Who wants a deposit?

2024-01-19

Introduction

The data we will use is related to marketing campaign of Portuguese banking. The data was collected in 2008-2010 when the big financial crisis happened. Our goal in the project is to find out if the client will open a deposit. In dataset it is marked by variable y which has values yes (if person opens a account) and no (in opposite). We obtained the data from Kaggle.com (https://www.kaggle.com/datasets/alexkataev/bank-marketing-data-set?fbclid=IwAR0H7PwD-OyhDKk14-ORzLdXEpEgff5vO_7Zt2vVSDhJvwZFFFBWxx-2eKI).

The campaigns were run by Portuguese institution and they were based on phone calls. Often it was needed to contact a client several times for instance to find out of a person subscribed a product. We notice we have both numeric and categorical types of data.

Data's description

We can categorize input variables into four groups:

Bank client data – type and description

1. age (numeric) – specifies age of client
2. job type of job (categorical) – gives information about person's work
3. marital (categorical) - marital status
4. education (categorical) – shows level of education
5. default (categorical) – checks if a client has a credit default earlier
6. housing (categorical) - checks if a client has a housing loan
7. loan (categorical) – has a client a personal loan

Related with the last contact of the current campaign

1. contact (categorical) – shows a type of contact
2. month (categorical) – in which month of the year the last contact occurred
3. day_of_week (categorical) - in which day of the week the last contact occurred
4. duration: (numeric) - last contact duration, in seconds.

Other attributes

1. campaign client (numeric, includes last contact) - number of contacts performed during this campaign and for this
2. pdays (numeric; 999 means client was not previously contacted) - number of days that passed by after the client was last contacted from a previous campaign
3. previous (numeric) - number of contacts performed before this campaign and for this client
4. poutcome (categorical) - outcome of the previous marketing campaign

Social and economic context attributes

1. emp.var.rate (numeric) - employment variation rate - quarterly indicator
2. cons.price.idx (numeric) - consumer price index, adequately scaled Portuguese inflation rate - monthly indicator
3. cons.conf.idx (numeric) - consumer confidence index - monthly indicator
4. euribor3m (numeric) - euribor 3 month rate - daily indicator
5. nr.employed (numeric) - number of employees altogether - quarterly indicator

Output variable (desired target)

1. y (binary: 'yes','no') - has the client subscribed a term deposit

The data

In this section we are going to get to know more about the data.

Used libraries

```
library(dplyr)
library(caret)
library(readr)
library(tidymodels)
library(corrplot)
library(ggplot2)
library(gmodels)
library(neuralnet)
library(C50)
library(rpart)
library(randomForest)
library(pROC)
```

Firstly we load the dataset then we check names of columns. We also use function summary to check properties of our data.

```
df<- read.csv("https://raw.githubusercontent.com/StanislawC/bank-marketing/main/bank-additional-full.csv
colnames(df)
```

```
##  [1] "age"            "job"            "marital"        "education"
##  [5] "default"        "housing"        "loan"           "contact"
##  [9] "month"          "day_of_week"    "duration"       "campaign"
## [13] "pdays"          "previous"       "poutcome"       "emp.var.rate"
## [17] "cons.price.idx" "cons.conf.idx"  "euribor3m"      "nr.employed"
## [21] "y"
```

```
summary(df)
```

```
##       age            job              marital           education
##  Min.   :17.00   Length:41188       Length:41188       Length:41188
##  1st Qu.:32.00   Class :character   Class :character   Class :character
##  Median :38.00   Mode  :character   Mode  :character   Mode  :character
##  Mean   :40.02
##  3rd Qu.:47.00
##  Max.   :98.00
##    default            housing              loan             contact
##  Length:41188       Length:41188       Length:41188       Length:41188
```

```
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##     month           day_of_week          duration         campaign
##  Length:41188       Length:41188       Min.   :   0.0   Min.   : 1.000
##  Class :character   Class :character   1st Qu.: 102.0   1st Qu.: 1.000
##  Mode  :character   Mode  :character   Median : 180.0   Median : 2.000
##                                        Mean   : 258.3   Mean   : 2.568
##                                        3rd Qu.: 319.0   3rd Qu.: 3.000
##                                        Max.   :4918.0   Max.   :56.000
##      pdays          previous       poutcome          emp.var.rate
##  Min.   :  0.0   Min.   :0.000   Length:41188       Min.   :-3.40000
##  1st Qu.:999.0   1st Qu.:0.000   Class :character   1st Qu.:-1.80000
##  Median :999.0   Median :0.000   Mode  :character   Median : 1.10000
##  Mean   :962.5   Mean   :0.173                      Mean   : 0.08189
##  3rd Qu.:999.0   3rd Qu.:0.000                      3rd Qu.: 1.40000
##  Max.   :999.0   Max.   :7.000                      Max.   : 1.40000
##  cons.price.idx  cons.conf.idx     euribor3m      nr.employed
##  Min.   :92.20   Min.   :-50.8   Min.   :0.634   Min.   :4964
##  1st Qu.:93.08   1st Qu.:-42.7   1st Qu.:1.344   1st Qu.:5099
##  Median :93.75   Median :-41.8   Median :4.857   Median :5191
##  Mean   :93.58   Mean   :-40.5   Mean   :3.621   Mean   :5167
##  3rd Qu.:93.99   3rd Qu.:-36.4   3rd Qu.:4.961   3rd Qu.:5228
##  Max.   :94.77   Max.   :-26.9   Max.   :5.045   Max.   :5228
##       y
##  Length:41188
##  Class :character
##  Mode  :character
##
##
##
```

Since we want to predict whether the client will open a deposit we are going to clean and make some changes to the dataset. It is worth to mentioning that if variable duration is equal 0 then y is equal no. Therefore once the phone call is completed y is known, we should not use this variable then. We cannot reject any variable now firstly we need to do analysis.

Data cleaning

Missing values

To look for NaNs we use table function for each variable. Due to it we also have a better look at data set.

```r
table(df$job)
```

```
##
##        admin.   blue-collar  entrepreneur     housemaid    management
##         10422          9254          1456          1060          2924
##       retired self-employed      services       student    technician
##          1720          1421          3969           875          6743
##    unemployed       unknown
##          1014           330
```

```r
table(df$default)
```

```
##
##      no unknown     yes
##   32588    8597       3
```

```r
table(df$campaign)
```

```
##
##     1     2     3     4     5     6     7     8     9    10    11    12    13
## 17642 10570  5341  2651  1599   979   629   400   283   225   177   125    92
##    14    15    16    17    18    19    20    21    22    23    24    25    26
##    69    51    51    58    33    26    30    24    17    16    15     8     8
##    27    28    29    30    31    32    33    34    35    37    39    40    41
##    11     8    10     7     7     4     4     3     5     1     1     2     1
##    42    43    56
##     2     2     1
```

```r
table(df$pdays)
```

```
##
##     0     1     2     3     4     5     6     7     8     9    10    11    12
##    15    26    61   439   118    46   412    60    18    64    52    28    58
##    13    14    15    16    17    18    19    20    21    22    25    26    27
##    36    20    24    11     8     7     3     1     2     3     1     1     1
##   999
## 39673
```

```r
table(df$previous)
```

```
##
##     0     1     2     3     4     5     6     7
## 35563  4561   754   216    70    18     5     1
```

```r
table(df$poutcome)
```

```
##
##     failure nonexistent     success
##        4252       35563        1373
```

```r
table(df$marital)
```

```
##
## divorced  married   single  unknown
##     4612    24928    11568       80
```

```r
table(df$education)
```

```
##
##            basic.4y            basic.6y            basic.9y         high.school
##                4176                2292                6045                9515
##          illiterate professional.course   university.degree             unknown
##                  18                5243               12168                1731
```

```r
table(df$housing)
```

```
##
##      no unknown     yes
##   18622     990   21576
```

```r
table(df$loan)
```

```
##
##      no unknown     yes
##   33950     990    6248
```

```r
table(df$contact)
```

```
##
##  cellular telephone
##     26144     15044
```

```r
table(df$month)
```

```
##
##   apr   aug   dec   jul   jun   mar   may   nov   oct   sep
##  2632  6178   182  7174  5318   546 13769  4101   718   570
```

```r
table(df$y)
```

```
##
##    no   yes
## 36548  4640
```

Conclusions:

- Yes to no ratio for target variable is equal $\frac{4640}{36548} = 0,127$. We should consider upsampling the set before making models.
- We see value "unknown" for a lot of categorical data. We can consider them as missing values and just delete it or leave it as they are. We choose first option and assume they are missing so we will delete this values. However before we do it let's select columns for further work. It allows us of keeping more data.

Now we can drop some columns: duration (strictly related with y), default (only 3 yes), pdays (weird 999 value), previous (related with poutcome). We also see the variable month do not help when year is unknown. It is also related with economical features.

