Red Wine Quality Score Prediction

Marcello Riderelli Belli

13/6/2020

Data Source and citation

This dataset is public available for research. The details are described in [Cortez et al., 2009]. Please include this citation if you plan to use this database:

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553. ISSN: 0167-9236.

Available at:

[@Elsevier] http://dx.doi.org/10.1016/j.dss.2009.05.016

[Pre-press (pdf)] http://www3.dsi.uminho.pt/pcortez/winequality09.pdf

[bib] http://www3.dsi.uminho.pt/pcortez/dss09.bib

- Title: Wine Quality
- Sources Created by: Paulo Cortez (Univ. Minho), Antonio Cerdeira, Fernando Almeida, Telmo Matos and Jose Reis (CVRVV) @ 2009
- Past Usage:

  P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553. ISSN: 0167-9236.

  In the above reference, two datasets were created, using red and white wine samples. The inputs include objective tests (e.g. PH values) and the output is based on sensory data (median of at least 3 evaluations made by wine experts). Each expert graded the wine quality between 0 (very bad) and 10 (very excellent). Several data mining methods were applied to model these datasets under a regression approach. The support vector machine model achieved the best results. Several metrics were computed: MAD, confusion matrix for a fixed error tolerance (T), etc. Also, we plot the relative importances of the input variables (as measured by a sensitivity analysis procedure).

- Relevant Information:

  The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. For more details, consult: http://www.vinhoverde.pt/en/ or the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

  These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are munch more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

- Number of Instances: red wine - 1599; white wine - 4898.
- Number of Attributes: 11 + output attribute

Note: several of the attributes may be correlated, thus it makes sense to apply some sort of feature selection.

- Attribute information:

  For more information, read [Cortez et al., 2009].

  Input variables (based on physicochemical tests):

  1. fixed acidity
  2. volatile acidity
  3. citric acid
  4. residual sugar
  5. chlorides
  6. free sulfur dioxide
  7. total sulfur dioxide
  8. density
  9. pH
  10. sulphates
  11. alcohol
  12. quality (score between 0 and 10) - output variable (based on sensory data):

- Missing Attribute Values: None

Introduction

This work is an application of supervised machine learning. We want to find a classification algorithm that will use physicochemical properties (measures) of red wines as predictors and predict the wine quality (score from wine experts) as an outcome.

Executive Summary

At first, before deep divig into data elaboration, we'll visualize data to check if there is a simple separation of classes, at least some o them, and point out the main challenge of this work: there are many wines with average score and few with a bad or excellent one.

Then will check for the performance of some classification algorithm like KNN, Classification Trees, and Random Forest.

After that we'll do some transformation of the dataset to overcome the fitting problem due to the unbalanced sampling. This will lead to overfitting in some classes.

Importing Data

The dataset is accessible as a text file structured as CSV

To use the code as is, please use getwd() command to know your working directory then create in it "R" and "R/Data" subdirectories, finally copy the file "winequality-red.csv" in "~/R/Data/"

All files of this work are accesible at:

https://github.com/Marce68/Wines_Capstone

```r
rw <- read.csv("~/R/Data/winequality-red.csv")
str(rw)
```

```
## 'data.frame':    1599 obs. of  12 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
```

