HarvardX: PH125.9x Data Science
Bank Marketing Dataset

Luis Sànchez

June 2020

Contents

1 Overview **1**

2 Background **2**

3 Modelling **5**
3.1 Decision Tree 6
3.2 Random Forest 10
3.3 XGBoost 14

4 Results **18**

5 Conclusions **18**

6 Appendix **19**

1 Overview

The Center for Machine Learning and Intelligent Systems, of the Bren School of Information and Computer Science of the University of California, Irvine maintains a large repository of datasets for machine learning projects. This repository contains datasets for regression, classification, clustering and other. The databases for this project was obtained there. You can click here, or visit their url https://archive.ics.uci.edu/ml/datasets.php.

This project is related to the direct marketing campaigns of a Portuguese banking institution, which conducted marketing campaigns based on phone calls. Often, more than one contact to the same client was required in order to determine if the product being sold (bank deposits) would be a 'yes' or a 'no' from the client. The set is here, https://archive.ics.uci.edu/ml/datasets/Bank+Marketing and contains data from 2008 to 2013, including the period of the 2008 financial crisis.

There are four datasets in the zip file available in the site:

1) bank-additional-full.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010).
2) bank-additional.csv with 10% of the examples (4119), randomly selected from 1), and 20 inputs.
3) bank-full.csv with all examples and 17 inputs, ordered by date (older version of this dataset with less inputs).
4) bank.csv with 10% of the examples and 17 inputs, randomly selected from 3 (older version of this dataset with less inputs).

The smallest datasets are provided to test more computationally demanding machine learning algorithms. However, for this project, I will work with the bank-additional-full.csv, which as mentioned above, has 41,188 examples and 20 inputs.

The data includes bank's client data, data related to current and/or previous marketing campaigns,and socioeconomic features (such as interest rates, local consumer price indices, etc.)

The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

2 Background

In a paper written by the original researchers of this data, they stated that "marketing selling campaigns constitute a typical strategy to enhance business. Companies use direct marketing when targeting segments of customers by contacting them to meet a specific goal. Centralizing customer remote interaction in a contact center eases operational management of campaigns"

The objective of this project is to measure to what extent machine learning algorithms can select the clients that are more likely to subscribe to a financial product offered. This can be used to better plan expenditures in marketing, thus increasing the bank's profitability.

The features in consideration are the following:

1) **age**
2) **job**
3) **marital status**
4) **education**
5) **default**: has credit in default?
6) **housing**: has housing loan?
7) **loan**: has personal loan?
8) **contact**: contact communication type
9) **month**: last contact month of year
10) **dayofweek**: last contact day of the week

11) **duration**: last contact duration, in seconds. Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no').
12) **campaign**: number of contacts performed during this campaign and for this client
13) **pdays**: number of days that passed by after the client was last contacted from a previous campaign
14) **previous**: number of contacts performed before this campaign and for this client (numeric)
15) **poutcome**: outcome of the previous marketing campaign
16) **emp.var.rate**: employment variation rate - quarterly indicator
17) **cons.price.idx**: consumer price index - monthly indicator
18) **cons.conf.idx**: consumer confidence index - monthly indicator
19) **euribor3m**: euribor 3 month rate - daily indicator
20) **nr.employed**: number of employees - quarterly indicator
21) **y**: has the client subscribed a term deposit? (binary: 'yes','no')

Since this is a classical classification model, the main metric I will use for it is the area under the curve (AUC) for receiver operating characteristic (ROC) metric, https://en.wikipedia.org/wiki/Receiver_operating_characteristic . This is a graphical plot that illustrates the diagnostic ability of a binary classifier system (in our case a "yes" or a "no" to offers to buy the product offered by the bank). Under this metric, I will test:

1) A Decision Tree classifier
2) A Random Forest classifier
3) An XGBoost classifier

We will start with installing the relevant packages, and downloading and inspecting the data:

```
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(lubridate))
  install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(gridExtra))
  install.packages("gridExtra", repos = "http://cran.us.r-project.org")
if(!require(knitr))
  install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(kableExtra))
  install.packages("kableExtra", repos = "http://cran.us.r-project.org")
if(!require(ggplot2))
  install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(e1071))
  install.packages("e1071", repos = "http://cran.us.r-project.org")
if(!require(rpart))
  install.packages("rpart", repos = "http://cran.us.r-project.org")
if(!require(rpart.plot))
  install.packages("rpart.plot", repos = "http://cran.us.r-project.org")
if(!require(rattle))
  install.packages("rattle", repos = "http://cran.us.r-project.org")
if(!require(randomForest))
  install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(caTools))
  install.packages("caTools", repos = "http://cran.us.r-project.org")
if(!require(descr))
```

```
  install.packages("descr", repos = "http://cran.us.r-project.org")
if(!require(pROC))
  install.packages("pROC", repos = "http://cran.us.r-project.org")
if(!require(Matrix))
  install.packages("Matrix", repos = "http://cran.us.r-project.org")
if(!require(xgboost))
  install.packages("xgboot", repos = "http://cran.us.r-project.org")

# here I download the file
wd = getwd()
dl <- tempfile(tmpdir = wd)
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip"
download.file(url, dl)
unzip(dl, junkpaths = TRUE)
data <- read.table("bank-additional-full.csv",
                   header = T,
                   sep = ";")
unlink(dl)
```

Below are a few samples of out data samples, which their corresponding values.

age	job	marital	education	default	housing	loan	contact
56	housemaid	married	basic.4y	no	no	no	telephone
57	services	married	high.school	unknown	no	no	telephone
37	services	married	high.school	no	yes	no	telephone
40	admin.	married	basic.6y	no	no	no	telephone
56	services	married	high.school	no	no	yes	telephone

month	day_of_week	duration	campaign	pdays	previous	poutcome
may	mon	261	1	999	0	nonexistent
may	mon	149	1	999	0	nonexistent
may	mon	226	1	999	0	nonexistent
may	mon	151	1	999	0	nonexistent
may	mon	307	1	999	0	nonexistent

emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
1.1	93.994	-36.4	4.857	5191	no
1.1	93.994	-36.4	4.857	5191	no
1.1	93.994	-36.4	4.857	5191	no
1.1	93.994	-36.4	4.857	5191	no
1.1	93.994	-36.4	4.857	5191	no

The types of data (categorical numerical, etc.) as well as missing values (if any) can be seen below:

```
sapply(data, class)
```

```
##           age           job       marital     education       default
```

```
##      "integer"       "factor"       "factor"       "factor"       "factor"
##        housing           loan        contact          month    day_of_week
##       "factor"       "factor"       "factor"       "factor"       "factor"
##       duration       campaign          pdays       previous       poutcome
##      "integer"      "integer"      "integer"      "integer"       "factor"
##   emp.var.rate cons.price.idx  cons.conf.idx      euribor3m    nr.employed
##      "numeric"      "numeric"      "numeric"      "numeric"      "numeric"
##              y
##       "factor"
```

```
sum(is.na(data))
```

```
## [1] 0
```

The distribution of the data in the features can be seen in the Appendix.

