22715   Mean   :0.22154   Mean   :0.20429   Mean   :0.1908072
## 3rd Qu.:0.21628   3rd Qu.:0.19963   3rd Qu.:0.15335   3rd Qu.:0.1246883
## Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.0000000
## repay_rate_may    repay_rate_apr
## Min.   :0.00000   Min.   :0.00000
## 1st Qu.:0.00000   1st Qu.:0.00000
## Median :0.04008   Median :0.03958
## Mean   :0.20193   Mean   :0.20447
## 3rd Qu.:0.14063   3rd Qu.:0.14697
## Max.   :1.00000   Max.   :1.00000
```

The new variables indeed show something interesting: persons who monthly repayed between 0 and 10% of the billed payments back have a significantly higher risk of being a payment default.

2.5 Modeling

We want to predict if a customer will be a payment default or not, so we are talking about a classification problem. Since we have a lot of observations in our data set, we will use an 80/20 split ratio.

```r
############## Create reduced dataset for model testing ##############

# Set seed fo reproducible results
set.seed(42)

# We generate a an 80/20 split...
split <- createDataPartition(dataset$payment_default, p = 0.8, list = FALSE, times = 1)

# ...and use it for creating our test and train data
train <- dataset[split,]
train.x <- train %>% select(-payment_default)
train.y <- train$payment_default
test <- dataset[-split,]
test.x <- test %>% select(-payment_default)
test.y <- test$payment_default

rm(dataset, split)
```

In order to use the caret package training features properly, we will create a one hot encoded matrix and avoid using the formula interface. By not using the formula interface caret uses less system ressources but we also need to separate the predictor variables from the target variable for both: the train and the test data.

```r
############## Create dummy variables (one hot encoding) ##############

# Since caret assumes that all of the data are numeric, we need to create dummy variables
dummyVariables <- dummyVars(payment_default ~ ., data = train)
dummyVariables
```

```
## Dummy Variable Object
##
## Formula: payment_default ~ .
## 30 variables, 4 factors
## Variables and levels will be separated by '.'
## A less than full rank encoding is used
```

```r
# We create our one hot encoded x-values...
train.x <- data.frame(predict(dummyVariables, newdata = train))

# ...as well as our y-values
train.y <- train$payment_default

train <- train.x
train$payment_default <- train.y

# We also need dummy variables for our test set
dummyVariables <- dummyVars(payment_default ~ ., data = test)
dummyVariables
```

```
## Dummy Variable Object
##
## Formula: payment_default ~ .
## 30 variables, 4 factors
## Variables and levels will be separated by '.'
## A less than full rank encoding is used
```

```
# We create our one hot encoded x-values...
test.x <- data.frame(predict(dummyVariables, newdata = test))

# ...as well as our y-values
test.y <- test$payment_default

test <- test.x
test$payment_default <- test.y

rm(dummyVariables)
```

We know that we have a calss imbalance in our target variable. To tackle this issue, we will use the SMOTE (synthetic minority over-sampling techniqe) package in order to generate more minority-class observations.

```
################ Reducing the class imbalance with SMOTE ################

set.seed(42)
train <- SMOTE(form = payment_default ~ ., data = train)

# Let's see if it worked...
train %>%
  group_by(payment_default) %>% summarize(n = n()) %>% ungroup() %>%
  mutate(percent = round(n/sum(n)*100, 3)) %>% arrange(desc(n))
```

```
## # A tibble: 2 x 3
##   payment_default     n percent
##   <fct>           <int>   <dbl>
## 1 non_default     21136    57.1
## 2 default         15852    42.9
```

```
train.x <- train %>% select(-payment_default)
train.y <- train$payment_default

rm(train.smote)
```

Now that we have generated more default-observations based on our dataset and reduced the imbalance between our two classes, we can define the common training parameters for our models:

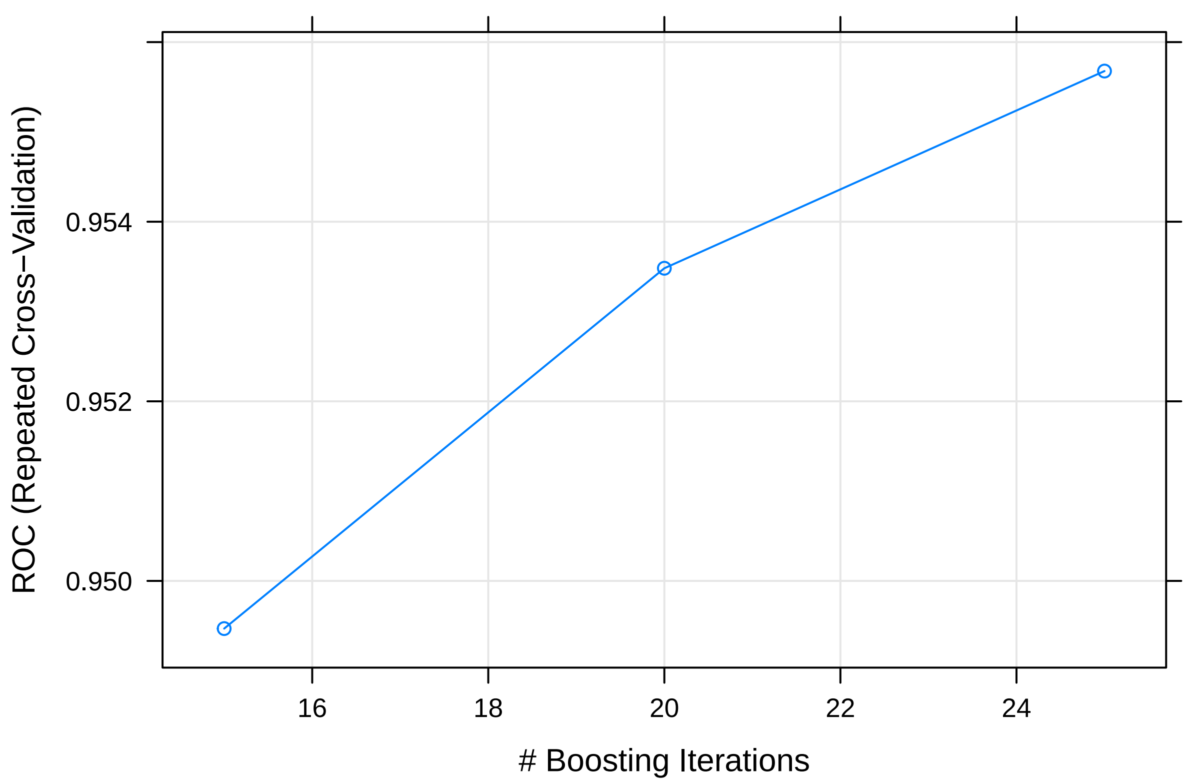
```
############## Train model ##############

metric <- "ROC"

TrainControl <- trainControl(method = "repeatedcv",
                             number = 10,
                             repeats = 3,
                             summaryFunction = twoClassSummary,
                             verboseIter = FALSE,
                             classProbs = TRUE,
                             savePredictions = TRUE,
                             allowParallel = TRUE)
```

In this train control settings, we defined a 10-fold cross validation with 3 repetitions. Since we will use more than one cpu core, verboseIter does not make sense (nothing is shown in this scenario). We also want caret to calculate class probabilities and save predictions.

In this project, we will ebvaluate the performance of different models for our dataset. Each model will be trained and tuned to get the best results possible and compared to each other.

2.5.1 C5.0

```r
# We tune the model (Parameters in modelLookup("C5.0"))...
TuneGrid <- expand.grid(trials = seq(from = 15, to = 25, by = 5),
                        model = "tree",
                        winnow = TRUE)

# ...and train the model
registerDoParallel(parallelCluster)
start.time <- Sys.time() %>% as_datetime()
set.seed(7)
model.c50 <- train(x = train.x, y = train.y, method="C5.0", metric = metric,
                   trControl = TrainControl, na.action = na.pass, tuneGrid = TuneGrid)
model.c50.time <- difftime(as_datetime(Sys.time()), start.time, units = "hours")