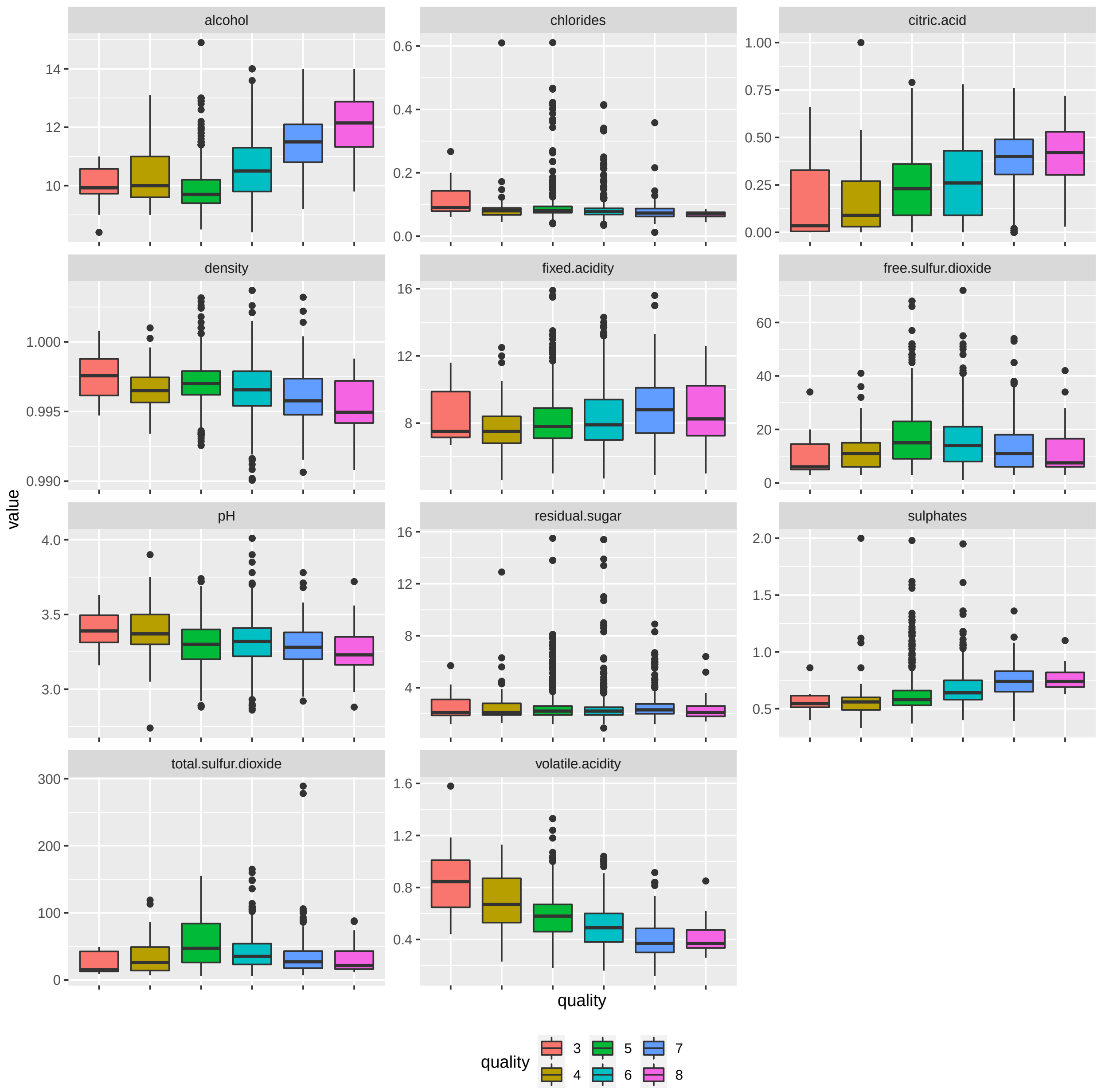
Visualization and Exploration

Let's give a look at the predictors distributions, before this we note that scores are categories so it is better to coerce the output as factor

```r
rw$quality <- as.factor(rw$quality)
```

Let's see if we can find any type of evident separation in the dataset

```r
rw %>% pivot_longer(colnames(rw)[-which(colnames(rw) == "quality")],"CF") %>%
  ggplot(aes(quality, value, fill = quality)) +
  geom_boxplot() +
  facet_wrap(~CF, scales = "free", ncol = 3) +
  theme(axis.text.x = element_blank(), legend.position="bottom")
```



The distribution are quite overlapped, so it is not easy to find a simple classification algorithm. Further, if we check the averages of the features (table below) we see that only for few physicochemical properties, like sulphates, there is a linear relation with quality scores