3 Modelling

As mentioned before, I will create 3 models:

1) A Decision Tree classifier: Here I will attempt to graphically show the features that are more relevant in this dataset in terms of explanatory power for the yes or no decision of customers, in an out of sample set. For more information about Decision Trees, Wikipedia has a good entry in this url: https://en.wikipedia.org/wiki/Decision_tree

2) A Random Forest classifier: Similarly, I will also attempt to show feature importance and the model's ROC. Here is Wikipedia's entry https://en.wikipedia.org/wiki/Random_forest for a Random Forest classifier.

3) An XGBoost classifier: the XGBoost algorithm is a machine learning algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. It is basically an ensemble of decision trees which are fit sequentially, and each new tree makes up for the errors of the previously existing set of trees, trying to correct the residual errors of the last version of the model.

To test our models, we will split the dataset in 75% for training and 25% for testing, and create a tabular presentation of out data. Again, our metric to judge the accuracy of the classification model will be the ROC curve explained before.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index <- createDataPartition(y = data$y, times = 1, p = 0.25, list = FALSE)
train <- data[-test_index,]
test <- data[test_index,]
```

3.1 Decision Tree

Below is the implementation of the Decision Tree and its AUC-ROC curve.

```
# setting up the decision tree
train_dt<-rpart(y ~ ., train , method = 'class')

# predictions and probbailities
predictions <- predict(train_dt, test , type = "class")
probabilities <- predict(train_dt, test , type = "prob")

# metrics
CrossTable(test$y, predictions,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual', 'predicted'))
```

```
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
## ================================
##           predicted
## actual       no     yes   Total
## --------------------------------
## no         8787     350    9137
##           0.853   0.034
## --------------------------------
## yes         542     618    1160
##           0.053   0.060
## --------------------------------
## Total      9329     968   10297
## ================================
```

```
# AUCROC
data_roc <-roc(test$y, probabilities[,2])
```

```
## Setting levels: control = no, case = yes
```

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

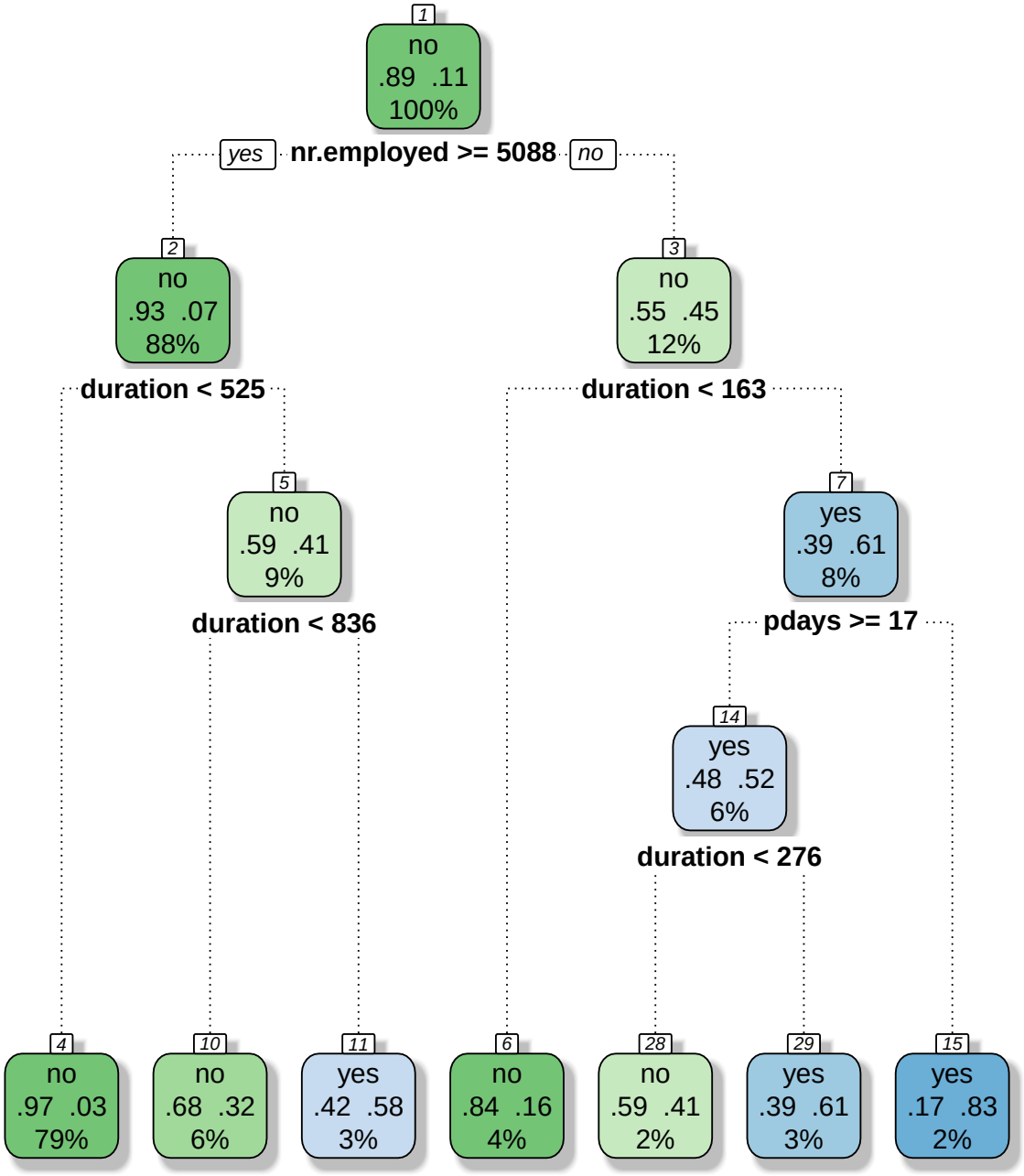
```
print(data_roc)
```