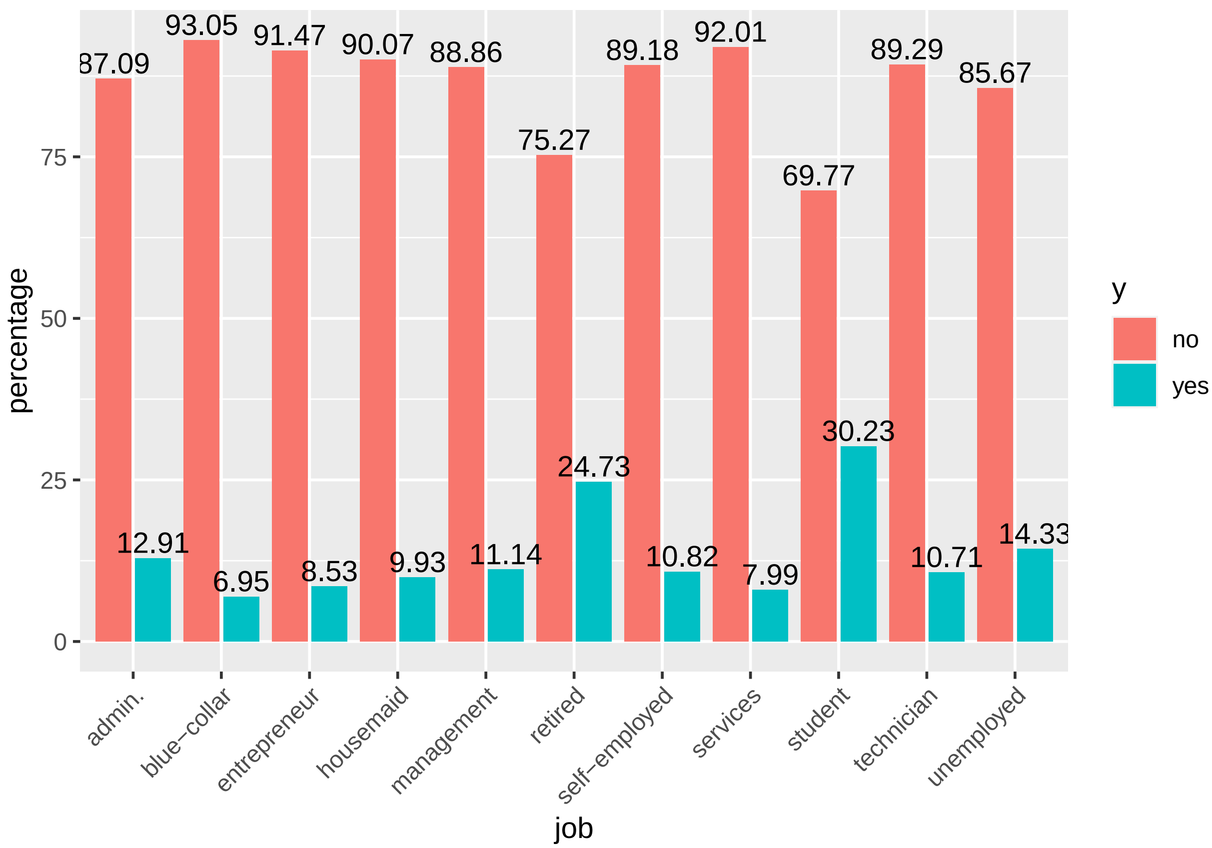
```r
df <- subset(df, select = -duration)
df <- subset(df, select = -pdays)
df <- subset(df, select = -previous)
df <- subset(df, select = -default)
df <- subset(df, select = -month)
```

Now we delete "unknown" values.

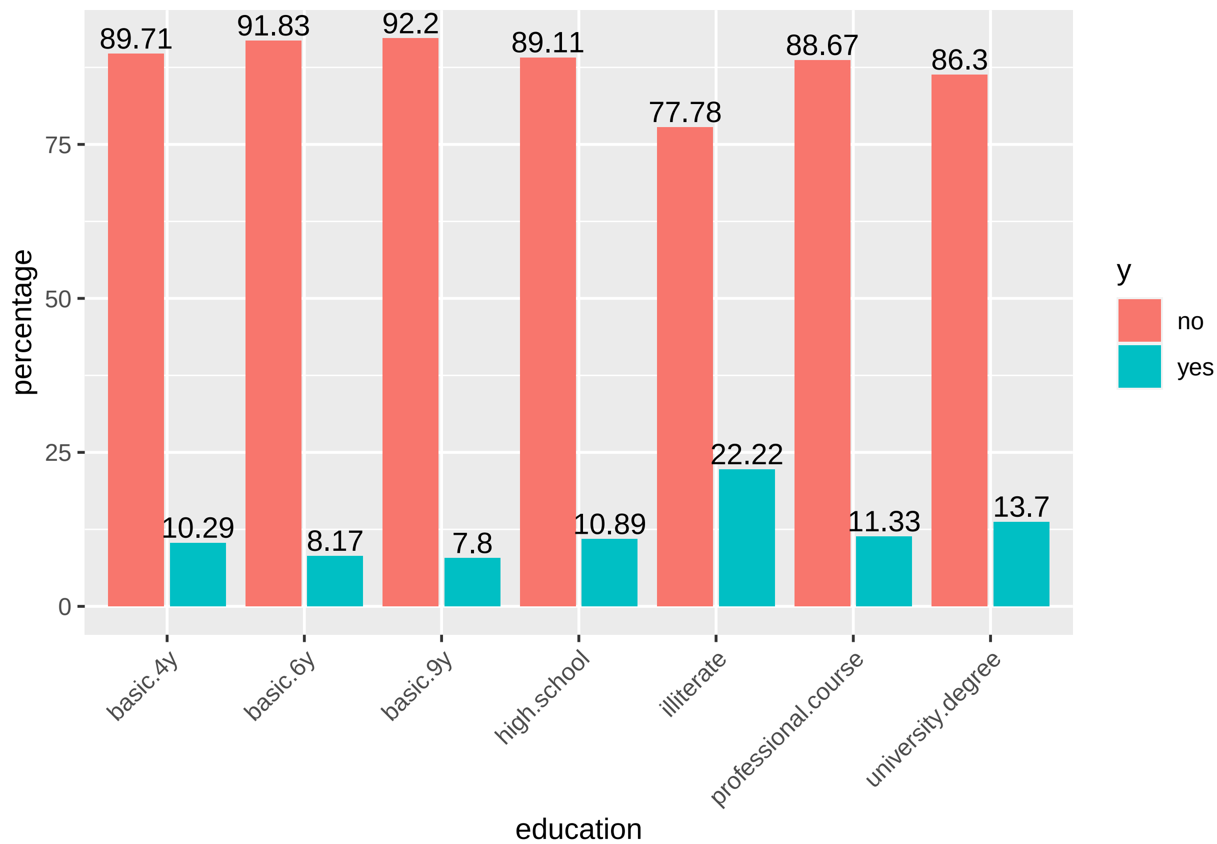
```r
df <- filter(df, job!= "unknown")
df <- filter(df, marital!= "unknown")
df <- filter(df, education!= "unknown")
df <- filter(df, housing!= "unknown")
df <- filter(df, loan!= "unknown")
```

Data visualization

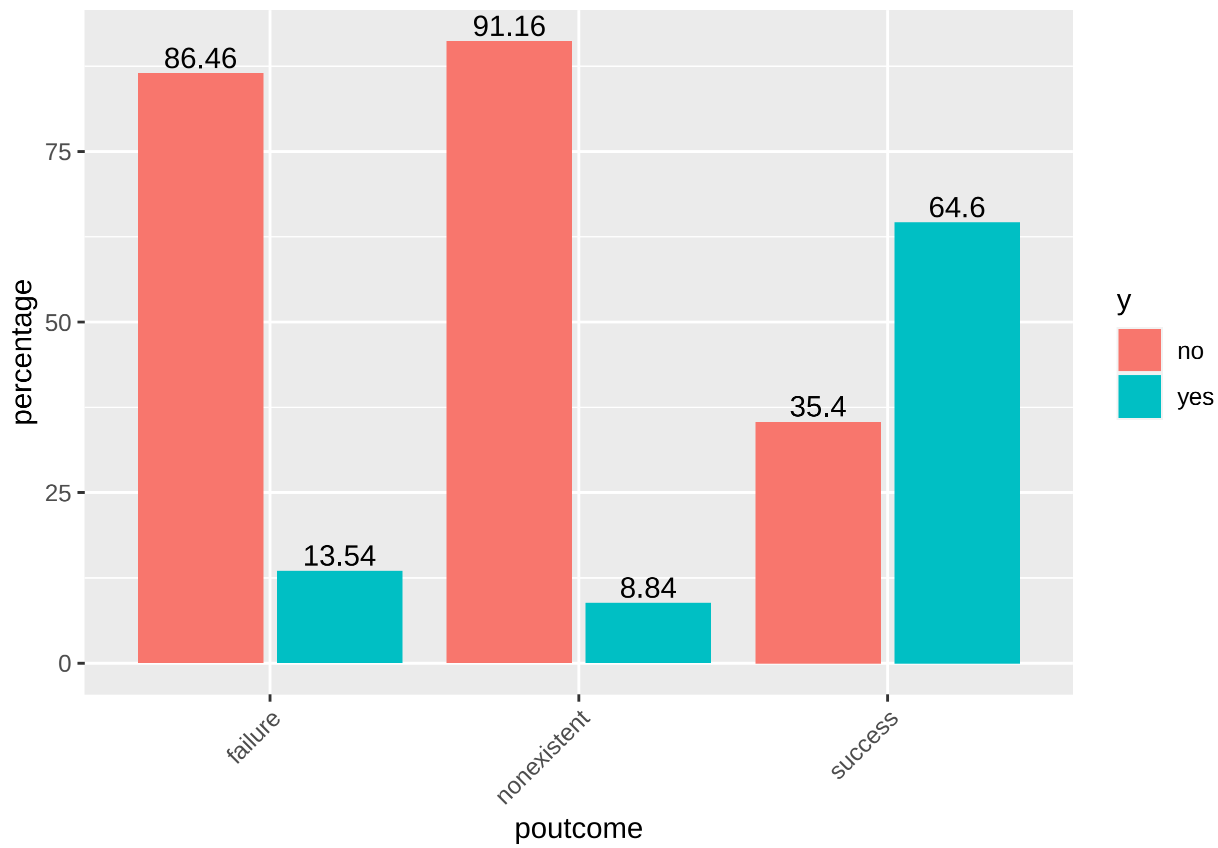
Now let's take a look at percentage representation of positive and negative answers for some variables.