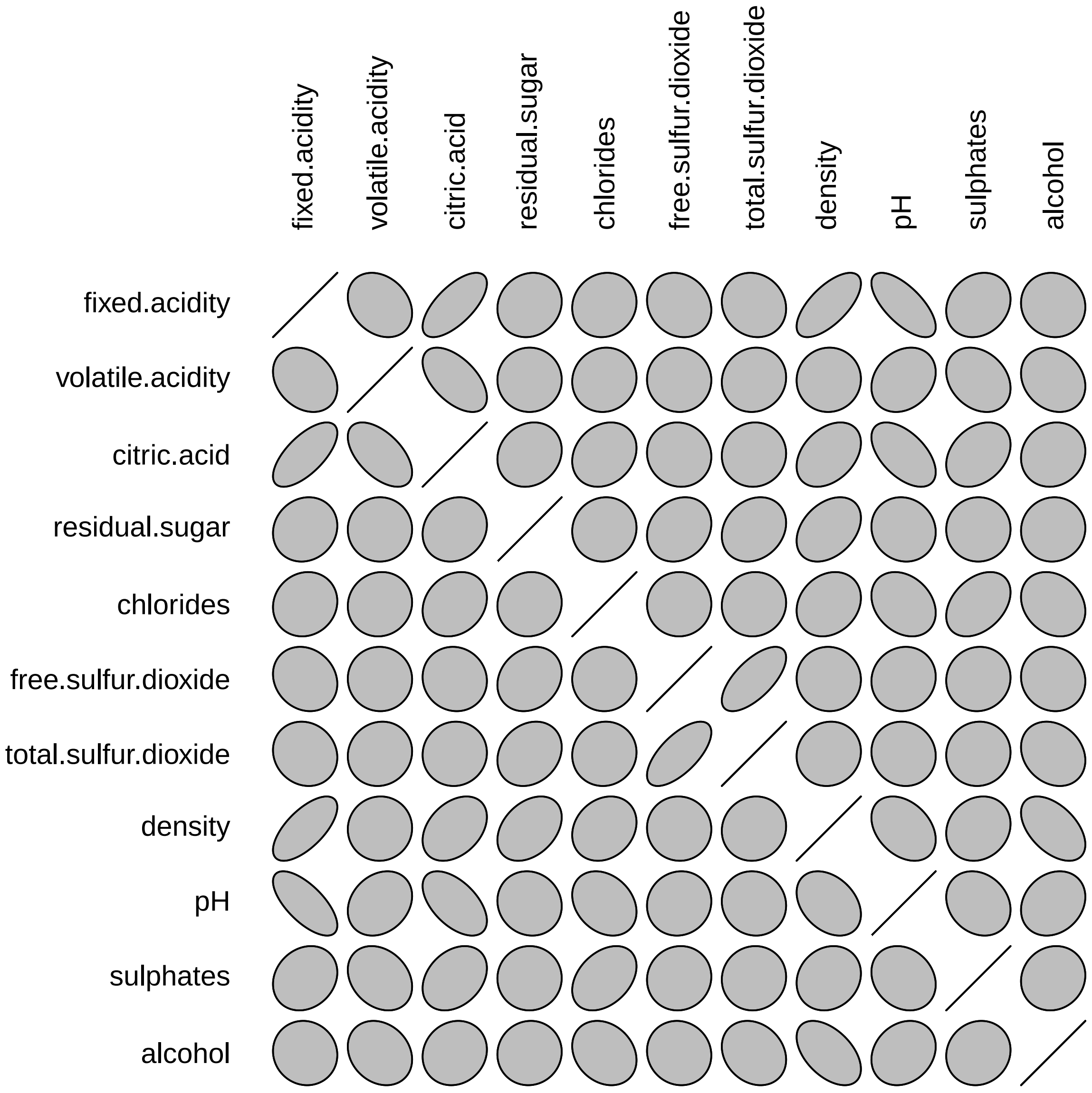
```r
avg_rw <- rw %>%
  group_by(quality) %>%
  summarize(fixed.acidity = sum(fixed.acidity)/n(),
            volatile.acidity = sum(volatile.acidity)/n(),
            citric.acid = sum(citric.acid)/n(),
            residual.sugar = sum(residual.sugar)/n(),
            chlorides = sum(chlorides)/n(),
            free.sulfur.dioxide = sum(free.sulfur.dioxide)/n(),
            total.sulfur.dioxide = sum(total.sulfur.dioxide)/n(),
            density = sum(density)/n(),
            pH = sum(pH)/n(),
            sulphates = sum(sulphates)/n(),
            alcohol = sum(alcohol)/n(),)

# three lines of code to fit the table in the page
avg_rw_t <- as.data.frame(as.matrix(t(avg_rw)))
colnames(avg_rw_t) <- rep("Q_Score",6)
avg_rw_t %>%  knitr::kable()
```