```
##
## Call:
## roc.default(response = test$y, predictor = probabilities[, 2])
```

```
##
## Data: probabilities[, 2] in 9137 controls (test$y no) < 1160 cases (test$y yes).
## Area under the curve: 0.8741
```

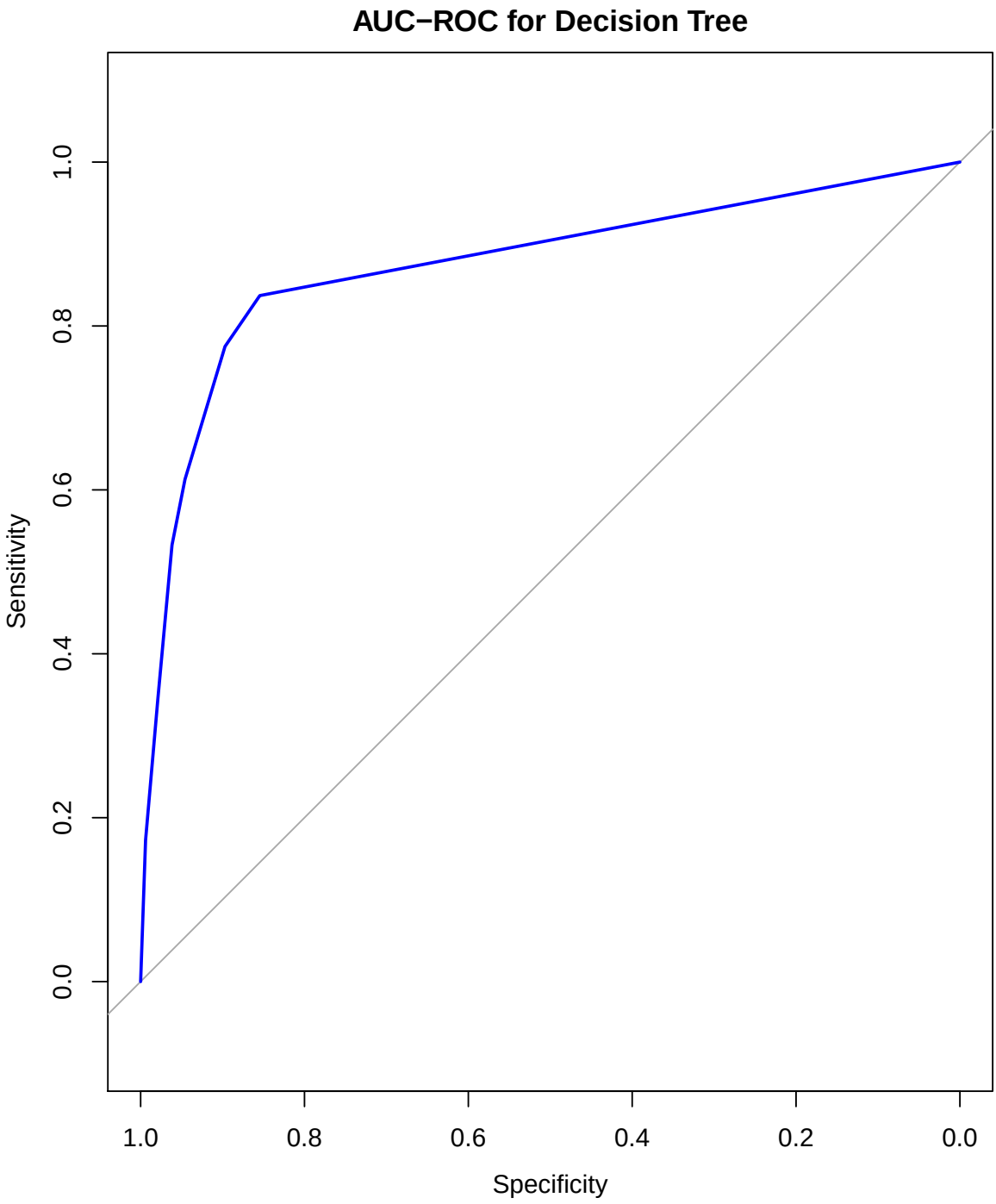
In the next page we can see that the Decision Tree achieved a 87.41% AUC-ROC in the test set. Let's visualize the actual decision tree and its ROC curve.

```
# par(mfrow=c(2,1))
fancyRpartPlot(train_dt , digits=2, caption="Decision Tree")
```



Decision Tree

```
plot(data_roc, main="AUC-ROC for Decision Tree",col="blue")
```


AUC−ROC for Decision Tree
Sensitivity
1.0
0.8
0.6
0.4
0.2
0.0
1.0
0.8
0.6
0.4
0.2
0.0
Specificity

3.2 Random Forest

Below is the implementation of the Random Forest algorithm and its AUC-ROC curve.

```
# setting up the Random Forest

rf_classifier = randomForest(y ~ ., train,
                             ntree=200,
                             mtry=3,
                             importance=TRUE)

# predict
rf_pred <- predict( rf_classifier,
                    test,
                    type = "class")
rf_prob <- predict( rf_classifier,
                    test,
                    type = "prob")

# Cross table validation for CART
CrossTable(test$y,
           rf_pred,
           prop.chisq = FALSE,
           prop.c = FALSE,
           prop.r = FALSE,
           dnn = c('actual default', 'predicted default'))
```

```
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
## ======================================
##                  predicted default
## actual default       no     yes   Total
## --------------------------------------
## no                 8840     297    9137
##                   0.859   0.029
## --------------------------------------
## yes                 574     586    1160
##                   0.056   0.057
## --------------------------------------
## Total              9414     883   10297
## ======================================
```