# We take a look at the model performance
plot(model.c50)
```




```r
model.c50
```

```
## C5.0
##
## 36988 samples
##    35 predictor
##     2 classes: 'default', 'non_default'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 33289, 33290, 33289, 33289, 33290, 33289, ...
## Resampling results across tuning parameters:
##
##   trials  ROC        Sens       Spec
##   15      0.9494691  0.8116960  0.9475934
##   20      0.9534816  0.8152707  0.9504638
##   25      0.9556775  0.8174156  0.9535863
##
## Tuning parameter 'model' was held constant at a value of tree
##
## Tuning parameter 'winnow' was held constant at a value of TRUE
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were trials = 25, model = tree
##  and winnow = TRUE.
```

```
model.c50.time
```

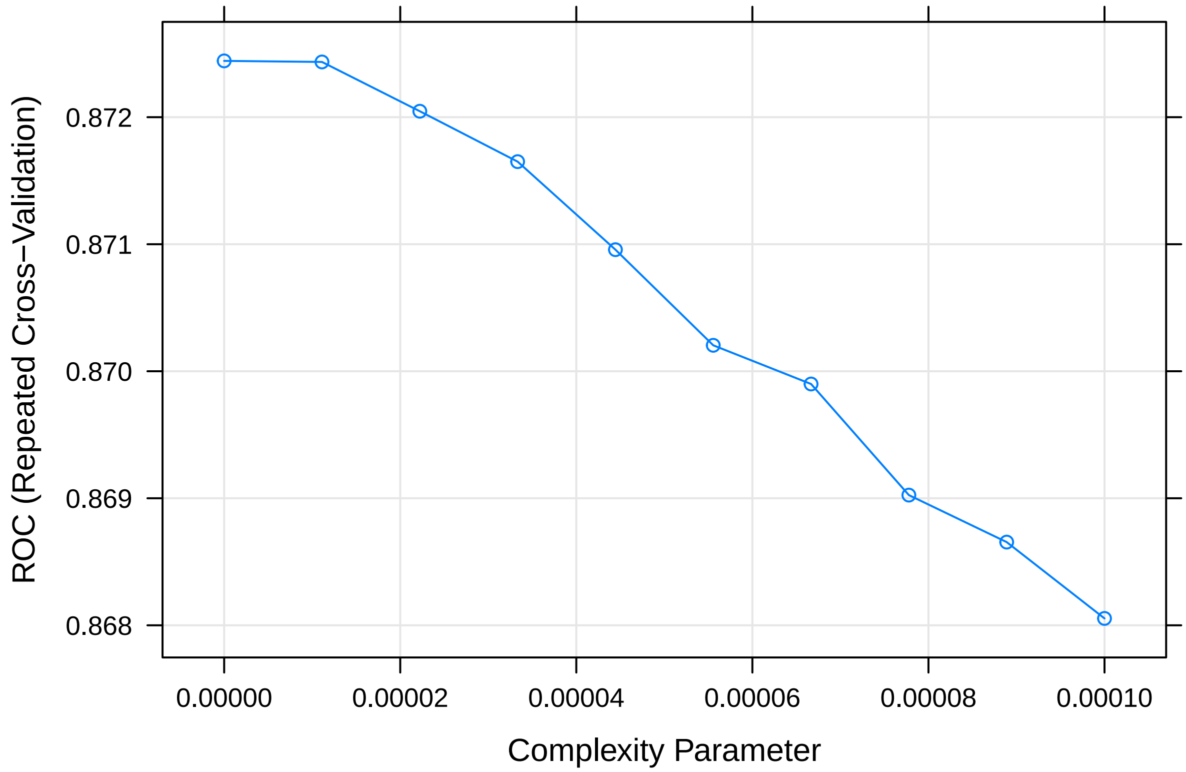
```
## Time difference of 0.09001166 hours
```

```
# We save the model and free memory
save(model.c50, model.c50.time, file = "~/data/model-c50.RData")
rm(model.c50, model.c50.time)
gc(verbose = FALSE, full = TRUE)
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2435579 130.1    4679642 250.0  3635350 194.2
## Vcells 11072243  84.5   26453880 201.9 21978221 167.7
```

2.5.2 RPART

```
# We tune the model (Parameters in modelLookup("rpart"))...
TuneGrid <- expand.grid(cp = seq(from = 0, to = 0.0001, length = 10))

# ...and train the model
registerDoParallel(parallelCluster)
start.time <- Sys.time() %>% as_datetime()
set.seed(7)
model.rpart <- train(x = train.x, y = train.y, method="rpart", metric = metric,
                     trControl = TrainControl, na.action = na.pass, tuneGrid = TuneGrid)
model.rpart.time <- difftime(as_datetime(Sys.time()), start.time, units = "hours")

# We take a look at the model performance
plot(model.rpart)
```




```
model.rpart
```

```
## CART
##
## 36988 samples
##    35 predictor
##     2 classes: 'default', 'non_default'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 33289, 33290, 33289, 33289, 33290, 33289, ...
## Resampling results across tuning parameters:
##
##   cp            ROC        Sens       Spec
##   0.000000e+00  0.8724432  0.7552773  0.8566582
##   1.111111e-05  0.8724348  0.7552984  0.8567686
##   2.222222e-05  0.8720469  0.7550881  0.8577306
##   3.333333e-05  0.8716500  0.7543943  0.8591499
##   4.444444e-05  0.8709571  0.7521444  0.8628561
##   5.555556e-05  0.8702042  0.7512822  0.8645593
##   6.666667e-05  0.8698996  0.7512611  0.8649851
##   7.777778e-05  0.8690248  0.7479598  0.8700318
##   8.888889e-05  0.8686550  0.7470766  0.8708678
##   1.000000e-04  0.8680542  0.7464879  0.8718298
##
## ROC was used to select the optimal model using the largest value.
```

```
## The final value used for the model was cp = 0.
```

```r
model.rpart.time
```

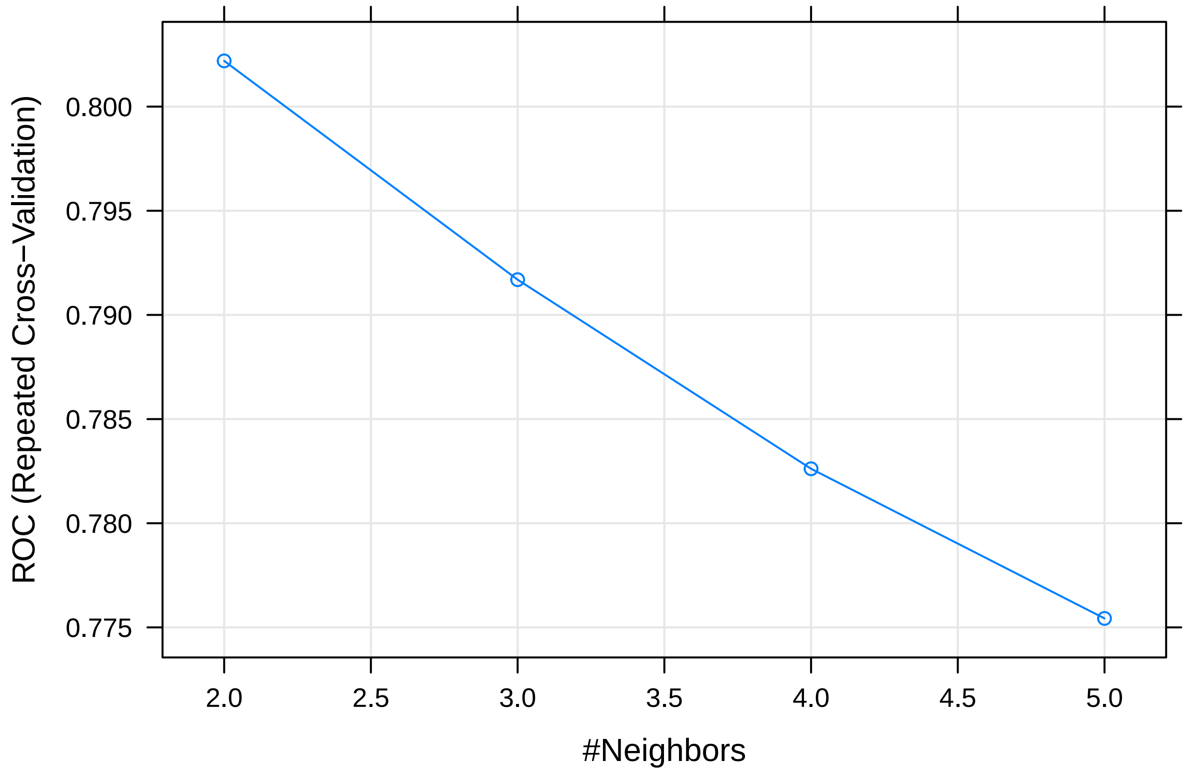
```
## Time difference of 0.004459607 hours
```