	Q_Score	Q_Score	Q_Score	Q_Score	Q_Score	Q_Score
quality	3	4	5	6	7	8
fixed.acidity	8.3600	7.7792	8.1673	8.3472	8.8724	8.5667
volatile.acidity	0.88450	0.69396	0.57704	0.49748	0.40392	0.42333
citric.acid	0.17100	0.17415	0.24369	0.27382	0.37518	0.39111
residual.sugar	2.6350	2.6943	2.5289	2.4772	2.7206	2.5778
chlorides	0.122500	0.090679	0.092736	0.084956	0.076588	0.068444
free.sulfur.dioxide	11.000	12.264	16.984	15.712	14.045	13.278
total.sulfur.dioxide	24.900	36.245	56.514	40.870	35.020	33.444
density	0.99746	0.99654	0.99710	0.99662	0.99610	0.99521
pH	3.3980	3.3815	3.3049	3.3181	3.2908	3.2672
sulphates	0.57000	0.59642	0.62097	0.67533	0.74126	0.76778
alcohol	9.9550	10.2651	9.8997	10.6295	11.4659	12.0944

```
rm(avg_rw_t)
```

The variables are scarcely correlated apart from those related to acidity

```
ctab <- cor(rw[,1:11])
# round(ctab, 3) %>% knitr::kable()
plotcorr(ctab, mar = c(0.1, 0.1, 0.1, 0.1))
```



Note that the dataset is unbalanced: we have low samples count in low quality and high quality classes, this is also quite obvious because given a location where wine production is a consolidated tradition, Portugal in this case, there are for sure many medium quality wines and very few poor and excellent wines.

```
rw %>%
  group_by(quality) %>%
  summarize(Wines = n()) %>%
  knitr::kable()
```

quality	Wines
3	10
4	53
5	681
6	638

quality	Wines
7	199
8	18

Preparing train and test sets

Let's build our train and test set, the test set is 20% of the full set

```
set.seed(1, sample.kind="Rounding") # `set.seed(1)` for R version <= 3.5
tst_idx <- createDataPartition(y = rw$quality,
                               times = 1,
                               p = 0.2,
                               list = FALSE)
rw_train <- rw[-tst_idx, ]
rw_test <- rw[tst_idx, ]
```

We have few wines with 3 and 8 quality score in the train set ...

```
rw_train %>%
  group_by(quality) %>%
  summarize(Wines = n()) %>%
  knitr::kable()
```

quality	Wines
3	8
4	42
5	544
6	510
7	159
8	14

... and even less in the test set

```
rw_test %>%
  group_by(quality) %>%
  summarize(Wines = n()) %>%
  knitr::kable()
```

quality	Wines
3	2
4	11
5	137
6	128
7	40
8	4

Machine Learning Models: a quick scan

We'll apply now several classification algorithms with increasing complexity and check for the confusion matrix and accuracy. Accuracy by itself is not a good metrics when working with unbalanced dataset, in fact we'll see that the confusion matrix will show mostly a poor performance of the algorithms especially in the minority class. The confusion matrix in the following pages are very self explaining.

Let's try a very simple classification based on cutoffs of sulphates content measures

```r
cut_offs <- (avg_rw$sulphates[1:5] + avg_rw$sulphates[2:6])/2
y_hat <- rep("0", length(rw_test$quality))

# Let's see if the variable sulphates contain some usefull information
y_hat[which(rw_test$sulphates <= cut_offs[1])] <- 3
y_hat[which(rw_test$sulphates > cut_offs[1] & rw_test$sulphates <= cut_offs[2])] <- 4
y_hat[which(rw_test$sulphates > cut_offs[2] & rw_test$sulphates <= cut_offs[3])] <- 5
y_hat[which(rw_test$sulphates > cut_offs[3] & rw_test$sulphates <= cut_offs[4])] <- 6
y_hat[which(rw_test$sulphates > cut_offs[4] & rw_test$sulphates <= cut_offs[5])] <- 7
y_hat[which(rw_test$sulphates > cut_offs[5])] <- 8

y_hat <- as.factor(y_hat)

confusionMatrix(y_hat, rw_test$quality)$table
```