```r
# ROC
rf_roc <-roc(test$y,
             rf_prob[,2])
```

```
## Setting levels: control = no, case = yes
```

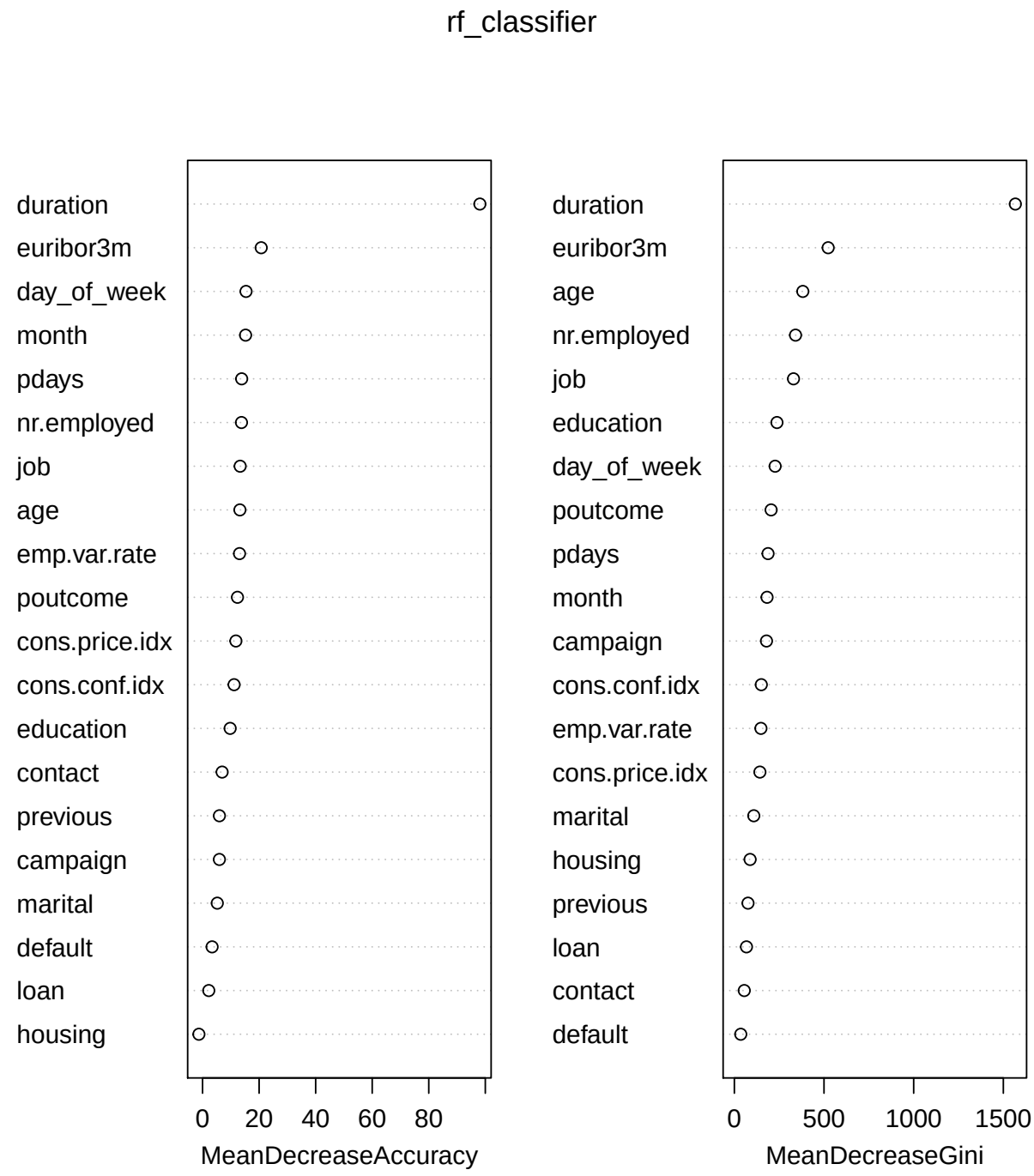
```
## Setting direction: controls < cases
```

```r
print(rf_roc)
```

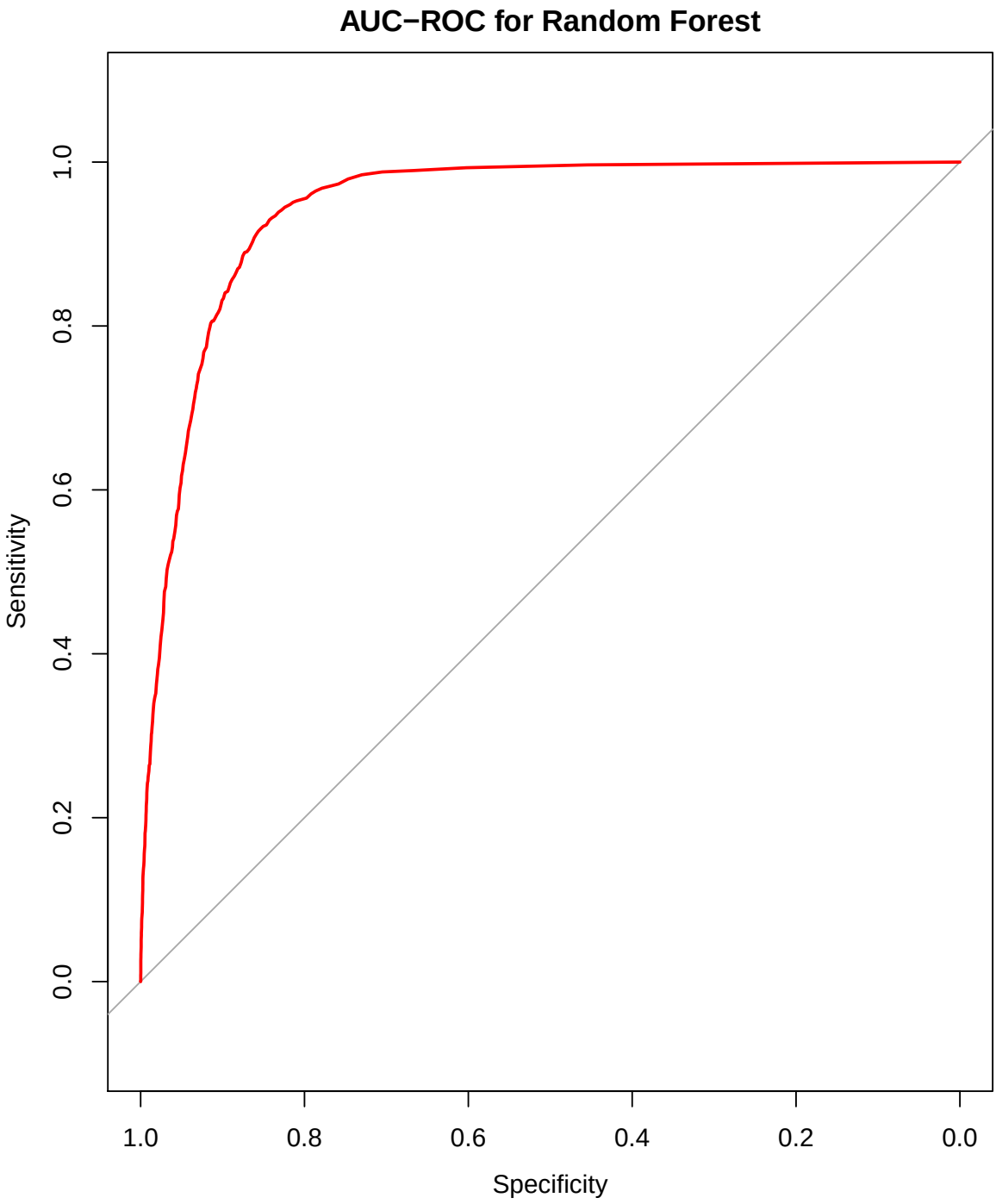
```
##
## Call:
## roc.default(response = test$y, predictor = rf_prob[, 2])
##
## Data: rf_prob[, 2] in 9137 controls (test$y no) < 1160 cases (test$y yes).
## Area under the curve: 0.9435
```

Here can see that the Random Forest achieved a 94.35% ROC in the test set. The feature importance of the model and its AUC-ROC curve follows in the next page.