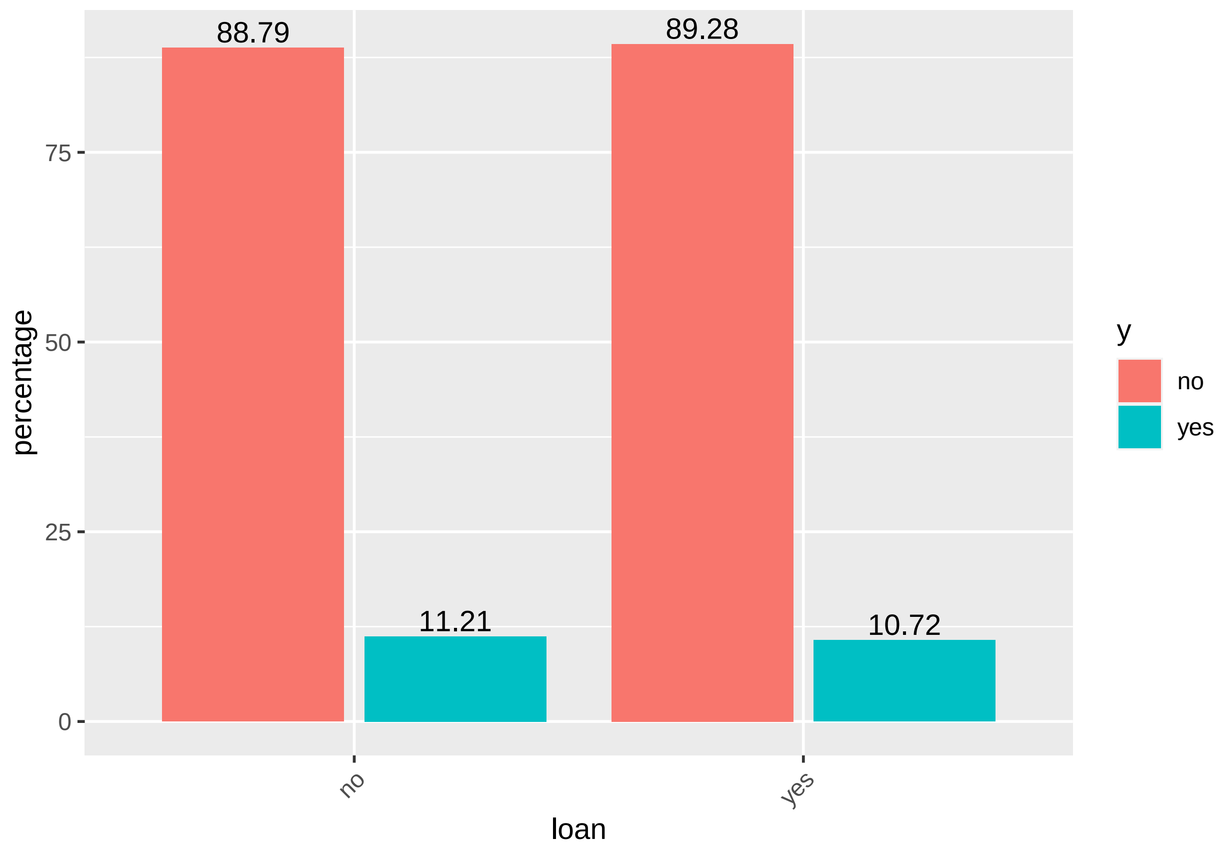
87.09
93.05
91.47
90.07
88.86
75.27
89.18
92.01
69.77
89.29
85.67
12.91
6.95
8.53
9.93
11.14
24.73
10.82
7.99
30.23
10.71
14.33
percentage
75
50
25
0
admin.
blue−collar
entrepreneur
housemaid
management
retired
self−employed
services
student
technician
unemployed
job
y
no
yes



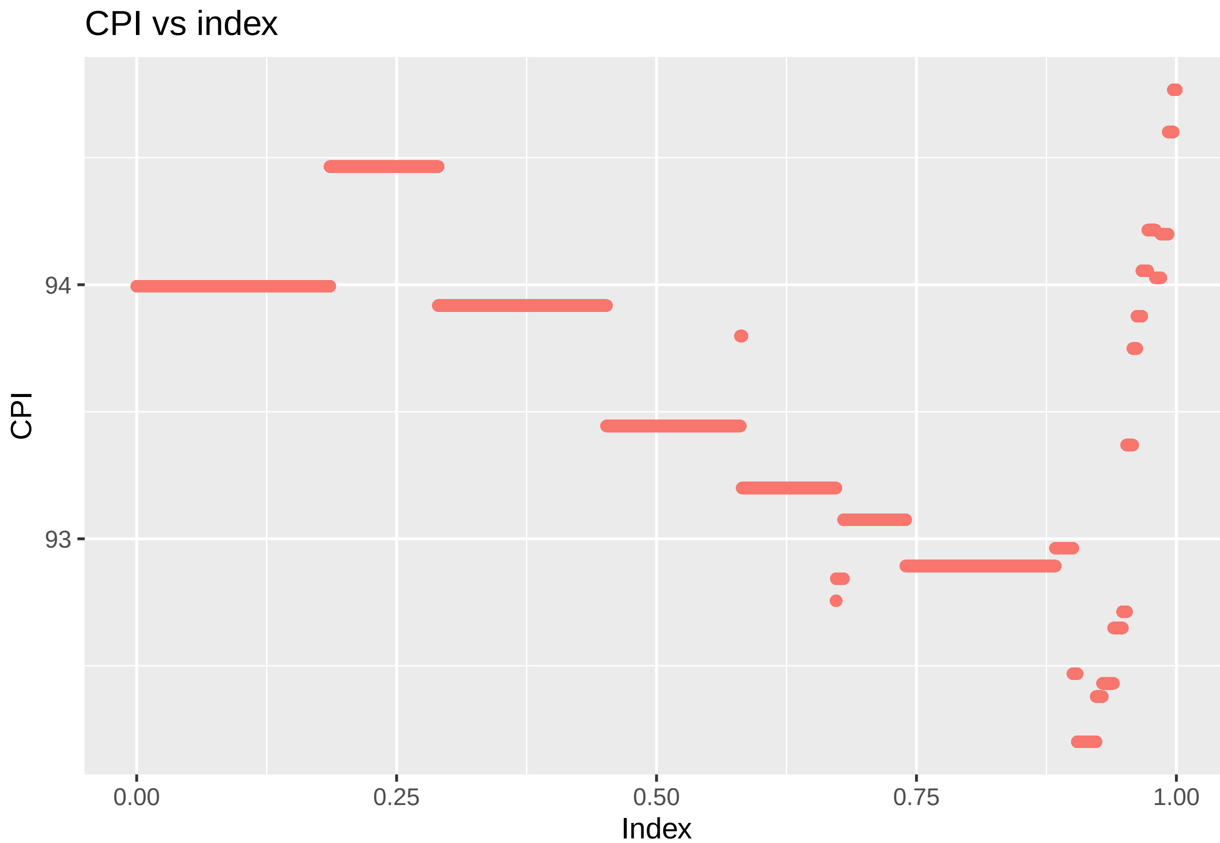
89.71
91.83
92.2
89.11
77.78
88.67
86.3
10.29
8.17
7.8
10.89
22.22
11.33
13.7
75
50
25
0
percentage
basic.4y
basic.6y
basic.9y
high.school
illiterate
professional.course
university.degree
education
y
no
yes

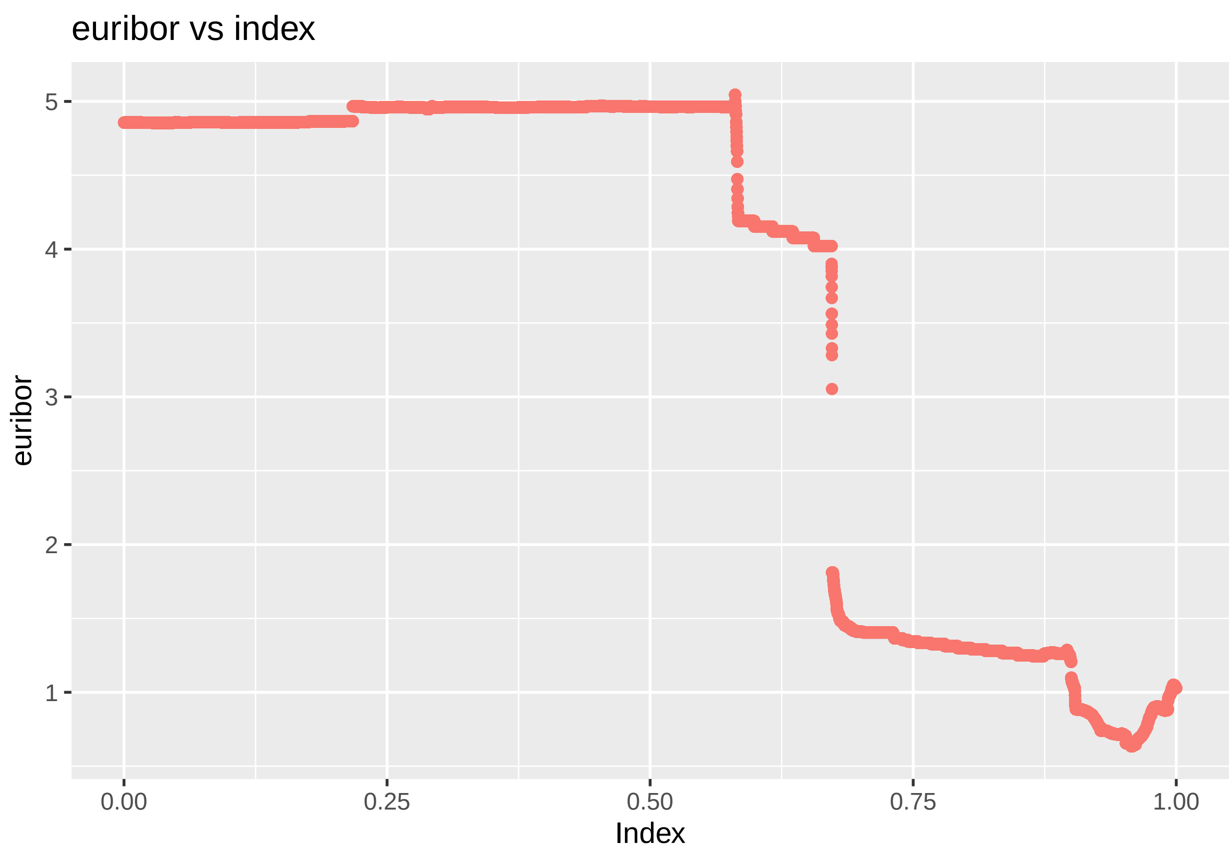

86.46
91.16
64.6
35.4
13.54
8.84
75
50
25
0
percentage
failure
nonexistent
success
poutcome
y
no
yes


percentage
100
75
50
25
0
25
50
75
100
age
y
no
yes



Now we plot the instability for the economic variables. We can easily there was harsh time then. We identify index as another phone call.