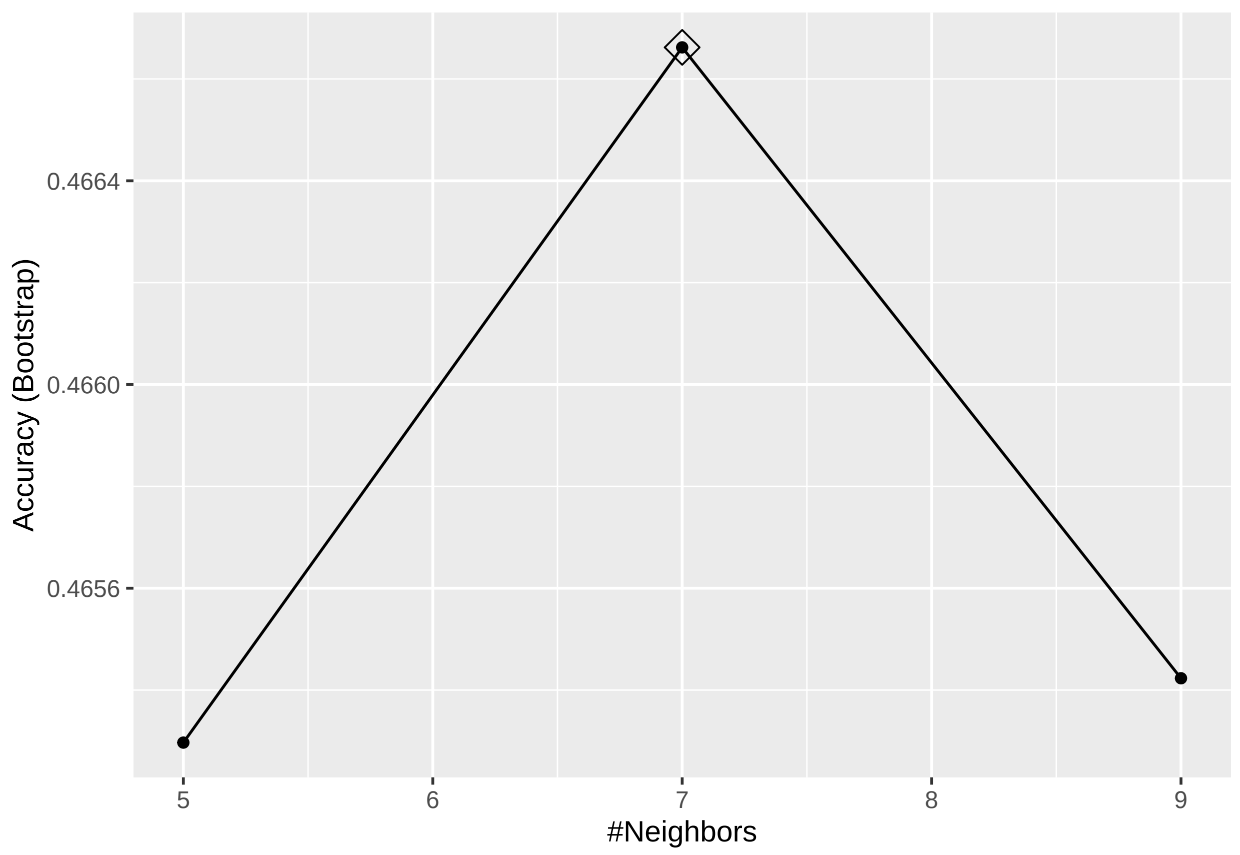
```
##           Reference
## Prediction  3  4  5  6  7  8
##          3  1  6 70 36  4  0
##          4  0  2  6 14  1  0
##          5  0  1 24 12  3  0
##          6  0  1 12 28  5  3
##          7  0  0  6  7  5  0
##          8  1  1 19 31 22  1
```

```r
cut_acc <- confusionMatrix(y_hat, rw_test$quality)$overall["Accuracy"]

acc_results <- data_frame(Method = "Cutoffs",
                          Accuracy = cut_acc)

acc_results %>% knitr::kable()
```

Method	Accuracy
Cutoffs	0.18944

Let's try KNN although we know that it's not performing well on unbalanced datasets

```r
rw_knn <- train(quality ~ .,
                method = "knn",
                data = rw_train)

ggplot(rw_knn, highlight = TRUE)
```



```
rw_knn$bestTune
```

```
##   k
## 2 7
```

```
rw_knn$finalModel
```

```
## 7-nearest neighbor model
## Training set outcome distribution:
##
##   3   4   5   6   7   8
##   8  42 544 510 159  14
```

```
y_hat_knn <- predict(rw_knn, rw_test, type = "raw")
confusionMatrix(y_hat_knn, rw_test$quality)$table
```