```r
# par(mfrow=c(2,1))
varImpPlot(rf_classifier)
```




```
plot(rf_roc,
     main="AUC-ROC for Random Forest",
     percent=TRUE,
     col="red")
```


AUC−ROC for Random Forest
Sensitivity
1.0
0.8
0.6
0.4
0.2
0.0
1.0
0.8
0.6
0.4
0.2
0.0
Specificity

3.3 XGBoost

Below is the implementation of the extreme gradient boosting algorithm and its AUC-ROC curve. In setting this up, we need to be particularly careful, since XGBoost only accepts numerical values.

Therefore, we need to transform all the categorical values into "dummy" columns. For example, a categorical column labeled "fruit" that contains "apples", "oranges" and "pears" as elements; gets transformed into spare vectors, for example, "fruit_apple", "fruit_oranges", and "fruit_pears"; each one containing "1" or "0" to indicate the presence or absence of that feature.

Needless to say, in some datasets, this might represent a computational challenge, since it is possible to expand the dimension of our dataset a great deal.

```
sparse.matrix.train= sparse.model.matrix(y~.-1, data = train)
# creates sparse matrix from the train set
output.vector.train <- as.numeric(as.factor(train$y))-1
# transforms y into "1" and "0", the values OK with xgboost
sparse.matrix.test=  sparse.model.matrix(y~.-1, data = test)
# creates sparse matrix from the test set
output.vector.test <- as.numeric(as.factor(test$y))-1
# transforms y into "1" and "0", the values OK with xgboost
```

The model was set with evaluation metric = 'auc', to maximize AUC-ROC score, and objective = "binary:logistic", for binary classification.

```
bst= xgboost(data = sparse.matrix.train, # train sparse matrix
             label = output.vector.train, # output vector to be predicted
             eval.metric = 'auc', # model maximizes auc
             objective = "binary:logistic", # clasification
             max.depth = 3,
             nround = 250,
             verbose = 0) #dont print out results of trainning to PDF

importance <- xgb.importance(feature_names = colnames(sparse.matrix.train),
                             model = bst)
```

```
probabilities.xgb <- predict(bst, sparse.matrix.test)
predictions.xgb <- as.numeric(probabilities.xgb > 0.5) # transform prediction to labels

# metrics
CrossTable(output.vector.test,
           predictions.xgb,
           prop.chisq = FALSE,
           prop.c = FALSE,
           prop.r = FALSE,
```

```
           dnn = c('actual', 'predicted'))
```

```
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
## ================================
##           predicted
## actual        0       1   Total
## --------------------------------
## 0          8774     363    9137
##           0.852   0.035
## --------------------------------
## 1           517     643    1160
##           0.050   0.062
## --------------------------------
## Total      9291    1006   10297
## ================================
```

```
roc.xgb <-roc(output.vector.test, probabilities.xgb)
```

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

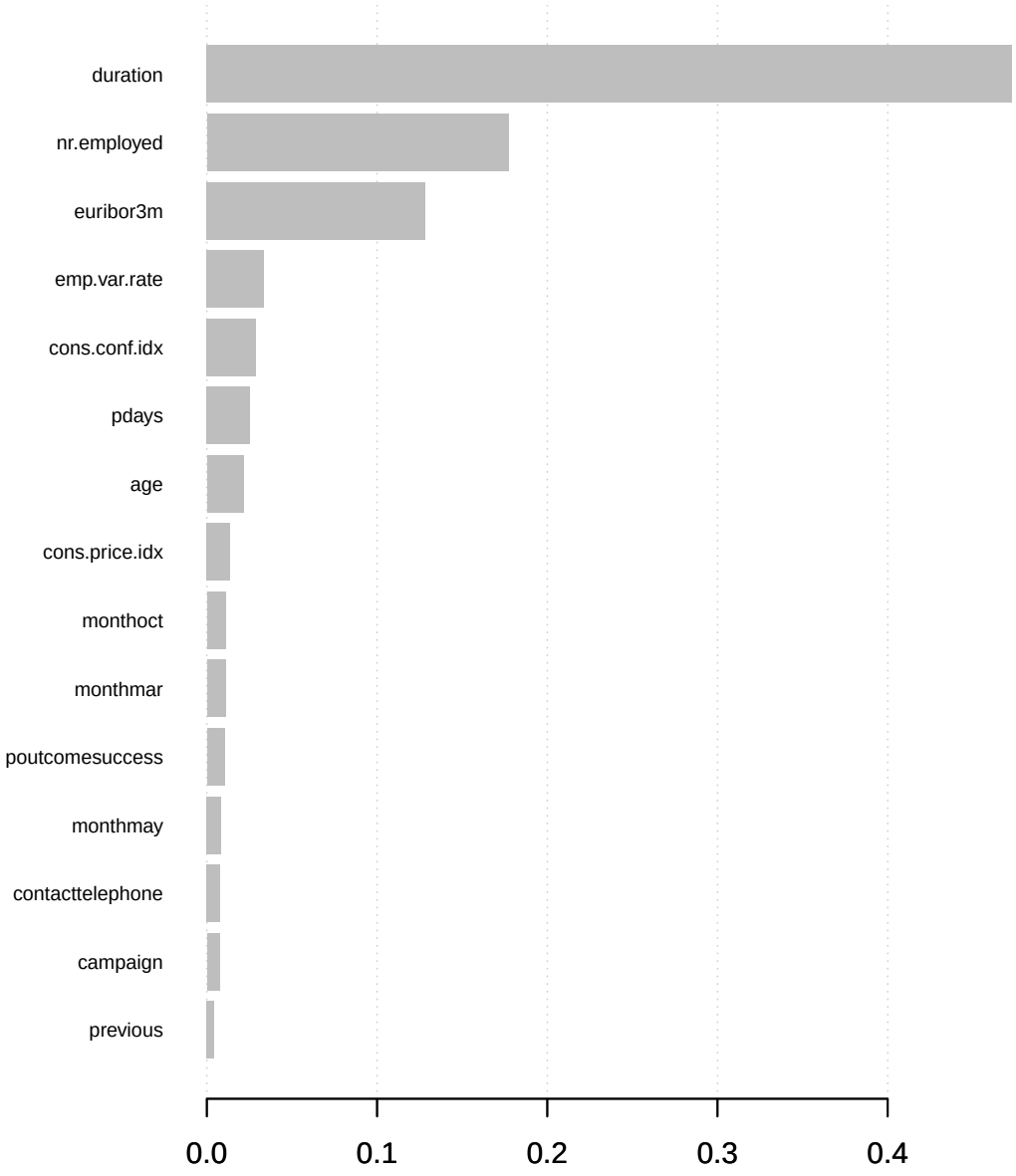
```
## Setting direction: controls < cases
```