CPI vs index
94
93
CPI
0.00
0.25
0.50
0.75
1.00
Index



For economic variables we can make correlation matrix. It could be seen that many of the variables are correlated.

```
a1 <- subset(df, select = c("emp.var.rate","cons.price.idx", "cons.conf.idx", "euribor3m",
"nr.employed"))

corrplot(cor(a1), method = "number")
```

Models

In this chapter we consider different machine learning models. The most important score for us is AUC (area under curve). However we will also look at Accuracy and sensitivity (it may be crucial because we want to find a lot of clients but we want to reduce phones with negative result). AUC is a compromise between Accuracy and Sensitivity.

Preprocessing

To obtain only numerical values we use self written function that makes dummy variables. It is crucial to make logistic regression and neural network.

```
td <- df
df_dummy <- td %>%
  mutate(y = if_else(td$y == "yes",1,0)) %>%
  mutate(JobAdmin = if_else(td$job == "admin.",1,0)) %>%# work:
  mutate(JobBlue = if_else(td$job == "blue-collar",1,0)) %>%
  mutate(JobEntrep = if_else(td$job == "entrepreneur",1,0)) %>%
  mutate(JobHaus = if_else(td$job == "housemaid",1,0)) %>%
  mutate(JobManagment = if_else(td$job == "management",1,0)) %>%
  mutate(JobRetired = if_else(td$job == "retired",1,0)) %>%
  mutate(JobSelf = if_else(td$job == "self-employed",1,0)) %>%
  mutate(JobServices = if_else(td$job == "services",1,0)) %>%
  mutate(JobStudent = if_else(td$job == "student",1,0)) %>%
  mutate(JobTechnican = if_else(td$job == "technician",1,0)) %>%
  mutate(JobUnemployed = if_else(td$job == "unemployed",1,0)) %>%
```

```r
  mutate(MaritalDivorce = if_else(td$marital == "divorced", 1, 0)) %>%
  mutate(MaritalMarried = if_else(td$marital == "married", 1, 0)) %>%
  mutate(MaritalSingle = if_else(td$marital == "single", 1, 0)) %>%
  mutate(Edu4y = if_else(td$education == "basic.4y", 1, 0)) %>%#education:
  mutate(Edu6y = if_else(td$education == "basic.6y", 1, 0)) %>%
  mutate(Edu9y = if_else(td$education == "basic.9y", 1, 0)) %>%
  mutate(EduHS = if_else(td$education == "high.school", 1, 0)) %>%
  mutate(EduIlliterate = if_else(td$education == "illiterate", 1, 0)) %>%
  mutate(EduCourse = if_else(td$education == "professional.course", 1, 0)) %>%
  mutate(EduUniDegree = if_else(td$education == "university.degree", 1, 0)) %>%
  mutate(HousYes = if_else(td$housing == "yes", 1, 0)) %>% #house:
  mutate(HousNo = if_else(td$housing == "no", 1, 0)) %>%
  mutate(LoanYes = if_else(td$loan == "yes", 1, 0)) %>% #loan:
  mutate(LoanNo = if_else(td$loan == "no", 1, 0)) %>%
  mutate(ContactCellular = if_else(td$contact == "cellular", 1, 0)) %>% #contact:
  mutate(ContactTelephone = if_else(td$contact == "telephone", 1, 0)) %>%
  mutate(PrevFailure = if_else(td$poutcome == "failure", 1, 0)) %>%# poutcome
  mutate(PrevNone = if_else(td$poutcome == "nonexistent", 1, 0)) %>%
  mutate(PrevSuccess = if_else(td$poutcome == "success", 1, 0)) %>%
  mutate(Mon = if_else(td$day_of_week == "mon", 1, 0)) %>%
  mutate(Thu = if_else(td$day_of_week == "thu", 1, 0)) %>%
  mutate(Wed = if_else(td$day_of_week == "wed", 1, 0)) %>%
  mutate(Tue = if_else(td$day_of_week == "tue", 1, 0)) %>%
  mutate(Fri = if_else(td$day_of_week == "fri", 1, 0)) %>%
  dplyr::select(c(y, age, JobAdmin, JobBlue, JobEntrep, JobHaus, JobManagment, JobRetired,
                JobSelf,
                JobServices, JobStudent, JobTechnican, #JobUnemployed,
                MaritalDivorce, MaritalMarried, #MaritalSingle,
                Edu4y, Edu6y, Edu9y, EduHS, EduCourse, EduUniDegree,# EduIlliterate,
                HousYes,# HousNo,
                LoanYes,# LoanNo,
                ContactCellular, #ContactTelephone,
                Mon, Thu, Wed, Tue, #Fri
                campaign, PrevFailure, PrevSuccess,# PrevNone,
                emp.var.rate,
                cons.price.idx,
                cons.conf.idx,
                euribor3m,
                nr.employed
  ))
```

Making train and test sets.

```r
set.seed(1)
train_indices <- createDataPartition(df_dummy$y, p=.8, list = FALSE)
train_df <- df_dummy[train_indices, ]
test_df <- df_dummy[-train_indices, ]
```

MinMax nomalization of the sets.

```r
df.maxs <- apply(train_df, 2, max)
df.mins <- apply(train_df, 2, min)
# Rescale the train set:
train_df.sc <- as.data.frame(scale(train_df, center = df.mins,
                            scale = df.maxs - df.mins))
```

```
# Rescale the test set:
test_df.sc <- as.data.frame(scale(test_df, center = df.mins,
                                  scale = df.maxs - df.mins))
```

Making the target variable as factor for upsampling with regard to y.

```
train_df.sc$y <- as.factor(train_df.sc$y)
```

Because our data is imbalanced it is recommended to perform upsampling.

```
set.seed(1)
train_df.sc.up <- upSample(x = train_df.sc[, -1], y = train_df.sc$y, yname = "y")
table(train_df.sc.up$y)
```

```
##
##     0     1
## 27214 27214
```

Logistic regession

Firstly we check a full model. However it is easy to see it does not work well. So we will try to improve it by backward selection. We will skip a looking for model process and show a final model.

```
set.seed(1)
glm1 <- glm(y~.-1, data = train_df.sc.up, family = "binomial")
summary(glm1)
```

```
##
## Call:
## glm(formula = y ~ . - 1, family = "binomial", data = train_df.sc.up)
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## age              -0.204249   0.097529  -2.094 0.036239 *
## JobAdmin         -0.018215   0.063610  -0.286 0.774610
## JobBlue          -0.148460   0.065446  -2.268 0.023303 *
## JobEntrep        -0.070795   0.080485  -0.880 0.379075
## JobHaus          -0.031734   0.088853  -0.357 0.720975
## JobManagment     -0.004855   0.071307  -0.068 0.945721
## JobRetired        0.554779   0.078978   7.024 2.15e-12 ***
## JobSelf           0.041874   0.079271   0.528 0.597330
## JobServices      -0.163370   0.069570  -2.348 0.018860 *
## JobStudent        0.478102   0.089954   5.315 1.07e-07 ***
## JobTechnican      0.001400   0.065824   0.021 0.983026
## MaritalDivorce   -0.015662   0.037564  -0.417 0.676721
## MaritalMarried   -0.019417   0.025221  -0.770 0.441375
## Edu4y            -0.377249   0.136695  -2.760 0.005784 **
## Edu6y            -0.270536   0.138661  -1.951 0.051050 .
## Edu9y            -0.261343   0.133539  -1.957 0.050341 .
## EduHS            -0.235323   0.132189  -1.780 0.075044 .
## EduCourse        -0.149410   0.133806  -1.117 0.264158
## EduUniDegree     -0.113309   0.131864  -0.859 0.390181
## HousYes           0.002867   0.020000   0.143 0.886018
## LoanYes          -0.059726   0.027575  -2.166 0.030318 *
## ContactCellular   0.964583   0.029857  32.306  < 2e-16 ***
## Mon              -0.169393   0.031948  -5.302 1.14e-07 ***
```

```
## Thu              0.017183   0.031178   0.551 0.581555
## Wed              0.104453   0.031580   3.308 0.000941 ***
## Tue             -0.074173   0.032262  -2.299 0.021499 *
## campaign        -0.985151   0.187380  -5.258 1.46e-07 ***
## PrevFailure     -0.514297   0.032657 -15.748  < 2e-16 ***
## PrevSuccess      1.348005   0.060988  22.103  < 2e-16 ***
## emp.var.rate    -4.162449   0.174035 -23.917  < 2e-16 ***
## cons.price.idx   3.170668   0.131849  24.048  < 2e-16 ***
## cons.conf.idx    0.992869   0.073953  13.426  < 2e-16 ***
## euribor3m        0.757456   0.183245   4.134 3.57e-05 ***
## nr.employed     -0.300787   0.193229  -1.557 0.119558
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 75453  on 54428  degrees of freedom
## Residual deviance: 60239  on 54394  degrees of freedom
## AIC: 60307
##
## Number of Fisher Scoring iterations: 5
```

```r
set.seed(1)
glm14 <- glm(y~.-JobSelf-JobAdmin-age-JobManagment-Thu-HousYes-MaritalDivorce-LoanYes-
               nr.employed-MaritalMarried-Tue-JobHaus-JobEntrep-JobTechnican-1, data = train_df.sc.up,
family = "binomial")
summary(glm14)
```

```
##
## Call:
## glm(formula = y ~ . - JobSelf - JobAdmin - age - JobManagment -
##     Thu - HousYes - MaritalDivorce - LoanYes - nr.employed -
##     MaritalMarried - Tue - JobHaus - JobEntrep - JobTechnican -
##     1, family = "binomial", data = train_df.sc.up)
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## JobBlue         -0.12994    0.03250  -3.998 6.39e-05 ***
## JobRetired       0.50943    0.04863  10.476  < 2e-16 ***
## JobServices     -0.14750    0.03822  -3.859 0.000114 ***
## JobStudent       0.53546    0.06676   8.020 1.06e-15 ***
## Edu4y           -0.65894    0.06439 -10.234  < 2e-16 ***
## Edu6y           -0.53727    0.06979  -7.699 1.38e-14 ***
## Edu9y           -0.52153    0.06025  -8.656  < 2e-16 ***
## EduHS           -0.48978    0.05612  -8.727  < 2e-16 ***
## EduCourse       -0.39680    0.05818  -6.820 9.11e-12 ***
## EduUniDegree    -0.36330    0.05404  -6.722 1.79e-11 ***
## ContactCellular  0.97113    0.02941  33.023  < 2e-16 ***
## Mon             -0.15486    0.02575  -6.014 1.81e-09 ***
## Wed              0.12508    0.02534   4.937 7.94e-07 ***
## campaign        -0.99569    0.18674  -5.332 9.72e-08 ***
## PrevFailure     -0.51828    0.03259 -15.902  < 2e-16 ***
## PrevSuccess      1.34997    0.06087  22.177  < 2e-16 ***
## emp.var.rate    -4.21186    0.16943 -24.860  < 2e-16 ***
## cons.price.idx   3.30619    0.09397  35.182  < 2e-16 ***
```

```
## cons.conf.idx    1.05390    0.05312  19.839  < 2e-16 ***
## euribor3m        0.53765    0.12209   4.404 1.06e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 75453  on 54428  degrees of freedom
## Residual deviance: 60266  on 54408  degrees of freedom
## AIC: 60306
##
## Number of Fisher Scoring iterations: 5
```