```r
# We save the model and free memory
save(model.rpart, model.rpart.time, file = "~/data/model-rpart.RData")
rm(model.rpart, model.rpart.time, TuneGrid)
gc(verbose = FALSE, full = TRUE)
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2444021 130.6    4679642 250.0  3767624 201.3
## Vcells 15824979 120.8   32077973 244.8 32077338 244.8
```

2.5.3 KNN

```r
# We tune the model (Parameters in modelLookup("knn"))...
TuneGrid <- expand.grid(k = seq(from = 2, to = 5, by = 1))

# ...and train the model
registerDoParallel(parallelCluster)
start.time <- Sys.time() %>% as_datetime()
set.seed(7)
model.knn <- train(x = train.x, y = train.y, method="knn", metric = metric,
                   trControl = TrainControl, tuneGrid = TuneGrid)
model.knn.time <- difftime(as_datetime(Sys.time()), start.time, units = "hours")

# We take a look at the model performance
plot(model.knn)
```




```r
model.knn
```

```
## k-Nearest Neighbors
##
## 36988 samples
##    35 predictor
##     2 classes: 'default', 'non_default'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 33289, 33290, 33289, 33289, 33290, 33289, ...
## Resampling results across tuning parameters:
##
##   k  ROC        Sens       Spec
##   2  0.8021949  0.6377952  0.8167423
##   3  0.7916932  0.6117848  0.8194230
##   4  0.7826140  0.5904627  0.8057811
##   5  0.7754308  0.5859834  0.8124366
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 2.
```

```r
model.knn.time
```

```
## Time difference of 0.2295331 hours
```

```r
# We save the model and free memory
save(model.knn, model.knn.time, file = "~/data/model-knn.RData")
```

```r
rm(model.knn, model.knn.time, TuneGrid)
gc(verbose = FALSE, full = TRUE)
```

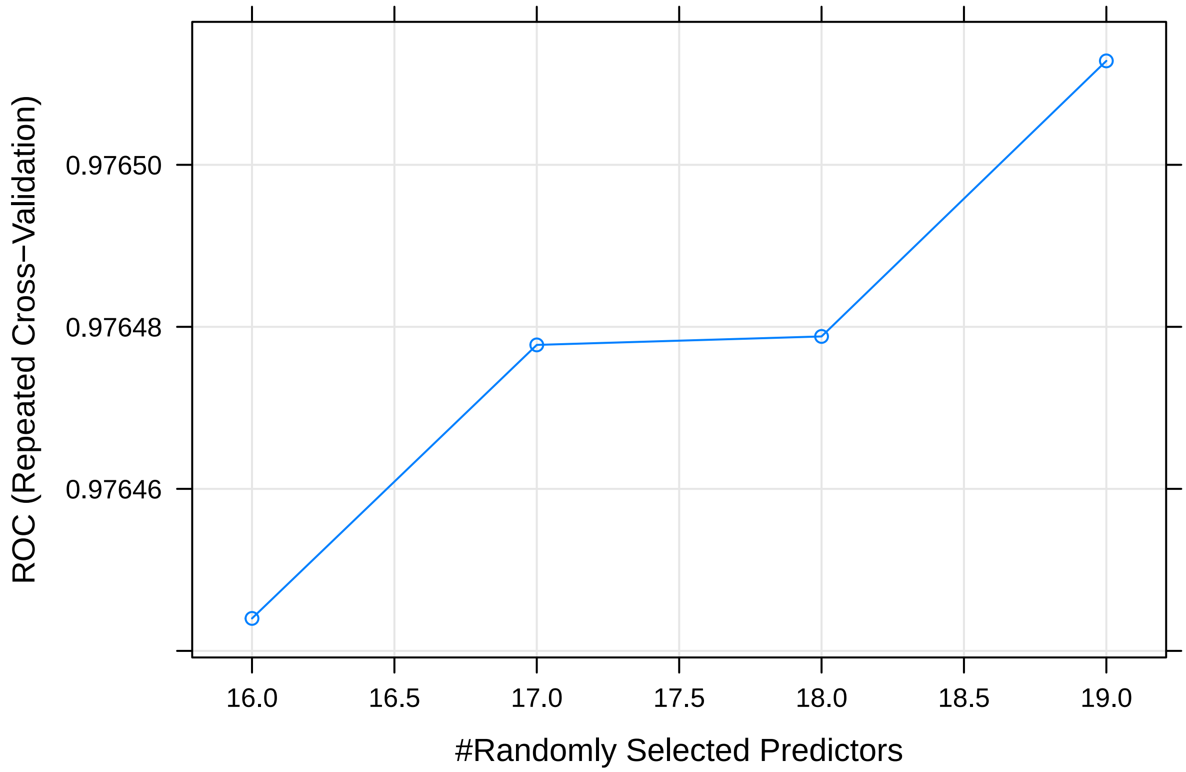
```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2442383 130.5    4679642 250.0  4305472 230.0
## Vcells 10668494  81.4   32077973 244.8 32077338 244.8
```

2.5.4 RANGER

```r
# We tune the model (Parameters in modelLookup("ranger"))...
TuneGrid <- expand.grid(mtry = seq(from = 16, to = 19, by = 1),
                        splitrule = "extratrees", min.node.size = 1)

# ...and train the model
registerDoParallel(parallelCluster)
start.time <- Sys.time() %>% as_datetime()
set.seed(7)
model.ranger <- train(x = train.x, y = train.y, method="ranger", metric = metric,
                      trControl = TrainControl, num.threads = 1, tuneGrid = TuneGrid)
```

```
## Growing trees.. Progress: 24%. Estimated remaining time: 1 minute, 36 seconds.
## Growing trees.. Progress: 49%. Estimated remaining time: 1 minute, 5 seconds.
## Growing trees.. Progress: 73%. Estimated remaining time: 34 seconds.
## Growing trees.. Progress: 98%. Estimated remaining time: 3 seconds.
```

```r
model.ranger.time <- difftime(as_datetime(Sys.time()), start.time, units = "hours")

# We take a look at the model performance
plot(model.ranger)
```




```
model.ranger
```

```
## Random Forest
##
## 36988 samples
##    35 predictor
##     2 classes: 'default', 'non_default'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 33289, 33290, 33289, 33289, 33290, 33289, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   16    0.9764440  0.8818232  0.9530344
##   17    0.9764778  0.8809609  0.9527032
##   18    0.9764788  0.8804775  0.9536653
##   19    0.9765128  0.8793629  0.9540753
##
## Tuning parameter 'splitrule' was held constant at a value of
##  extratrees
## Tuning parameter 'min.node.size' was held constant at a
##  value of 1
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 19, splitrule =
##  extratrees and min.node.size = 1.
```