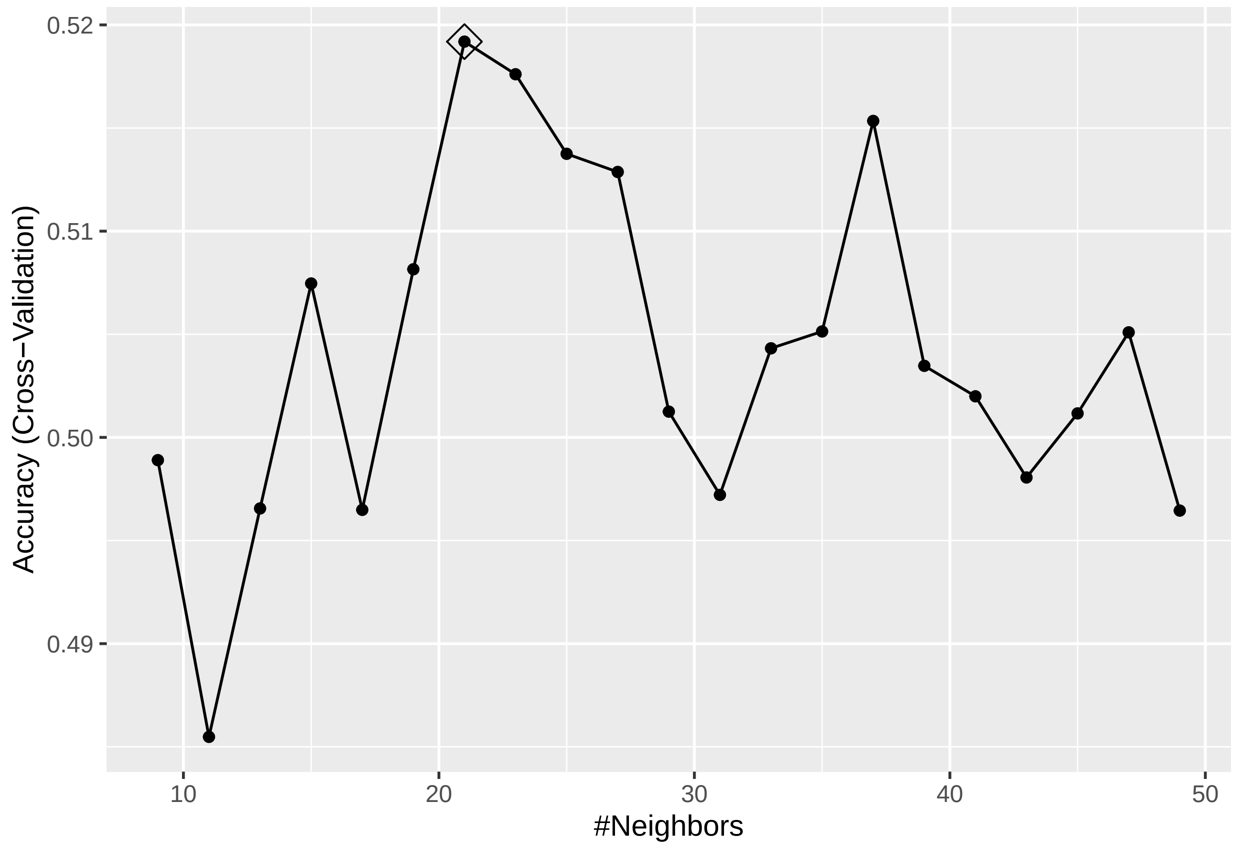
```
##           Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  0  0  0  0  0  0
##          5  2  6 90 58  8  1
##          6  0  5 43 60 19  2
##          7  0  0  4  9 13  0
##          8  0  0  0  1  0  1
```

```
knn_acc <- confusionMatrix(y_hat_knn, rw_test$quality)$overall["Accuracy"]

acc_results <- bind_rows(acc_results,
                         data_frame(Method="KNN",
                                    Accuracy = knn_acc))
acc_results %>% knitr::kable()
```

Method	Accuracy
Cutoffs	0.18944
KNN	0.50932

Let's perform also some tuning to find the best value for K, this computation is quite long

```
control <- trainControl(method = "cv", number = 10, p = .9)
rw_knn_cv <- train(quality ~ ., method = "knn",
                   data = rw_train,
                   tuneGrid = data.frame(k = seq(9, 50, 2)),
                   trControl = control)
ggplot(rw_knn_cv, highlight = TRUE)
```



```
# rw_knn_cv$bestTune
# rw_knn_cv$finalModel

y_hat_knn_cv <- predict(rw_knn_cv, rw_test, type = "raw")
confusionMatrix(y_hat_knn_cv, rw_test$quality)$table
```