```
print(roc.xgb)
```

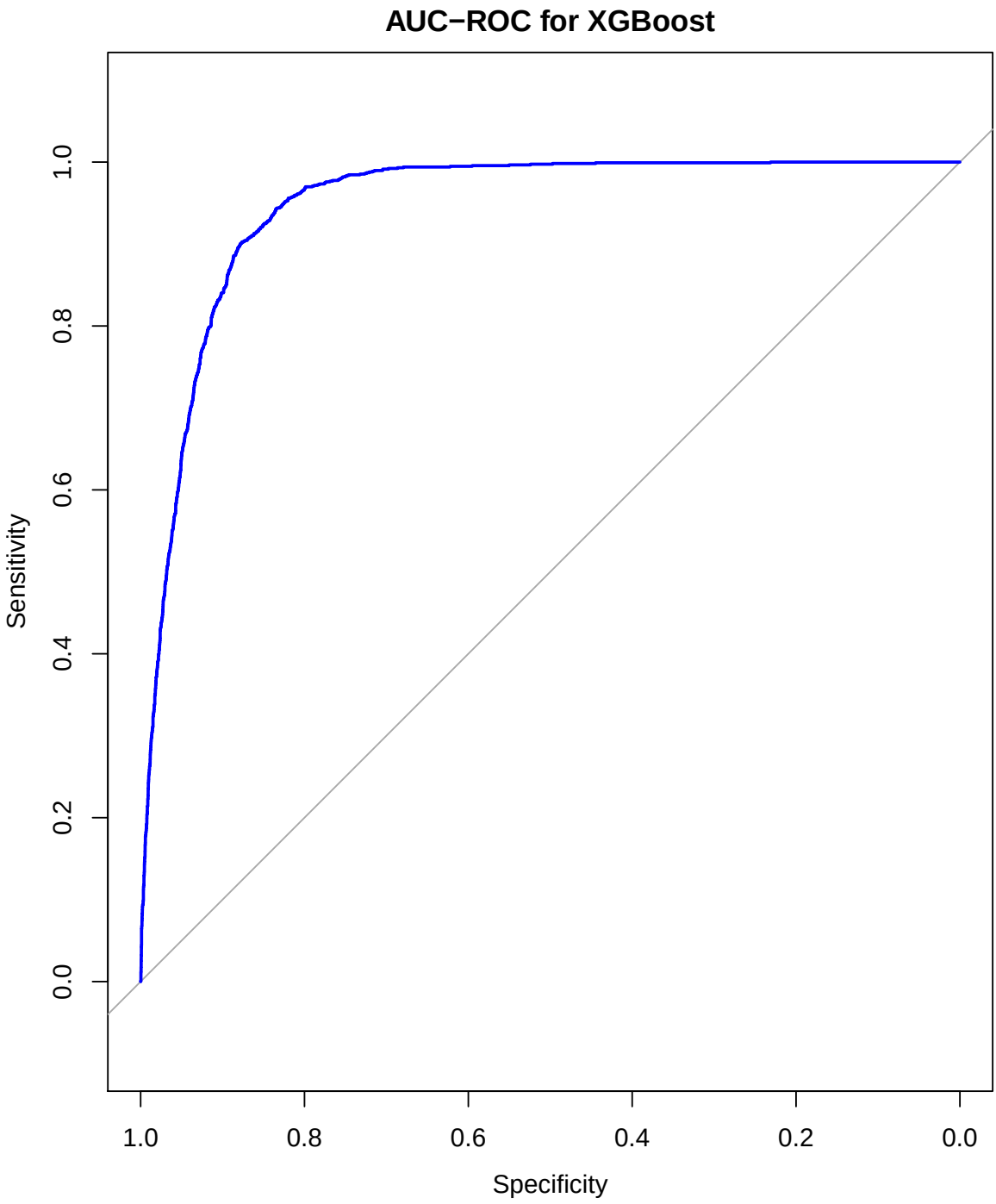
```
##
## Call:
## roc.default(response = output.vector.test, predictor = probabilities.xgb)
##
## Data: probabilities.xgb in 9137 controls (output.vector.test 0) < 1160 cases (output.
## Area under the curve: 0.9469
```

Here can see that the XGBoost model achieved a 94.69% AUC-ROC in the test set. Let's visualize the top 15 features of the model and the model's out of sample AUC-ROC curve.