```
model.ranger.time
```

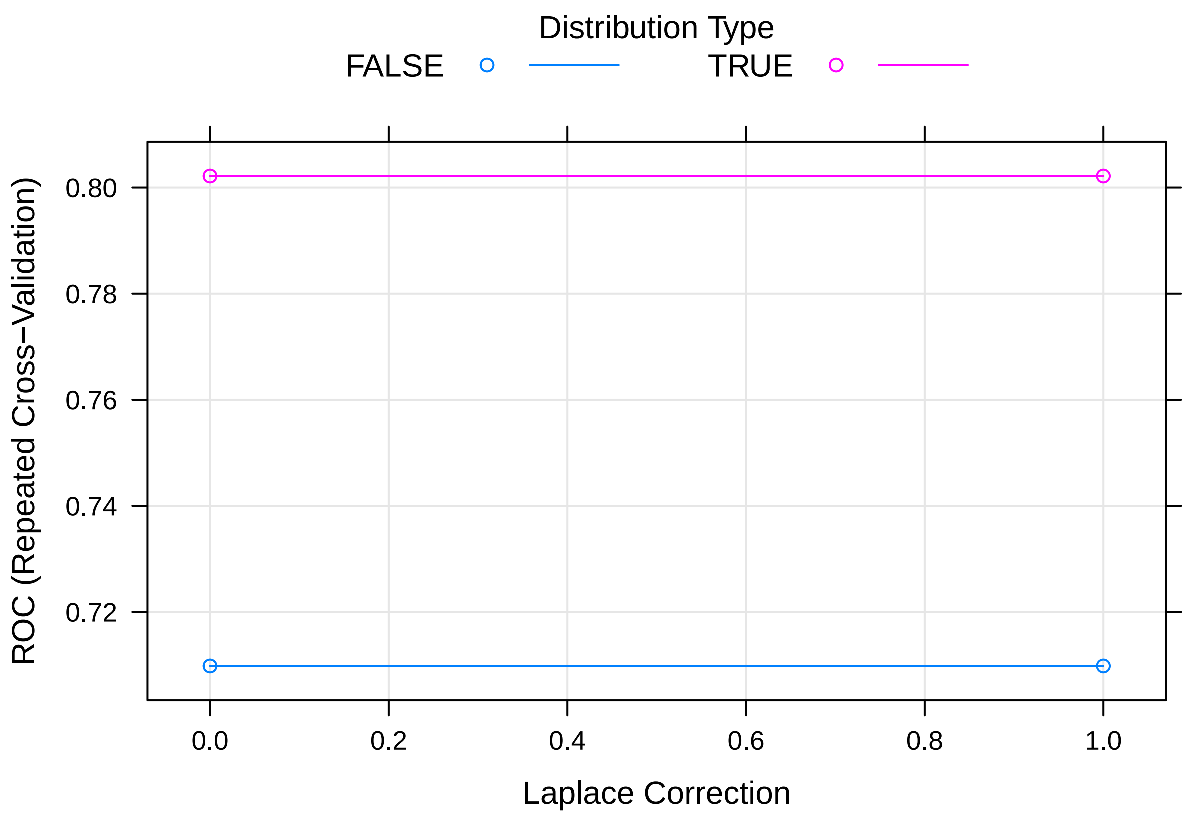
```
## Time difference of 1.075599 hours
```

```
# We save the model and free memory
save(model.ranger, model.ranger.time, file = "~/data/model-ranger.RData")
rm(model.ranger, model.ranger.time, TuneGrid)
gc(verbose = FALSE, full = TRUE)
```

```
##             used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  7630998 407.6   16007690 855.0  8196263 437.8
## Vcells 41315807 315.3   73330096 559.5 49089053 374.6
```

2.5.5 NAIVE BAYES

```
# We tune the model (Parameters in modelLookup("naive_bayes"))...
TuneGrid <- expand.grid(laplace = c(0, 1),
                        usekernel = c(TRUE, FALSE),
                        adjust = 1)

# ...and train the model
registerDoParallel(parallelCluster)
start.time <- Sys.time() %>% as_datetime()
set.seed(7)
model.naivebayes <- train(x = train.x, y = train.y, method="naive_bayes", metric = metric,
                          trControl = TrainControl, na.action = na.pass, tuneGrid = TuneGrid)
model.naivebayes.time <- difftime(as_datetime(Sys.time()), start.time, units = "hours")

# We take a look at the model performance
plot(model.naivebayes)
```




```r
model.naivebayes
```

```
## Naive Bayes
##
## 36988 samples
##    35 predictor
##     2 classes: 'default', 'non_default'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 33289, 33290, 33289, 33289, 33290, 33289, ...
## Resampling results across tuning parameters:
##
##   laplace  usekernel  ROC        Sens       Spec
##   0        FALSE      0.7098193  0.8242698  0.3487581
##   0         TRUE      0.8021704  0.5669100  0.8871755
##   1        FALSE      0.7098193  0.8242698  0.3487581
##   1         TRUE      0.8021704  0.5669100  0.8871755
##
## Tuning parameter 'adjust' was held constant at a value of 1
## ROC 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.
```

```r
model.naivebayes.time
```

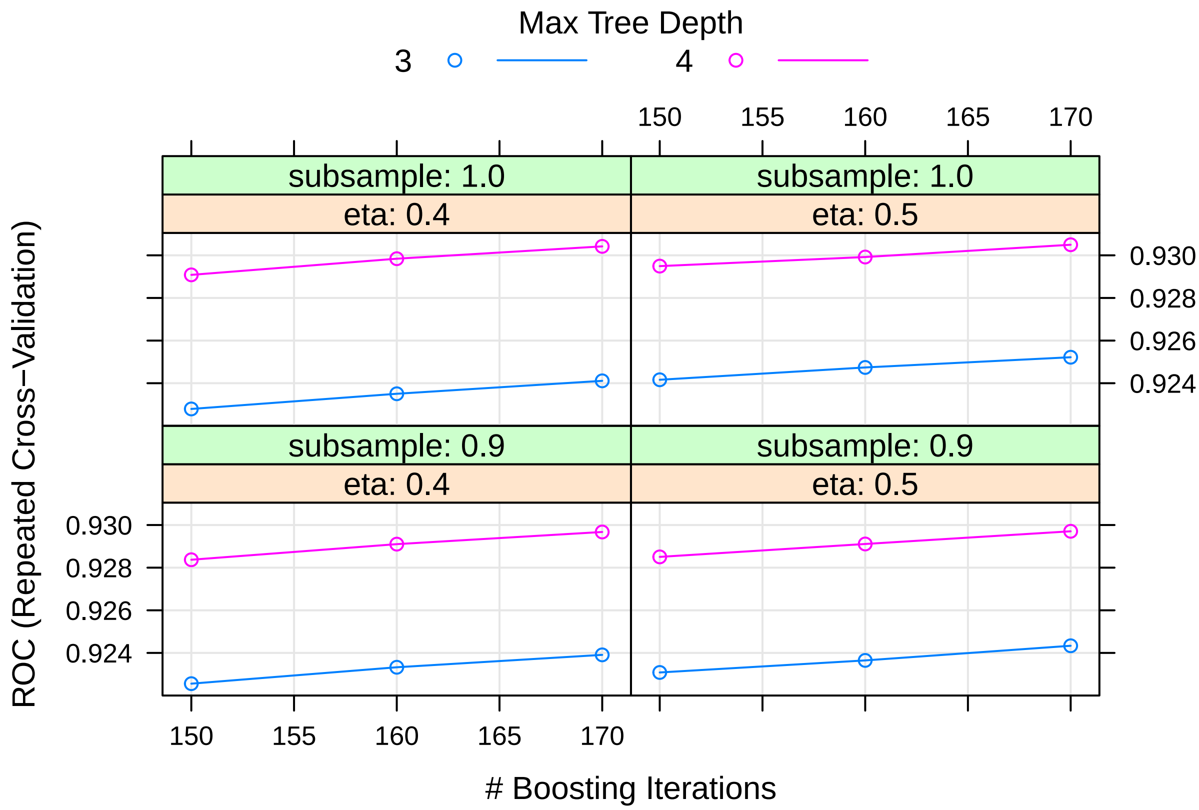
```
## Time difference of 0.002420178 hours
```

```r
# We save the model and free memory
save(model.naivebayes, model.naivebayes.time, file = "~/data/model-naivebayes.RData")
rm(model.naivebayes, model.naivebayes.time, TuneGrid)
gc(verbose = FALSE, full = TRUE)
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2462272 131.5   12806152 684.0  9698563 518.0
## Vcells 11443941  87.4   58664076 447.6 70103433 534.9
```

2.5.6 XGBOOST