Now we will take a look at scores for final model (glm14).

```
fitted.results <- predict(glm14, newdata=test_df.sc,type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)
accuracy2(fitted.results, test_df.sc$y)
```

```
## [1] 0.7878154
```

```
sensitivity(fitted.results, test_df.sc$y)
```

```
## [1] 0.6849315
```

```
CrossTable(test_df.sc$y, fitted.results,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  7649
##
##
##              | fitted.results
## test_df.sc$y |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |      5426 |      1347 |      6773 |
##              |     0.709 |     0.176 |           |
## -------------|-----------|-----------|-----------|
##            1 |       276 |       600 |       876 |
##              |     0.036 |     0.078 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      5702 |      1947 |      7649 |
## -------------|-----------|-----------|-----------|
##
##
```

```
roc_score_glm=roc(response = test_df.sc$y, predictor = fitted.results)
auc(roc_score_glm)
```

```
## Area under the curve: 0.743
```

We can see the results for logistic regression:

- $accuracy = 0.7878$
- $sensitivity = 0.6849$
- $AUC = 0.7431$

Neural network

To make a neural network faster we perform downsapling instead of (shown above) upsampling.

```
train_df.sc.down <- downSample(x = train_df.sc[, -1], y = train_df.sc$y, yname = "y")
table(train_df.sc.down$y)
```

```
##
##    0    1
## 3382 3382
```

Here we construct variable frm which will be useful to shorten neuralnet function.

```
nm <- names(train_df.sc.down)
frm <- as.formula(paste("y ~", paste(nm[!nm %in% "y"], collapse = " + ")))
print(frm)
```

```
## y ~ age + JobAdmin + JobBlue + JobEntrep + JobHaus + JobManagment +
##     JobRetired + JobSelf + JobServices + JobStudent + JobTechnican +
##     MaritalDivorce + MaritalMarried + Edu4y + Edu6y + Edu9y +
##     EduHS + EduCourse + EduUniDegree + HousYes + LoanYes + ContactCellular +
##     Mon + Thu + Wed + Tue + campaign + PrevFailure + PrevSuccess +
##     emp.var.rate + cons.price.idx + cons.conf.idx + euribor3m +
##     nr.employed
```

Very first idea is to take all our data and fit it in the model. With our limited processing resources we have to find a model that is both accurate and compile in relatively reasonable time. Even after taking downsampled set, due to the size of data and number of columns our algorithm do not always converge, so we increase number of steps and take simple model with 1 hidden neuron. Bigger number of hidden neurons in our case lead to lack of convergence, much increased computation time and sometimes worse results. In this case we know that our function is not linear and activation function tanh seems to shorten time we have to wait for instruction to compile. Increasing threshold seems like a solution to our problems with convergence and time, but lead to decrease in accuracy and precision.

```
set.seed(1)
nn1 <- neuralnet(frm, data = train_df.sc.down, hidden = 1, threshold = 0.01,
stepmax = 1e7, learningrate.factor = list(minus = 0.5, plus = 1.2),
act.fct = "tanh", linear.output = FALSE)

pr.nn1 <- compute(nn1,test_df.sc[,-1])

pr.nn1$score <- if_else(max.col(pr.nn1$net.result) == 2, 1, 0)
```

```
#accuracy
table(pr.nn1$score == test_df.sc$y)[2]/(sum(table(pr.nn1$score == test_df.sc$y)))
```

```
##      TRUE
## 0.8520068
```

```r
sensitivity(pr.nn1$score, test_df.sc$y)
```

```
## [1] 0.5958904
```

```r
CrossTable(test_df.sc$y, pr.nn1$score,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, prop.t = TRUE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  7649
##
##
##              | pr.nn1$score
## test_df.sc$y |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |      5995 |       778 |      6773 |
##              |     0.784 |     0.102 |           |
## -------------|-----------|-----------|-----------|
##            1 |       354 |       522 |       876 |
##              |     0.046 |     0.068 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      6349 |      1300 |      7649 |
## -------------|-----------|-----------|-----------|
##
##
```

```r
roc_score_nn1=roc(response = test_df.sc$y, predictor = pr.nn1$score)
auc(roc_score_nn1)
```

```
## Area under the curve: 0.7405
```

We can see the results for neural network:

- $accuracy = 0.8520$
- $sensitivity = 0.5959$
- $AUC = 0.7405$

Different idea: we only take data that seems reasonable and allow our construct to compute in reasonable amount of time and steps.

```r
set.seed(1)
nn2 <- neuralnet(y ~ age + PrevSuccess + cons.price.idx + emp.var.rate,
                 data = train_df.sc.down[,c("y", "age", "PrevSuccess",
                                            "cons.price.idx", "emp.var.rate")],
                 threshold = 0.05, hidden = c(5,2), linear.output = FALSE,
                 stepmax = 1e7)
                 #act.fct = "tanh", algorithm = "rprop+",
```

To take more complex model / bigger number of hidden neurons we decided to sacrifice our low threshold.

Default model "rprop+" seems to work as fast or faster than other algotirhms that give comparable results.

```
set.seed(1)
pr.nn2 <- compute(nn2, test_df.sc[,c("y", "age", "PrevSuccess",
                                     "cons.price.idx", "emp.var.rate")])

pr.nn2$score <- if_else(max.col(pr.nn2$net.result) == 2, 1, 0)

table(pr.nn2$score == test_df.sc$y)
```

```
##
## FALSE  TRUE
##  1289  6360
```

```
sum(table(pr.nn2$score == test_df.sc$y))
```

```
## [1] 7649
```

```
table(pr.nn2$score == test_df.sc$y)[2]/sum(table(pr.nn2$score == test_df.sc$y))
```

```
##      TRUE
## 0.8314812
```

```
sensitivity(pr.nn2$score, test_df.sc$y)
```

```
## [1] 0.6221461
```

```
CrossTable(test_df.sc$y, pr.nn2$score,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, prop.t = TRUE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  7649
##
##
##              | pr.nn2$score
## test_df.sc$y |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |      5815 |       958 |      6773 |
##              |     0.760 |     0.125 |           |
## -------------|-----------|-----------|-----------|
##            1 |       331 |       545 |       876 |
##              |     0.043 |     0.071 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      6146 |      1503 |      7649 |
## -------------|-----------|-----------|-----------|
##
##
```

```
roc_score_nn2=roc(response = test_df.sc$y, predictor = pr.nn2$score)
auc(roc_score_nn2)
```

```
## Area under the curve: 0.7404
```

We can see the results for neural network 2:

- $accuracy = 0.8315$
- $sensitivity = 0.6221$
- $AUC = 0.7404$

Decision tree

For decision tree and random forest we have to change data type to factors.

```
df$job <- as.factor(df$job)
df$marital <- as.factor(df$marital)
df$education <- as.factor(df$education)
df$housing <- as.factor(df$housing)
df$loan <- as.factor(df$loan)
df$contact <- as.factor(df$contact)
df$day_of_week <- as.factor(df$day_of_week)
df$poutcome <- as.factor(df$poutcome)
df$y <- as.factor(df$y)
```

We split data to train and test sets.

```
train_df_fct <- df[train_indices, ]
test_df_fct <- df[-train_indices, ]
```

Because our data is imbalanced it is recommended to perform upsampling (or downsampling that we made before).

```
set.seed(1)
train_up <- upSample(x = train_df_fct[, -ncol(train_df_fct)],
                     y = train_df_fct$y, yname = "y")
table(train_up$y)
```

```
##
##    no   yes
## 27214 27214
```

To easy check tree potential and importance of variables we use firstly C5.0 library instead of rpart. We make a tree and sum it up.

```
set.seed(1)
tree <- C5.0(train_up[-ncol(train_up)], train_up$y, trials =1)

C5imp(tree)
```

```
## Warning in (varStart + 1):length(treeDat): numerical expression has 2 elements:
## only the first used
```

```
##                Overall
## poutcome        100.00
## nr.employed     100.00
## campaign         86.67
## job              86.31
## euribor3m        83.77
## day_of_week      79.85
## cons.conf.idx    76.19
```

```
## education        72.88
## age              72.27
## cons.price.idx   64.42
## contact          57.95
## marital          44.82
## loan             36.46
## emp.var.rate     35.15
## housing          21.06
```

```r
tree_pred <- predict(tree, test_df_fct)
accuracy2(tree_pred, test_df_fct$y)
```

```
## [1] 0.8131782
```

```r
sensitivity2(tree_pred, test_df_fct$y)
```

```
## [1] 0.4737443
```

```r
CrossTable(test_df_fct$y, tree_pred,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  7649
##
##
##               | tree_pred
## test_df_fct$y |        no |       yes | Row Total |
## --------------|-----------|-----------|-----------|
##            no |      5805 |       968 |      6773 |
##               |     0.759 |     0.127 |           |
## --------------|-----------|-----------|-----------|
##           yes |       461 |       415 |       876 |
##               |     0.060 |     0.054 |           |
## --------------|-----------|-----------|-----------|
##  Column Total |      6266 |      1383 |      7649 |
## --------------|-----------|-----------|-----------|
##
##
```

```r
test_df_fct_ynum <- if_else(test_df_fct$y == "yes",1,0)
tree_pred_num <- if_else(tree_pred == "yes",1, 0)

roc_score=roc(response = test_df_fct_ynum, predictor = tree_pred_num)
auc(roc_score)
```

```
## Area under the curve: 0.6654
```