```
# ROC
# par(mfrow=c(2,1))
xgb.plot.importance(importance_matrix = importance, top_n = 15)
```




```
plot(roc.xgb, main="AUC-ROC for XGBoost", percent=TRUE, col="blue")
```


AUC−ROC for XGBoost
Sensitivity
1.0
0.8
0.6
0.4
0.2
0.0
1.0
0.8
0.6
0.4
0.2
0.0
Specificity

4 Results

We successfully built several machine learning models to forecast the likelihood of a particular client of the Portuguese bank to buy additional services from them.

The models chosen show relative high AUC-ROC curves for this set, which looks very promising for the applicability in production.

Improvements in the scores could be achieved by exploring hyper parameter tuning, cross validation, etc., and even ensemble models, however, they are beyond the scope of this assignment.

Below are the final rankings.

Model	AUC-ROC Score
Decision Tree	0.8741
Random Forest	0.9435
XGBoost	0.9469

5 Conclusions

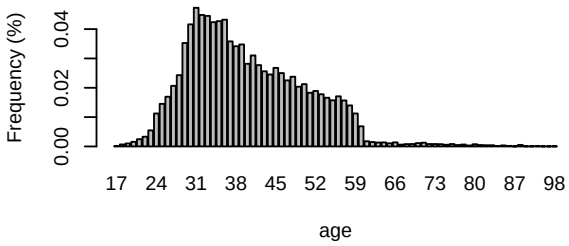
The results of the Random Forest model and the XGBoost model are very similar in score, however, the ranking of the feature importance of each model is slightly different, as we saw in previous charts. This might be an interesting area for further exploration.

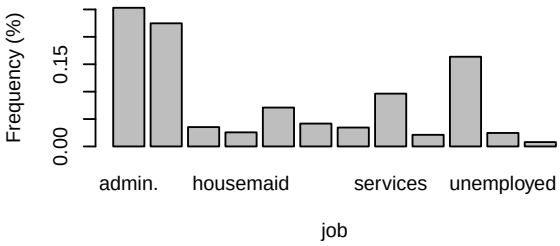
Since the AUC-ROC implies certain levels of acceptable False Positives and False Negatives in the results of a machine learning model, in practice, I believe that the model can definitely be improved by taking into account the opportunity cost of False Positives vs False Negatives.

For example, what is the cost of False Positive relative to False Negatives, meaning, should the model be improved to minimize FP or FN to achieve a specific financial return objective?

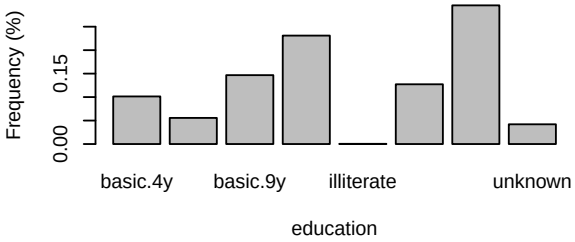
This sort of business decision is usually absent in machine learning exercises, however, it is a very fundamental part of why these models are built to begin with.

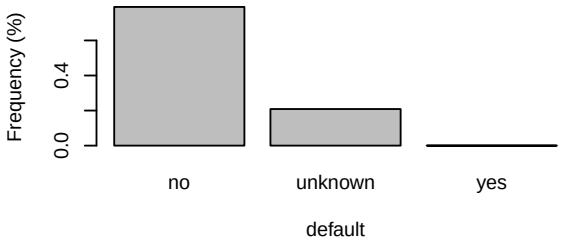
6 Appendix

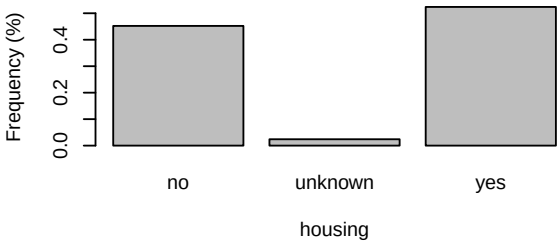