```r
# We tune the model (Parameters in modelLookup("xgbTree"))...
TuneGrid <- expand.grid(nrounds = c(150, 160, 170),
                        max_depth = c(3, 4),
                        eta = c(0.4, 0.5),
                        gamma = 0 ,
                        colsample_bytree = c(0.9),
                        min_child_weight = 1, subsample = c(0.9, 1))

# ...and train the model
registerDoParallel(parallelCluster)
start.time <- Sys.time() %>% as_datetime()
set.seed(7)
model.xgboost <- train(x = train.x, y = train.y, method="xgbTree", metric = metric,
                       trControl = TrainControl, nthread = 1, tuneGrid = TuneGrid)
model.xgboost.time <- difftime(as_datetime(Sys.time()), start.time, units = "hours")

# We take a look at the model performance
plot(model.xgboost)
```




```
model.xgboost
```

```
## eXtreme Gradient Boosting
##
## 36988 samples
##    35 predictor
##     2 classes: 'default', 'non_default'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 33289, 33290, 33289, 33289, 33290, 33289, ...
## Resampling results across tuning parameters:
##
##   eta  max_depth  subsample  nrounds  ROC        Sens       Spec
##   0.4  3          0.9        150      0.9225572  0.7693247  0.9460636
##   0.4  3          0.9        160      0.9233277  0.7707967  0.9468837
##   0.4  3          0.9        170      0.9239085  0.7725419  0.9467417
##   0.4  3          1.0        150      0.9227928  0.7697667  0.9467259
##   0.4  3          1.0        160      0.9235042  0.7720797  0.9467102
##   0.4  3          1.0        170      0.9241121  0.7725423  0.9479087
##   0.4  4          0.9        150      0.9283709  0.7847592  0.9496593
##   0.4  4          0.9        160      0.9291035  0.7865254  0.9504479
##   0.4  4          0.9        170      0.9296741  0.7876399  0.9505899
##   0.4  4          1.0        150      0.9290833  0.7839603  0.9518832
##   0.4  4          1.0        160      0.9298419  0.7856215  0.9523564
##   0.4  4          1.0        170      0.9304193  0.7863786  0.9527348
```

```
##   0.5  3          0.9        150      0.9230831  0.7748553  0.9465368
##   0.5  3          0.9        160      0.9236493  0.7764746  0.9469468
##   0.5  3          0.9        170      0.9243361  0.7779886  0.9471518
##   0.5  3          1.0        150      0.9241618  0.7757172  0.9482085
##   0.5  3          1.0        160      0.9247396  0.7767686  0.9487762
##   0.5  3          1.0        170      0.9252191  0.7782407  0.9493282
##   0.5  4          0.9        150      0.9285055  0.7893222  0.9476566
##   0.5  4          0.9        160      0.9291104  0.7902683  0.9479090
##   0.5  4          0.9        170      0.9297061  0.7915512  0.9482087
##   0.5  4          1.0        150      0.9294931  0.7889228  0.9498488
##   0.5  4          1.0        160      0.9299198  0.7899952  0.9504639
##   0.5  4          1.0        170      0.9304960  0.7909413  0.9509370
##
## Tuning parameter 'gamma' was held constant at a value of 0
##
## Tuning parameter 'colsample_bytree' was held constant at a value of
##  0.9
## Tuning parameter 'min_child_weight' was held constant at a value of 1
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 170, max_depth = 4,
##  eta = 0.5, gamma = 0, colsample_bytree = 0.9, min_child_weight = 1
##  and subsample = 1.
```

```r
model.xgboost.time
```

```
## Time difference of 0.1483164 hours
```

```r
# We save the model and free memory
save(model.xgboost, model.xgboost.time, file = "~/data/model-xgboost.RData")
rm(model.xgboost, model.xgboost.time, TuneGrid)
gc(verbose = FALSE, full = TRUE)
```

```
##            used  (Mb) gc trigger  (Mb)  max used  (Mb)
## Ncells  2453681 131.1    8195936 437.8   9698563 518.0
## Vcells 37598386 286.9  122945444 938.0 102386336 781.2
```

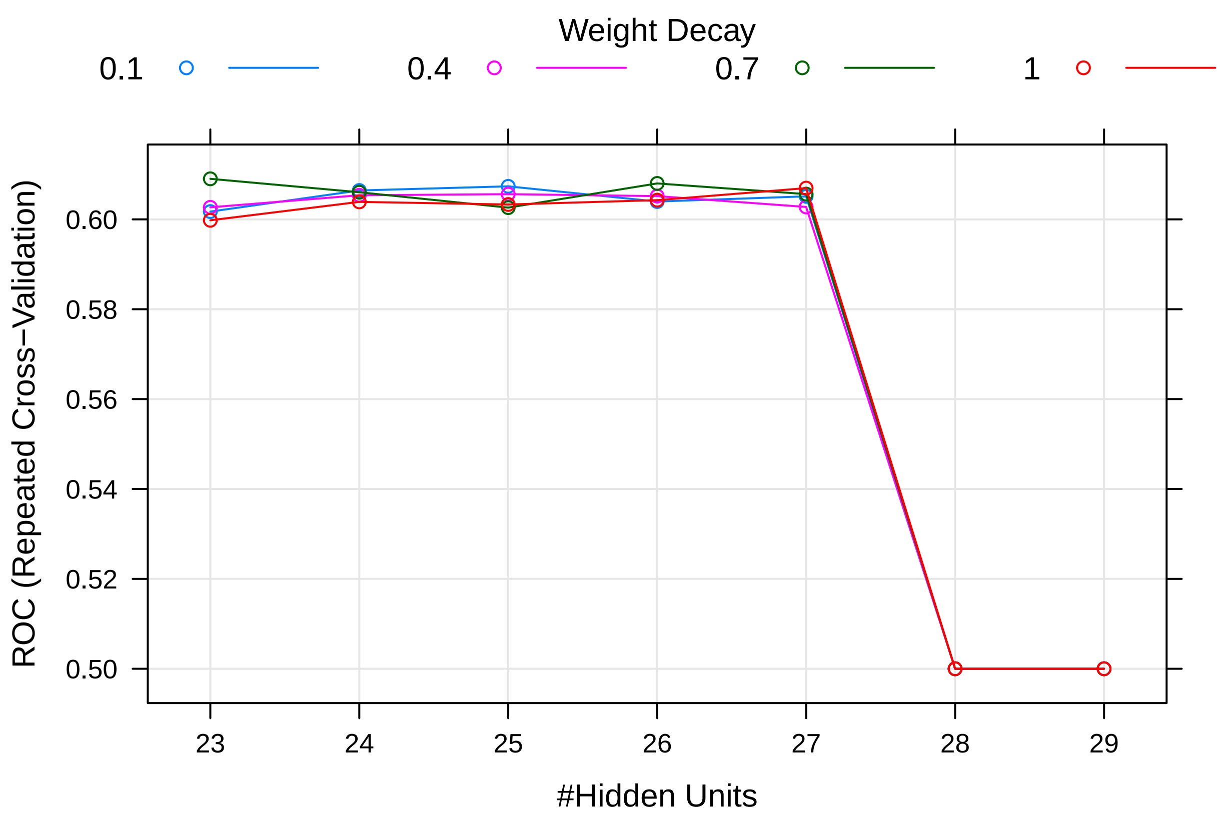
2.5.7 NEURONAL NETWORK

```r
# We tune the model (Parameters in modelLookup("nnet"))...
TuneGrid <- expand.grid(size = seq(from = 23, to = 29, by = 1),
                        decay = seq(from = 0.1, to = 1, length = 4))

# ...and train the model
registerDoParallel(parallelCluster)
start.time <- Sys.time() %>% as_datetime()
set.seed(7)
model.nnet <- train(x = train.x, y = train.y, method="nnet", metric = metric,
                    trControl = TrainControl, na.action = na.pass, tuneGrid = TuneGrid)
```