We can see results for the default C5.0 tree:

- $accuracy = 0.8132$

- $sensitivity = 0.4737$
- $AUC = 0.6654$

Random forest

Now we move to random forest model.

```
set.seed(1)
forest <- randomForest(formula = y~.,
                       data = train_up,
                       xtest = test_df_fct[, -ncol(test_df_fct)],
                       ytest = test_df_fct$y)

forest_pred <- forest$test$predicted

accuracy2(forest_pred, test_df_fct$y)
```

```
## [1] 0.8748856
```

```
confusionMatrix(data = forest_pred, reference = test_df_fct$y, positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  6262  446
##        yes  511  430
##
##                Accuracy : 0.8749
##                  95% CI : (0.8673, 0.8822)
##     No Information Rate : 0.8855
##     P-Value [Acc > NIR] : 0.99809
##
##                   Kappa : 0.4024
##
##  Mcnemar's Test P-Value : 0.03856
##
##             Sensitivity : 0.49087
##             Specificity : 0.92455
##          Pos Pred Value : 0.45696
##          Neg Pred Value : 0.93351
##              Prevalence : 0.11452
##          Detection Rate : 0.05622
##    Detection Prevalence : 0.12302
##       Balanced Accuracy : 0.70771
##
##        'Positive' Class : yes
##
```

```
test_df_fct_ynum <- if_else(test_df_fct$y == "yes",1,0)
forest_pred_num <- if_else(forest_pred == "yes",1, 0)
roc_score=roc(response = test_df_fct_ynum, predictor = forest_pred_num)
auc(roc_score)
```

```
## Area under the curve: 0.7077
```

We can see results for the default random forest:

- $accuracy = 0.8749$
- $sensitivity = 0.4909$
- $AUC = 0.7077$

We have quite good scores and the best accuracy so far. However very low sensitivity.

Therefore we will try to improve results for tree and forest by grid search.

Hyperparameter tuning - tree

We want to perform a 5-crossvalidation for the models with chosen parameters.

```
minsplit = c(10, 20, 40, 100)
cp = c(0.1, 0.01, 0.001, 0.0001)
maxdepth = c(10, 20, 30)

m <- length(minsplit)
n <- length(cp)
o <- length(maxdepth)

rp_cv_results <- as.data.frame(matrix(rep(0, m*n*o*6), nrow = m*n*o))
names(rp_cv_results) <- c("minsplit", "cp", "maxdepth", "Accuracy", "Sensitivity", "AUC")
```

```
folds_indices <- createFolds(train_df_fct$y, k = 5)
for (k in 1:m)
{
  for (l in 1:n)
  {
    for (q in 1:o)
    {
      set.seed(1)
      index <- (k-1)*n*o + (l-1)*o+q
      Accuracy <- 0
      Sensitivity <- 0
      AUC <- 0
      for (i in 1:5)
      {
        cv_indices <- c()
        for (j in 1:5)
        {
          if (j != i)
          {
            cv_indices <- c(cv_indices, unlist(folds_indices[j]))
          }
        }
        train_cv <- train_df_fct[cv_indices, ]
        test_cv <- train_df_fct[unlist(folds_indices[i]), ]
        train_cv_up <- upSample(x = train_cv[, -ncol(train_df_fct)],
                                y = train_cv$y, yname = "y")

        rpart_cv <- rpart(formula = y~.,
                          data = train_cv_up,
                          method = "class",
```

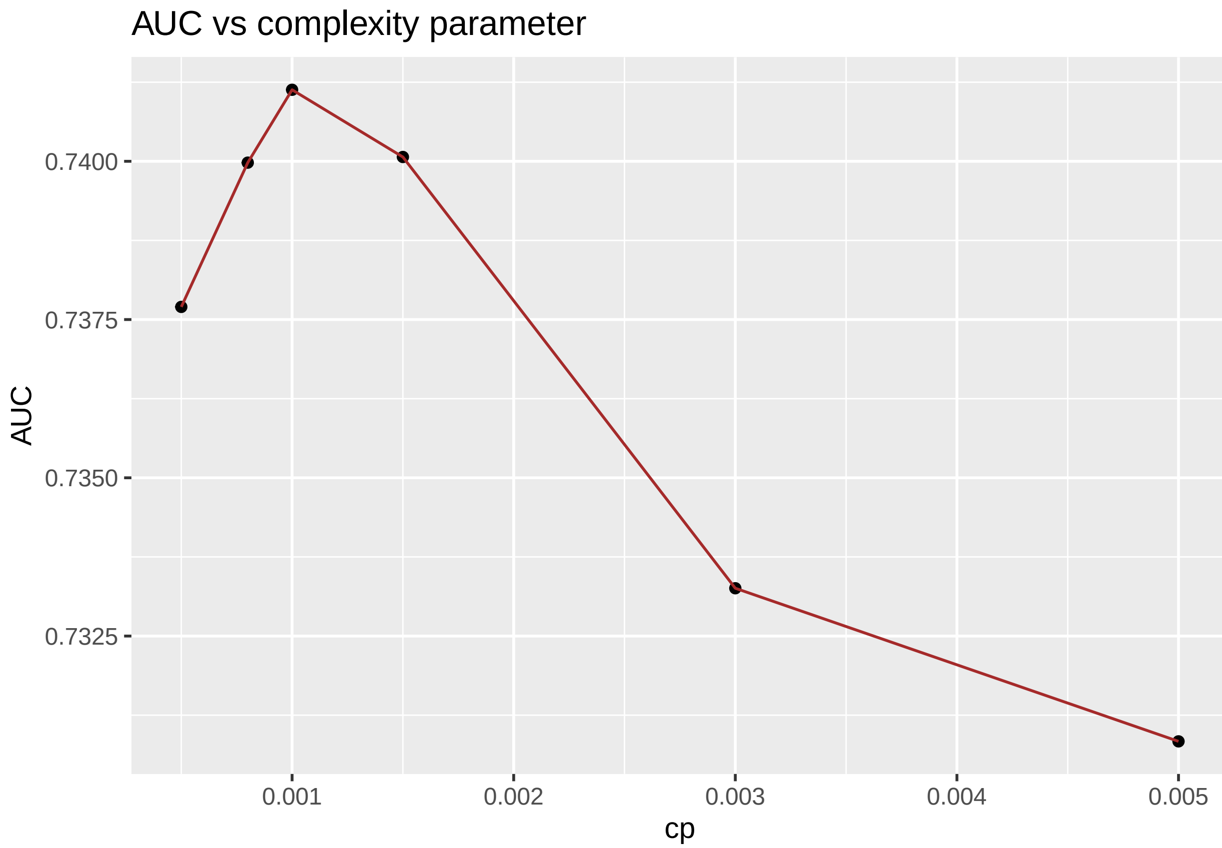
```r
                                   control = rpart.control(
                                     minsplit = minsplit[k],
                                     cp = cp[l],
                                     maxdepth = maxdepth[q]))

        rp_cv_pred <- predict(rpart_cv, test_cv, type = "class")
        Accuracy <- accuracy2(rp_cv_pred, test_cv$y) + Accuracy
        Sensitivity <- sensitivity2(rp_cv_pred, test_cv$y) + Sensitivity

        test_cv_ynum <- if_else(test_cv$y == "yes",1,0)
        rp_cv_pred_num <- if_else(rp_cv_pred == "yes",1, 0)
        roc_score=roc(response = test_cv_ynum,
                      predictor = rp_cv_pred_num)
        AUC <- auc(roc_score) + AUC
      }
      Accuracy <- Accuracy/5
      Sensitivity <- Sensitivity/5
      AUC <- AUC/5
      rp_cv_results$minsplit[index] <- minsplit[k]
      rp_cv_results$cp[index] <- cp[l]
      rp_cv_results$maxdepth[index] <- maxdepth[q]
      rp_cv_results$Accuracy[index] <- Accuracy
      rp_cv_results$Sensitivity[index] <- Sensitivity
      rp_cv_results$AUC[index] <- AUC
    }
  }
}
```

We conclude that the most important parameter is cp. Let's perform grid search for cp.

```r
minsplit = c(20)
cp = c(0.0005,0.0008, 0.001, 0.0015, 0.003, 0.005)
maxdepth = c(30)

m <- length(minsplit)
n <- length(cp)
o <- length(maxdepth)

rp_cv_results <- as.data.frame(matrix(rep(0, m*n*o*6), nrow = m*n*o))
names(rp_cv_results) <- c("minsplit", "cp", "maxdepth", "Accuracy", "Sensitivity", "AUC")

folds_indices <- createFolds(train_df_fct$y, k = 5)
for (k in 1:m)
{
  for (l in 1:n)
  {
    for (q in 1:o)
    {
      set.seed(1)
      index <- (k-1)*n*o + (l-1)*o+q
      Accuracy <- 0
      Sensitivity <- 0
      AUC <- 0
      for (i in 1:5)
```