Frequency (%)
0.00
0.02
0.04
17 24 31 38 45 52 59 66 73 80 87 98
age

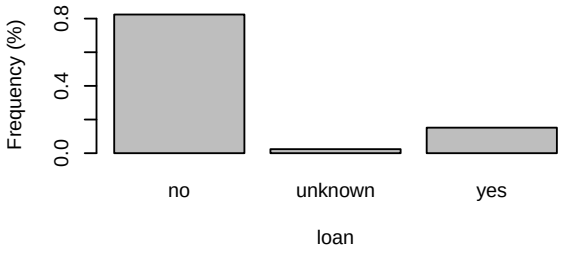

Frequency (%)
0.00
0.15
admin.
housemaid
services
unemployed
job

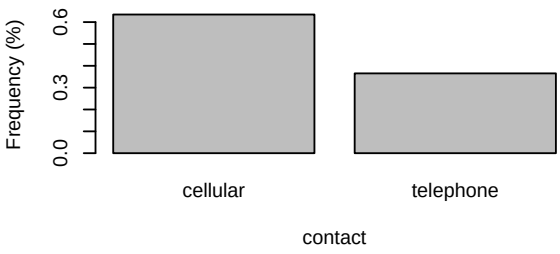

Frequency (%)
0.0
0.2
0.4
0.6
divorced
married
single
unknown
marital

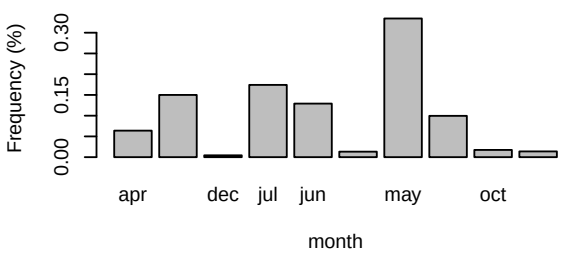

Frequency (%)
0.00
0.15
basic.4y
basic.9y
illiterate
unknown
education

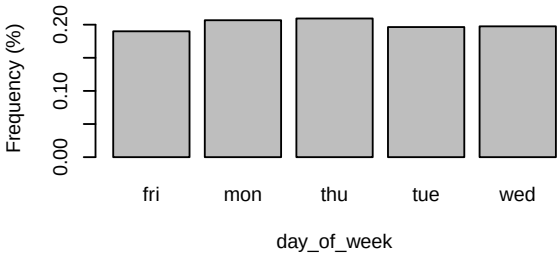

Frequency (%)
0.0
0.4
no
unknown
yes
default

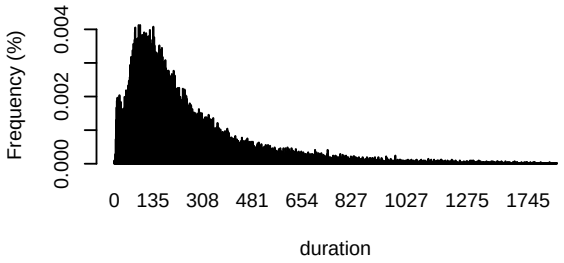

Frequency (%)
0.0
0.2
0.4
no
unknown
yes
housing

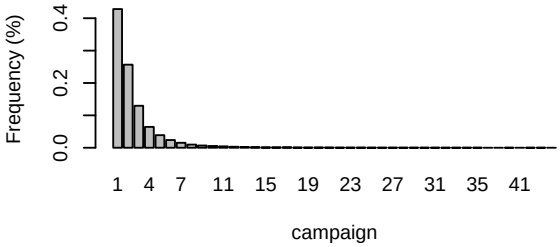

Frequency (%)
0.0
0.4
0.8
no
unknown
yes
loan

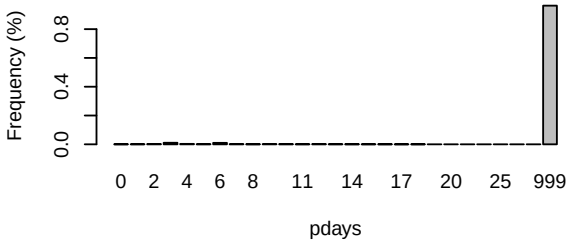

Frequency (%)
0.0
0.3
0.6
cellular
telephone
contact

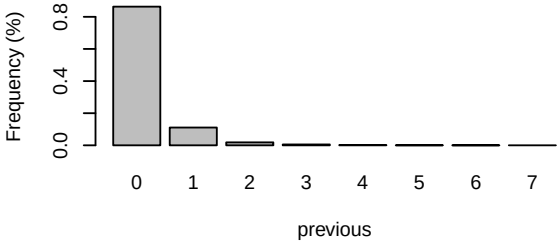

Frequency (%)
0.00
0.15
0.30
apr
dec
jul
jun
may
oct
month

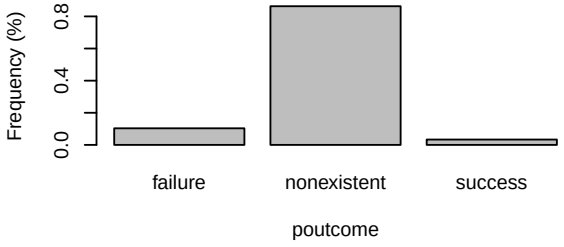

Frequency (%)
0.00
0.10
0.20
fri
mon
thu
tue
wed
day_of_week

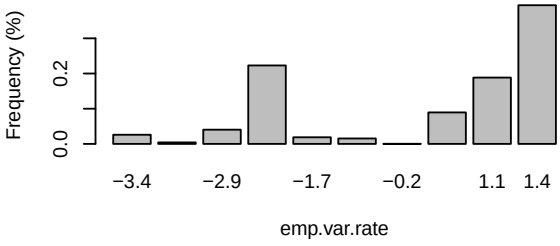

Frequency (%)
0.000
0.002
0.004
0
135
308
481
654
827
1027
1275
1745
duration

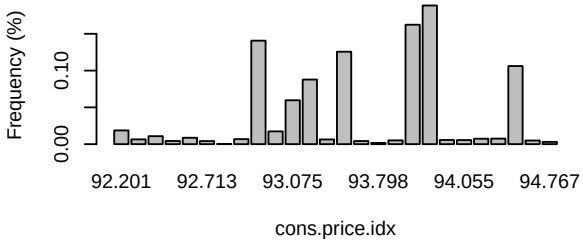

Frequency (%)
0.0
0.2
0.4
1
4
7
11
15
19
23
27
31
35
41
campaign

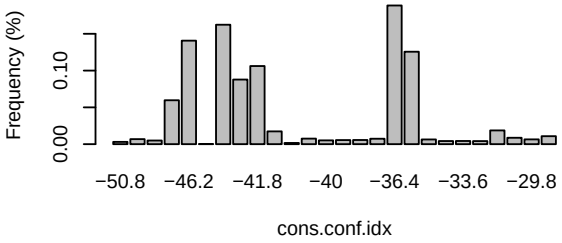

Frequency (%)
0.0
0.4
0.8
0
2
4
6
8
11
14
17
20
25
999
pdays

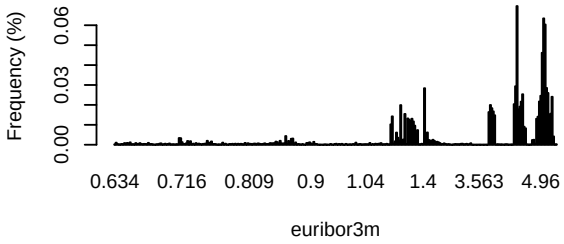

Frequency (%)
0.0
0.4
0.8
0
1
2
3
4
5
6
7
previous

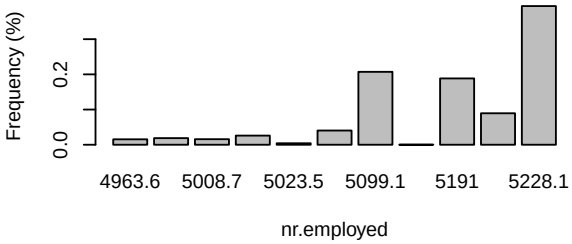

Frequency (%)
0.0
0.4
0.8
failure
nonexistent
success
poutcome

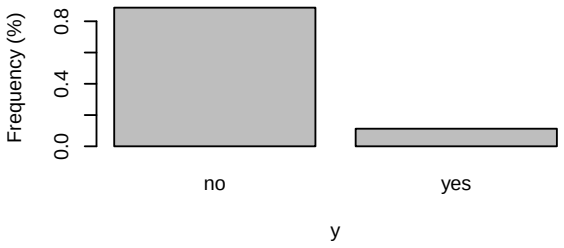

Frequency (%)
0.0
0.2
−3.4
−2.9
−1.7
−0.2
1.1
1.4
emp.var.rate


Frequency (%)
0.00
0.10
92.201
92.713
93.075
93.798
94.055
94.767
cons.price.idx


Frequency (%)
0.00
0.10
−50.8
−46.2
−41.8
−40
−36.4
−33.6
−29.8
cons.conf.idx


Frequency (%)
0.00
0.03
0.06
0.634
0.716
0.809
0.9
1.04
1.4
3.563
4.96
euribor3m


Frequency (%)
0.0
0.2
4963.6
5008.7
5023.5
5099.1
5191
5228.1
nr.employed


Frequency (%)
0.0
0.4
0.8
no
yes
y