```
## # weights:  852
## initial  value 25764.182135
## iter  10 value 24929.961238
## iter  20 value 24919.310263
## iter  30 value 24868.948700
## iter  40 value 24826.308579
```

```
## iter  50 value 24785.844398
## iter  60 value 24779.270632
## iter  70 value 24770.288786
## iter  80 value 24767.675389
## iter  90 value 24765.310083
## iter 100 value 24759.255733
## final  value 24759.255733
## stopped after 100 iterations
```

```r
model.nnet.time <- difftime(as_datetime(Sys.time()), start.time, units = "hours")

# We take a look at the model performance
plot(model.nnet)
```




```r
model.nnet
```

```
## Neural Network
##
## 36988 samples
##    35 predictor
##     2 classes: 'default', 'non_default'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 33289, 33290, 33289, 33289, 33290, 33289, ...
## Resampling results across tuning parameters:
##
```

```
##   size  decay  ROC        Sens       Spec
##   23    0.1    0.6017230  0.3830989  0.7634224
##   23    0.4    0.6026757  0.3707357  0.7784104
##   23    0.7    0.6090182  0.3822709  0.7698529
##   23    1.0    0.5997920  0.3981148  0.7517377
##   24    0.1    0.6064256  0.3959135  0.7537828
##   24    0.4    0.6053551  0.3704312  0.7767754
##   24    0.7    0.6060288  0.3958097  0.7578712
##   24    1.0    0.6038818  0.3874307  0.7602380
##   25    0.1    0.6073458  0.3807067  0.7659007
##   25    0.4    0.6056008  0.3931621  0.7573637
##   25    0.7    0.6026105  0.3932594  0.7556919
##   25    1.0    0.6033006  0.3730145  0.7741284
##   26    0.1    0.6039644  0.3953279  0.7530435
##   26    0.4    0.6051672  0.3454906  0.7933845
##   26    0.7    0.6079881  0.4020514  0.7529129
##   26    1.0    0.6042676  0.3822393  0.7655535
##   27    0.1    0.6051139  0.3838449  0.7696636
##   27    0.4    0.6027586  0.4024255  0.7483153
##   27    0.7    0.6056172  0.3664734  0.7770935
##   27    1.0    0.6069431  0.3850386  0.7636719
##   28    0.1    0.5000000        NaN        NaN
##   28    0.4    0.5000000        NaN        NaN
##   28    0.7    0.5000000        NaN        NaN
##   28    1.0    0.5000000        NaN        NaN
##   29    0.1    0.5000000        NaN        NaN
##   29    0.4    0.5000000        NaN        NaN
##   29    0.7    0.5000000        NaN        NaN
##   29    1.0    0.5000000        NaN        NaN
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were size = 23 and decay = 0.7.
```

```r
model.nnet.time
```

```
## Time difference of 1.38924 hours
```

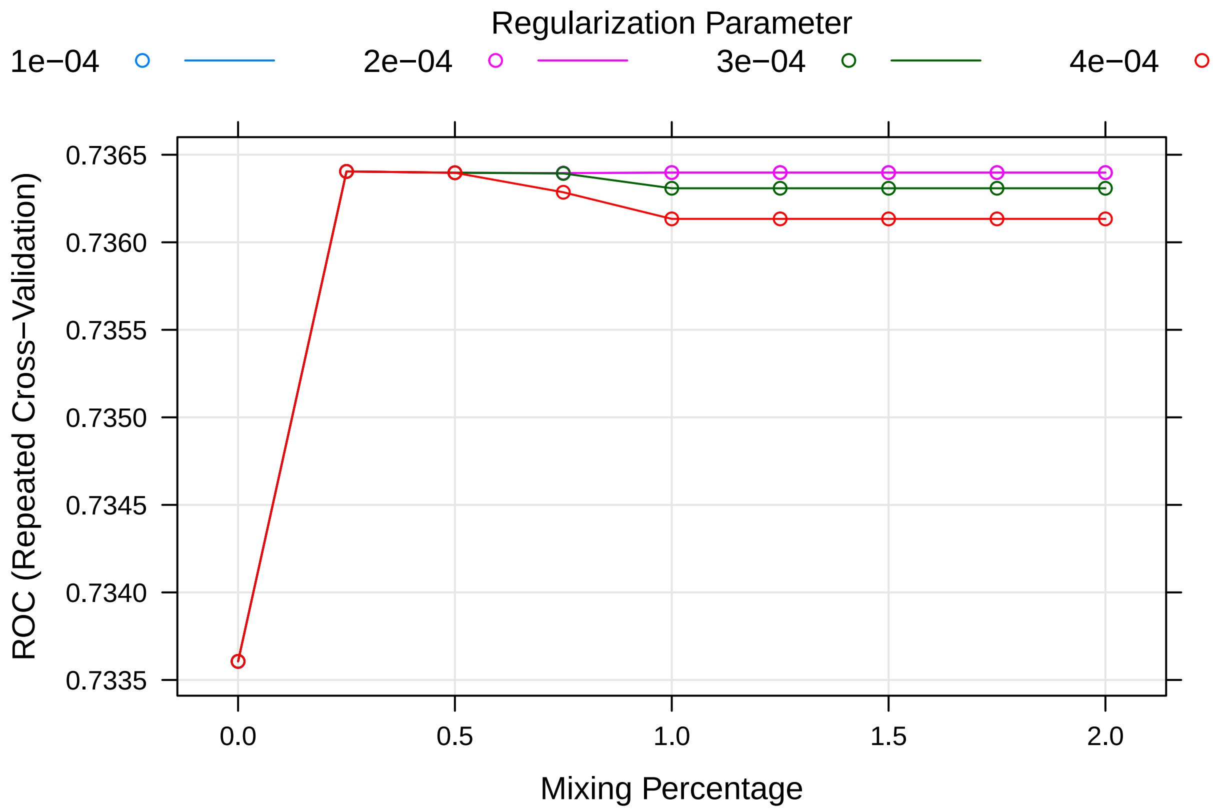
```r
# We save the model and free memory
save(model.nnet, model.nnet.time, file = "~/data/model-nnet.RData")
rm(model.nnet, model.nnet.time, TuneGrid)
gc(verbose = FALSE, full = TRUE)
```

```
##             used  (Mb) gc trigger  (Mb)  max used  (Mb)
## Ncells   2477954 132.4    8195936 437.8   9698563 518.0
## Vcells  28654198 218.7   98356355 750.4 121035002 923.5
```

2.5.8 GLMNET

```r
# We tune the model (Parameters in modelLookup("glmnet"))
TuneGrid <- expand.grid(alpha = seq(from = 0, to = 2, by = 0.25),
                        lambda = seq(from = 0.0001, to = 0.0004, length = 4))

# ...and train the model
registerDoParallel(parallelCluster)
```

```r
start.time <- Sys.time() %>% as_datetime()
set.seed(7)
model.glmnet <- train(x = train.x, y = train.y, method="glmnet", metric = metric,
                        trControl = TrainControl, tuneGrid = TuneGrid)
model.glmnet.time <- difftime(as_datetime(Sys.time()), start.time, units = "hours")

# We take a look at the model performance
plot(model.glmnet)
```




```r
model.glmnet
```

```
## glmnet
##
## 36988 samples
##    35 predictor
##     2 classes: 'default', 'non_default'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 33289, 33290, 33289, 33289, 33290, 33289, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda  ROC        Sens       Spec
##   0.00   1e-04   0.7336059  0.4758807  0.8918115
##   0.00   2e-04   0.7336059  0.4758807  0.8918115
##   0.00   3e-04   0.7336059  0.4758807  0.8918115
```