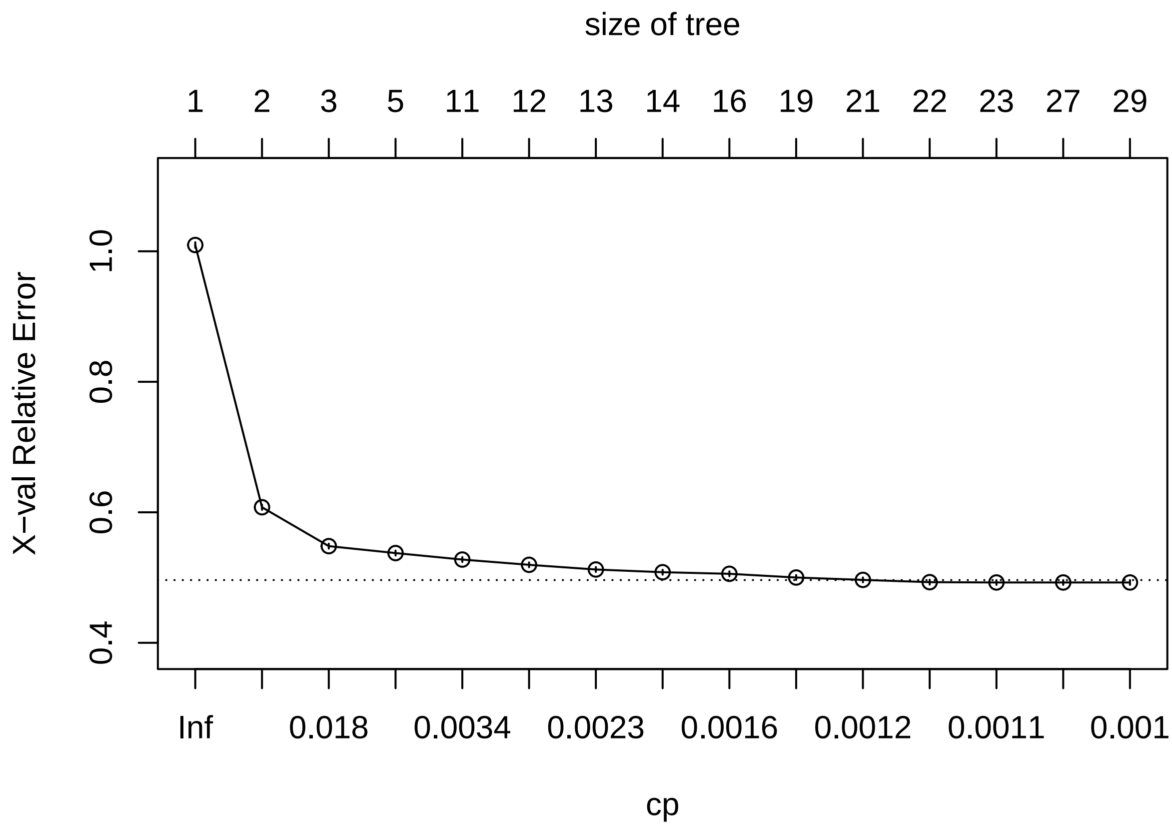
```r
      {
        cv_indices <- c()
        for (j in 1:5)
        {
          if (j != i)
          {
            cv_indices <- c(cv_indices, unlist(folds_indices[j]))
          }
        }
        train_cv <- train_df_fct[cv_indices, ]
        test_cv <- train_df_fct[unlist(folds_indices[i]), ]
        train_cv_up <- upSample(x = train_cv[, -ncol(train_df_fct)],
                                y = train_cv$y, yname = "y")

        rpart_cv <- rpart(formula = y~.,
                          data = train_cv_up,
                          method = "class",
                          control = rpart.control(
                            minsplit = minsplit[k],
                            cp = cp[l],
                            maxdepth = maxdepth[q]))

        rp_cv_pred <- predict(rpart_cv, test_cv, type = "class")
        Accuracy <- accuracy2(rp_cv_pred, test_cv$y) + Accuracy
        Sensitivity <- sensitivity2(rp_cv_pred, test_cv$y) + Sensitivity

        test_cv_ynum <- if_else(test_cv$y == "yes",1,0)
        rp_cv_pred_num <- if_else(rp_cv_pred == "yes",1, 0)
        roc_score=roc(response = test_cv_ynum,
                      predictor = rp_cv_pred_num)
        AUC <- auc(roc_score) + AUC
      }
      Accuracy <- Accuracy/5
      Sensitivity <- Sensitivity/5
      AUC <- AUC/5
      rp_cv_results$minsplit[index] <- minsplit[k]
      rp_cv_results$cp[index] <- cp[l]
      rp_cv_results$maxdepth[index] <- maxdepth[q]
      rp_cv_results$Accuracy[index] <- Accuracy
      rp_cv_results$Sensitivity[index] <- Sensitivity
      rp_cv_results$AUC[index] <- AUC
    }
  }
}
```

```r
len <- length(rp_cv_results$AUC)
ggplot(data = rp_cv_results, mapping = aes(x = cp, y = AUC)) +
  geom_point() +
  geom_line(colour = "brown")+
  labs(x = "cp",
       y = "AUC",
       title = "AUC vs complexity parameter")
```



For parameters proposed above ($cp = 0.001, minsplit = 20, maxdepth = 30$) we make a tree model.

```
set.seed(1)
tree_rp <-rpart(y~., train_up, method = "class",
                control = rpart.control(cp = 0.001, minsplit = 20))
rp_predict <- predict(tree_rp, test_df_fct, type = "class")
plotcp(tree_rp)
```

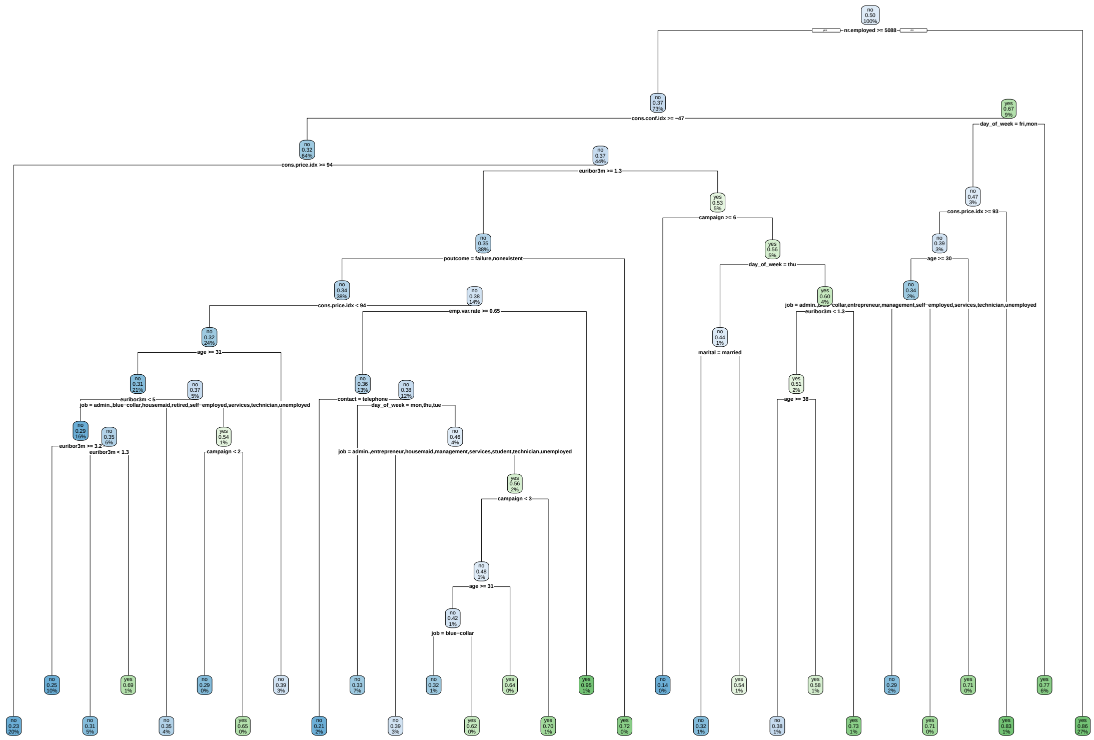



```r
tree_rp$variable.importance
```

```
##     nr.employed       euribor3m   cons.conf.idx    emp.var.rate cons.price.idx
##     5566.458261     5310.722160     4225.316206     3126.392195    3036.263282
##        poutcome     day_of_week         contact             age            job
##     1642.125845      295.787664      262.079259      201.567902     126.898315
##        campaign       education         marital            loan
##      112.327104       31.814713       17.441862        1.028272
```

```r
rpart.plot::rpart.plot(tree_rp)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```r
test_df_fct_ynum <- if_else(test_df_fct$y == "yes",1,0)
tree_pred_num <- if_else(rp_predict == "yes",1, 0)

roc_score=roc(response = test_df_fct_ynum, predictor = tree_pred_num)
auc(roc_score)
```

```
## Area under the curve: 0.7481
```

```r
accuracy2(rp_predict, test_df_fct$y)
```

```
## [1] 0.826644
```

```r
confusionMatrix(data = rp_predict, reference = test_df_fct$y, positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  5757  310
##        yes 1016  566
##
##                Accuracy : 0.8266
##                  95% CI : (0.818, 0.8351)
##     No Information Rate : 0.8855
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3673
##
```

```
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6461
##             Specificity : 0.8500
##          Pos Pred Value : 0.3578
##          Neg Pred Value : 0.9489
##              Prevalence : 0.1145
##          Detection Rate : 0.0740
##    Detection Prevalence : 0.2068
##       Balanced Accuracy : 0.7481
##
##        'Positive' Class : yes
##
```

We can see results for the tuned tree:

- $accuracy = 0.8266$
- $sensitivity = 0.6461$
- $AUC = 0.7481$

Hyperparameter tuning - forest

Now we will try to improve random forest.

```r
set.seed(1)
folds_indices <- createFolds(train_df$y, k = 5)

mtry = c(2,3,4,5,7,10)
ntree = c(30, 50, 80)
m = length(mtry)
n = length(ntree)

cv_results_forest <- as.data.frame(matrix(rep(0, m*n*5), nrow = m*n))
names(cv_results_forest) <- c("mtry", "ntree", "Accuracy", "Sensitivity", "AUC")
```

```r
for (k in 1:m)
{
  for (l in 1:n)
  {
    set.seed(1)
    index <- (k-1)*n+l
    Accuracy <- 0
    Sensitivity <- 0
    AUC <- 0
    for (i in 1:5)
    {
      cv_indices <- c()
      for (j in 1:5)
      {
        if (j != i)
        {
          cv_indices <- c(cv_indices, unlist(folds_indices[j]))
        }
      }
      train_cv <- train_df_fct[cv_indices, ]
```