```
##   0.00   4e-04   0.7336059  0.4758807  0.8918115
##   0.25   1e-04   0.7364047  0.4876774  0.8798258
##   0.25   2e-04   0.7364047  0.4876774  0.8798258
##   0.25   3e-04   0.7364047  0.4876774  0.8798258
##   0.25   4e-04   0.7364047  0.4876774  0.8798258
##   0.50   1e-04   0.7363972  0.4876143  0.8795262
##   0.50   2e-04   0.7363972  0.4876143  0.8795262
##   0.50   3e-04   0.7363972  0.4876143  0.8795262
##   0.50   4e-04   0.7363970  0.4876143  0.8795262
##   0.75   1e-04   0.7363952  0.4877194  0.8794789
##   0.75   2e-04   0.7363952  0.4877194  0.8794789
##   0.75   3e-04   0.7363931  0.4877194  0.8794789
##   0.75   4e-04   0.7362857  0.4874250  0.8800150
##   1.00   1e-04   0.7363982  0.4877195  0.8794631
##   1.00   2e-04   0.7363982  0.4877195  0.8794631
##   1.00   3e-04   0.7363085  0.4875512  0.8797785
##   1.00   4e-04   0.7361337  0.4871726  0.8803304
##   1.25   1e-04   0.7363982  0.4877195  0.8794631
##   1.25   2e-04   0.7363982  0.4877195  0.8794631
##   1.25   3e-04   0.7363085  0.4875512  0.8797785
##   1.25   4e-04   0.7361337  0.4871726  0.8803304
##   1.50   1e-04   0.7363982  0.4877195  0.8794631
##   1.50   2e-04   0.7363982  0.4877195  0.8794631
##   1.50   3e-04   0.7363085  0.4875512  0.8797785
##   1.50   4e-04   0.7361337  0.4871726  0.8803304
##   1.75   1e-04   0.7363982  0.4877195  0.8794631
##   1.75   2e-04   0.7363982  0.4877195  0.8794631
##   1.75   3e-04   0.7363085  0.4875512  0.8797785
##   1.75   4e-04   0.7361337  0.4871726  0.8803304
##   2.00   1e-04   0.7363982  0.4877195  0.8794631
##   2.00   2e-04   0.7363982  0.4877195  0.8794631
##   2.00   3e-04   0.7363085  0.4875512  0.8797785
##   2.00   4e-04   0.7361337  0.4871726  0.8803304
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.25 and lambda = 4e-04.
```

```
model.glmnet.time
```

```
## Time difference of 0.04595848 hours
```

```
# We save the model and free memory
save(model.glmnet, model.glmnet.time, file = "~/data/model-glmnet.RData")
rm(model.glmnet, model.glmnet.time, TuneGrid)
gc(verbose = FALSE, full = TRUE)
```

```
##            used  (Mb) gc trigger  (Mb)  max used  (Mb)
## Ncells  2486647 132.9    8195936 437.8   9698563 518.0
## Vcells 34322084 261.9  118107626 901.1 121035002 923.5
```

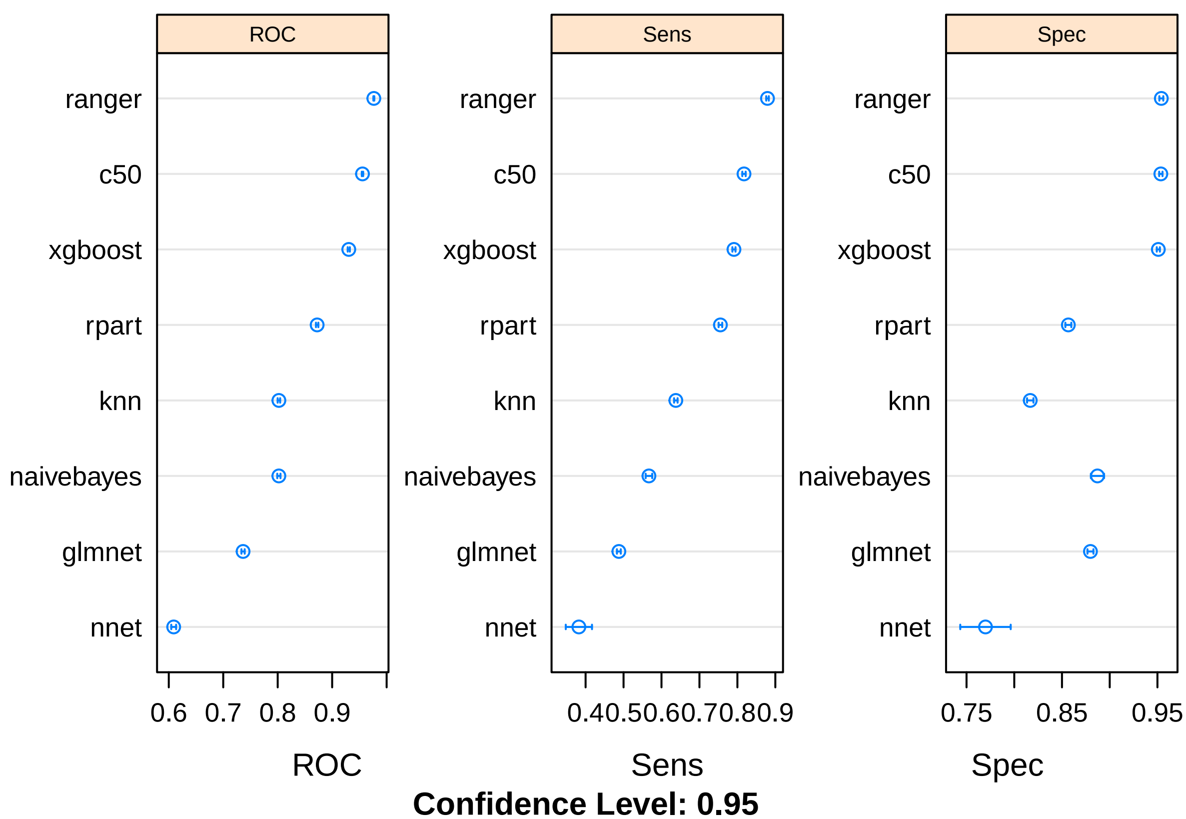
2.6 Model Comparison

After training eight different models we compare their performance.

```r
# We load the previously trained models
load(file = "~/data/model-rpart.RData")
load(file = "~/data/model-knn.RData")
load(file = "~/data/model-ranger.RData")
load(file = "~/data/model-xgboost.RData")
load(file = "~/data/model-naivebayes.RData")
load(file = "~/data/model-nnet.RData")
load(file = "~/data/model-glmnet.RData")
load(file = "~/data/model-c50.RData")

# We summarize the accuracy of models
results <- resamples(list(rpart = model.rpart, knn = model.knn, ranger = model.ranger,
                          naivebayes = model.naivebayes, xgboost = model.xgboost,
                          nnet = model.nnet, glmnet = model.glmnet, c50 = model.c50))
```

Now that we have fit our models, we compare them regarding their respective performance. In order to compare our models, we use the metrics ROC (actually area under curve) and sensitivity as well as specificity.

```r
# We take a first look at the models overall accuracies
dotplot(results, scales = scales, strip = strip)
```

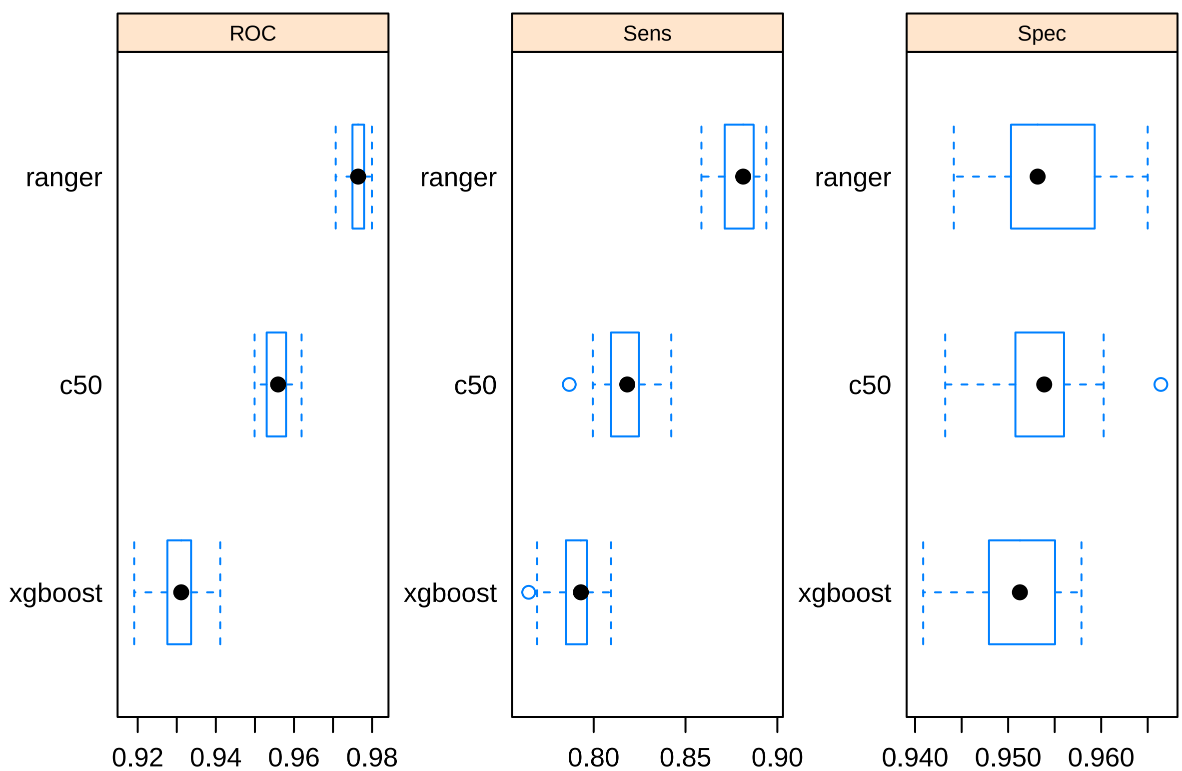



```r
# Show all model training times in comparison
data.frame(model = c("ranger", "c50", "xgboost", "rpart", "knn", "naivebayes",
                     "glmnet", "nnet"),
           time = c(model.ranger.time, model.c50.time, model.xgboost.time,
                    model.rpart.time, model.knn.time, model.naivebayes.time,
                    model.glmnet.time, model.nnet.time)) #%>% arrange(desc(time))
```

```
##        model                time
## 1     ranger 1.075598995 hours
## 2        c50 0.090011656 hours
## 3    xgboost 0.148316384 hours
## 4      rpart 0.004459607 hours
## 5        knn 0.229533100 hours
## 6 naivebayes 0.002420178 hours
## 7     glmnet 0.045958481 hours
## 8       nnet 1.389240255 hours
```

A first look at the used training times also show a big bandwidth of time consumption over all models. Ranked by their overall ROC performance, our top three models are ranger, c50 and xgboost. Lets take a closer look to our top 3 models:

```
# We summarize the accuracy of models
results <- resamples(list(c50 = model.c50, xgboost = model.xgboost, ranger = model.ranger))

# We take a first look at the models overall accuracies
bwplot(results, scales = scales, strip = strip)
```




```
# Show top 3 model training times in comparison
data.frame(model = c("xgboost", "ranger", "c50"),
           time = c(model.xgboost.time, model.ranger.time, model.c50.time)) %>% arrange(desc(time))
```

```
##     model             time
## 1  ranger 1.07559900 hours
## 2 xgboost 0.14831638 hours
## 3     c50 0.09001166 hours
```

When it comes to overall ROC and sensitivity performance, our ranger model is clearly the winner. If we take a look at the specificity it is not so clear anymore, since our top three models have similar specificity performance. Our ranger model took more than twice as much time as the xgboost model and even more than ten times more than the c50 model.

2.7 Predictions

In this final step we will use our top three models to predict outcomes and evaluate their performance on the test data.

```
############## Make Predictions ##############
# RANGER
model.ranger.predictions <- predict(model.ranger, test.x, na.action = na.pass)
# C50
model.c50.predictions <- predict(model.c50, test.x, na.action = na.pass)
# XGBOOST
model.xgboost.predictions <- predict(model.xgboost, test.x, na.action = na.pass)

############## Evaluate Performance with Confusion Matrix ##############
# RANGER
confusionMatrix(reference = test.y, data = model.ranger.predictions, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    default non_default
##   default         697         595
##   non_default     624        4004
##
##                Accuracy : 0.7941
##                  95% CI : (0.7836, 0.8043)
##     No Information Rate : 0.7769
##     P-Value [Acc > NIR] : 0.0006978
##
##                   Kappa : 0.4014
##
##  Mcnemar's Test P-Value : 0.4225724
##
##             Sensitivity : 0.5276
##             Specificity : 0.8706
##          Pos Pred Value : 0.5395
##          Neg Pred Value : 0.8652
##               Precision : 0.5395
##                  Recall : 0.5276
##                      F1 : 0.5335
##              Prevalence : 0.2231
##          Detection Rate : 0.1177
##    Detection Prevalence : 0.2182
##       Balanced Accuracy : 0.6991
##
##        'Positive' Class : default
##
```

```r
# C50
confusionMatrix(reference = test.y, data = model.c50.predictions, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    default non_default
##   default         649         553
##   non_default     672        4046
##
##                Accuracy : 0.7931
##                  95% CI : (0.7825, 0.8033)
##     No Information Rate : 0.7769
##     P-Value [Acc > NIR] : 0.0013290
##
##                   Kappa : 0.3834
##
##  Mcnemar's Test P-Value : 0.0007478
##
##             Sensitivity : 0.4913
##             Specificity : 0.8798
##          Pos Pred Value : 0.5399
##          Neg Pred Value : 0.8576
##               Precision : 0.5399
##                  Recall : 0.4913
##                      F1 : 0.5145
##              Prevalence : 0.2231
##          Detection Rate : 0.1096
##    Detection Prevalence : 0.2030
##       Balanced Accuracy : 0.6855
##
##        'Positive' Class : default
##
```

```r
# XGBOOST
confusionMatrix(reference = test.y, data = model.xgboost.predictions, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    default non_default
##   default         525         390
##   non_default     796        4209
##
##                Accuracy : 0.7997
##                  95% CI : (0.7892, 0.8098)
##     No Information Rate : 0.7769
##     P-Value [Acc > NIR] : 1.073e-05
##
##                   Kappa : 0.3511
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.39743
##             Specificity : 0.91520
```

```
##          Pos Pred Value : 0.57377
##          Neg Pred Value : 0.84096
##               Precision : 0.57377
##                  Recall : 0.39743
##                      F1 : 0.46959
##              Prevalence : 0.22314
##          Detection Rate : 0.08868
##    Detection Prevalence : 0.15456
##       Balanced Accuracy : 0.65631
##
##        'Positive' Class : default
##
```

Based on the outcome of our test data, our xgboost model gets the best overall accuracy, while ranger is still second. c50 takes the third place.

3 Results

We will break down the results into three categories: costs, in-sample performance and out-of-sample performance.

3.1 Costs

The ranger model took more than twice as much time as the xgboost model and even more than ten times more than the c50 model. If computing costs (time, money) are a critical decision factor to choose the correct model this has to be considered.

3.2 In-sample performance

The ranger model does not only provide the best sensitivity, it also provides the overall best specificity. C50 and xgboost show not that good results.

3.3 Out-of-sample performance

When it comes to the overall performance and specificity, xgboost shows the best results compared to the other two models. If the model needs to have the best sensitivity, the ranger model delivers the best result (although ~0.5276 is not really a good result).

4 Conclusion

We used the dataset of a credit card company from Taiwan where we trained models that predict if a customer will default or not. Unfortunately, the out-of-sample performance could be better (especially regarding to the sinsitivity). Evaluating additional models and their performance could probably help get better results.

Another possible next step could be the usage of ensembles, which usually gives the results another boost but takes a lot more system ressources than our system environment provided (our system had 8 Cores with 32GBM RAM).

With a bigger number of cpu cores and system memory it might also be interesting using more than just one version of the 80/20 data split, since we might not have gotten the best result with our data partition.