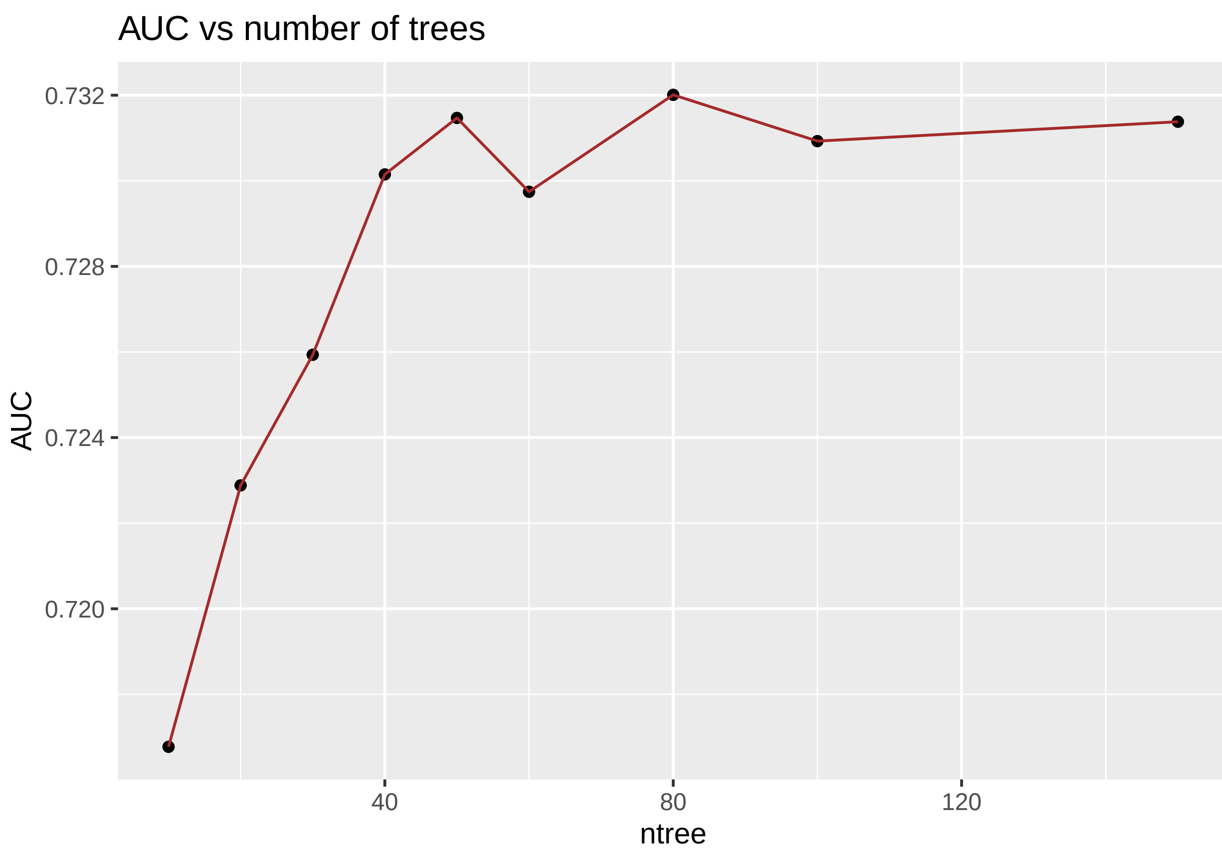
```
      test_cv <- train_df_fct[unlist(folds_indices[i]), ]
      train_cv_up <- upSample(x = train_cv[, -ncol(train_df_fct)],
                            y = train_cv$y, yname = "y")

      forest_cv <- randomForest(formula = y~.,
                                data = train_cv_up,
                                xtest = test_cv[, -ncol(test_cv)],
                                ytest = test_cv$y,
                                mtry = mtry[k],
                                ntree = ntree[l])
      forest_cv_pred <- forest_cv$test$predicted
      Accuracy <- accuracy2(forest_cv_pred, test_cv$y) + Accuracy
      Sensitivity <- sensitivity2(forest_cv_pred, test_cv$y) +       Sensitivity

        test_cv_ynum <- if_else(test_cv$y == "yes",1,0)
        forest_cv_pred_num <- if_else(forest_cv_pred == "yes",1, 0)
        roc_score=roc(response = test_cv_ynum,
                      predictor = forest_cv_pred_num)
        AUC <- auc(roc_score) + AUC
    }
    Accuracy <- Accuracy/5
    Sensitivity <- Sensitivity/5
    AUC <- AUC/5
    cv_results_forest$mtry[index] <- mtry[k]
    cv_results_forest$ntree[index] <- ntree[l]
    cv_results_forest$Accuracy[index] <- Accuracy
    cv_results_forest$Sensitivity[index] <- Sensitivity
    cv_results_forest$AUC[index] <- AUC
  }
}
```

We conclude that $mtry = 2$ is the best for various number of trees. Now we look for optimal ntree parameter.

```
set.seed(1)
folds_indices <- createFolds(train_df$y, k = 5)

mtry = c(2)
ntree = c(10,20,30,40,50,60,80,100,150)
m = length(mtry)
n = length(ntree)

cv_results_forest <- as.data.frame(matrix(rep(0, m*n*5), nrow = m*n))
names(cv_results_forest) <- c("mtry", "ntree", "Accuracy", "Sensitivity", "AUC")

for (k in 1:m)
{
  for (l in 1:n)
  {
    set.seed(1)
    index <- (k-1)*n+l
    Accuracy <- 0
    Sensitivity <- 0
    AUC <- 0
    for (i in 1:5)
```

```
    {
      cv_indices <- c()
      for (j in 1:5)
      {
        if (j != i)
        {
          cv_indices <- c(cv_indices, unlist(folds_indices[j]))
        }
      }
      train_cv <- train_df_fct[cv_indices, ]
      test_cv <- train_df_fct[unlist(folds_indices[i]), ]
      train_cv_up <- upSample(x = train_cv[, -ncol(train_df_fct)],
                              y = train_cv$y, yname = "y")

      forest_cv <- randomForest(formula = y~.,
                                data = train_cv_up,
                                xtest = test_cv[, -ncol(test_cv)],
                                ytest = test_cv$y,
                                mtry = mtry[k],
                                ntree = ntree[l])
      forest_cv_pred <- forest_cv$test$predicted
      Accuracy <- accuracy2(forest_cv_pred, test_cv$y) + Accuracy
      Sensitivity <- sensitivity2(forest_cv_pred, test_cv$y) +       Sensitivity

        test_cv_ynum <- if_else(test_cv$y == "yes",1,0)
        forest_cv_pred_num <- if_else(forest_cv_pred == "yes",1, 0)
        roc_score=roc(response = test_cv_ynum,
                      predictor = forest_cv_pred_num)
        AUC <- auc(roc_score) + AUC
    }
    Accuracy <- Accuracy/5
    Sensitivity <- Sensitivity/5
    AUC <- AUC/5
    cv_results_forest$mtry[index] <- mtry[k]
    cv_results_forest$ntree[index] <- ntree[l]
    cv_results_forest$Accuracy[index] <- Accuracy
    cv_results_forest$Sensitivity[index] <- Sensitivity
    cv_results_forest$AUC[index] <- AUC
  }
}
```

```
len <- length(cv_results_forest$AUC)
ggplot(data = cv_results_forest, mapping = aes(x = ntree, y = AUC)) +
  geom_point() +
  geom_line(colour = "brown")+
  labs(x = "ntree",
       y = "AUC",
       title = "AUC vs number of trees")
```



We make a random forest for choosen parameters $(ntree = 80, mtry = 2)$.

```
set.seed(1)
forest <- randomForest(formula = y~.,
                       data = train_up,
                       xtest = test_df_fct[, -ncol(test_df_fct)],
                       ytest = test_df_fct$y,
                       ntree = 80,
                       mtry = 2)

forest_pred <- forest$test$predicted

accuracy2(forest_pred, test_df_fct  $y)
```

```
## [1] 0.8695254
```

```
confusionMatrix(data = forest_pred, reference = test_df_fct$y, positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  6157  382
##        yes  616  494
##
##                Accuracy : 0.8695
##                  95% CI : (0.8618, 0.877)
##     No Information Rate : 0.8855
```

```
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.4237
##
##  Mcnemar's Test P-Value : 1.637e-13
##
##             Sensitivity : 0.56393
##             Specificity : 0.90905
##          Pos Pred Value : 0.44505
##          Neg Pred Value : 0.94158
##              Prevalence : 0.11452
##          Detection Rate : 0.06458
##    Detection Prevalence : 0.14512
##       Balanced Accuracy : 0.73649
##
##        'Positive' Class : yes
##
```

```
test_df_fct_ynum <- if_else(test_df_fct$y == "yes",1,0)
forest_pred_num <- if_else(forest_pred == "yes",1, 0)

roc_score_forest=roc(response = test_df_fct_ynum, predictor = forest_pred_num)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
auc(roc_score_forest)
```

```
## Area under the curve: 0.7365
```

We can see results for the tuned tree:

- $accuracy = 0.8695$
- $sensitivity = 0.5639$
- $AUC = 0.7365$

Conclusions

We have presented a few models and different ways to handle the problem. Now we will sum it up. We have to think what is more important for us. Do we want to reduce marketing costs and make less phone calls but do not find many possible clients. We may also make more but then we risk contact with people that do not want to take part in deposit program. As we state at the very beginnig our mains score is AUC.

Let's remind what values we consider:

- Logistic regression - for model with significant variables.
- Neural network - for first (full) model, for second try (when we use only four variables).
- Decision tree - for tree after hyperparameter tuning.
- Random forest - for forest after hyperparameter tuning.

Score	Logistic regression	Neural network 1	Neural network 2	Decision tree	Random forest
Accuracy	78.78%	85.20%	83.15%	82.66%	86.95%

Score	Logistic regression	Neural network 1	Neural network 2	Decision tree	Random forest
Sensitivity	68.49%	59.59%	62.21%	64.61%	56.39%
AUC	74.31%	74.05%	74.04%	74.81%	73.65%

It could be seen that all models could be valuable. Therefore the decision is not so simple. After thorough analysis, we have concluded that a decision tree is the most suitable model for our dataset.