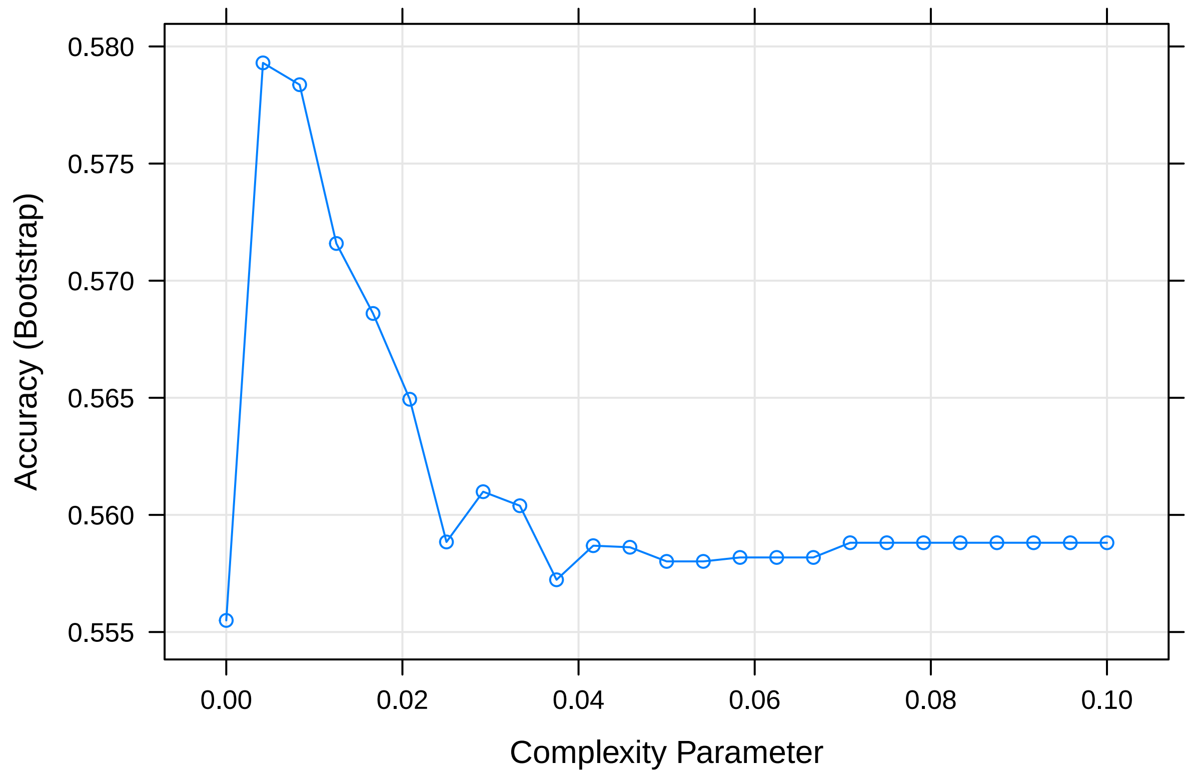
```
##             Reference
## Prediction   3   4   5   6   7   8
##           3   0   0   0   0   0   0
##           4   0   0   0   0   0   0
##           5   2   6 100  52  11   0
##           6   0   5  36  71  22   4
##           7   0   0   1   5   7   0
##           8   0   0   0   0   0   0
```

```r
knn_cv_acc <- confusionMatrix(y_hat_knn_cv, rw_test$quality)$overall["Accuracy"]

acc_results <- bind_rows(acc_results,
                         data_frame(Method="KNN CV",
                                    Accuracy = knn_cv_acc))
acc_results %>% knitr::kable()
```

Method	Accuracy
Cutoffs	0.18944
KNN	0.50932
KNN CV	0.55280

Now Classification Tree with some tuning

```r
rw_rpart <- train(quality ~ .,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
                     data = rw_train)
plot(rw_rpart, highlight = TRUE)
```



```r
# rw_rpart$bestTune
# rw_rpart$finalModel

y_hat_rpart <- predict(rw_rpart, rw_test, type = "raw")
confusionMatrix(y_hat_rpart, rw_test$quality)$table
```

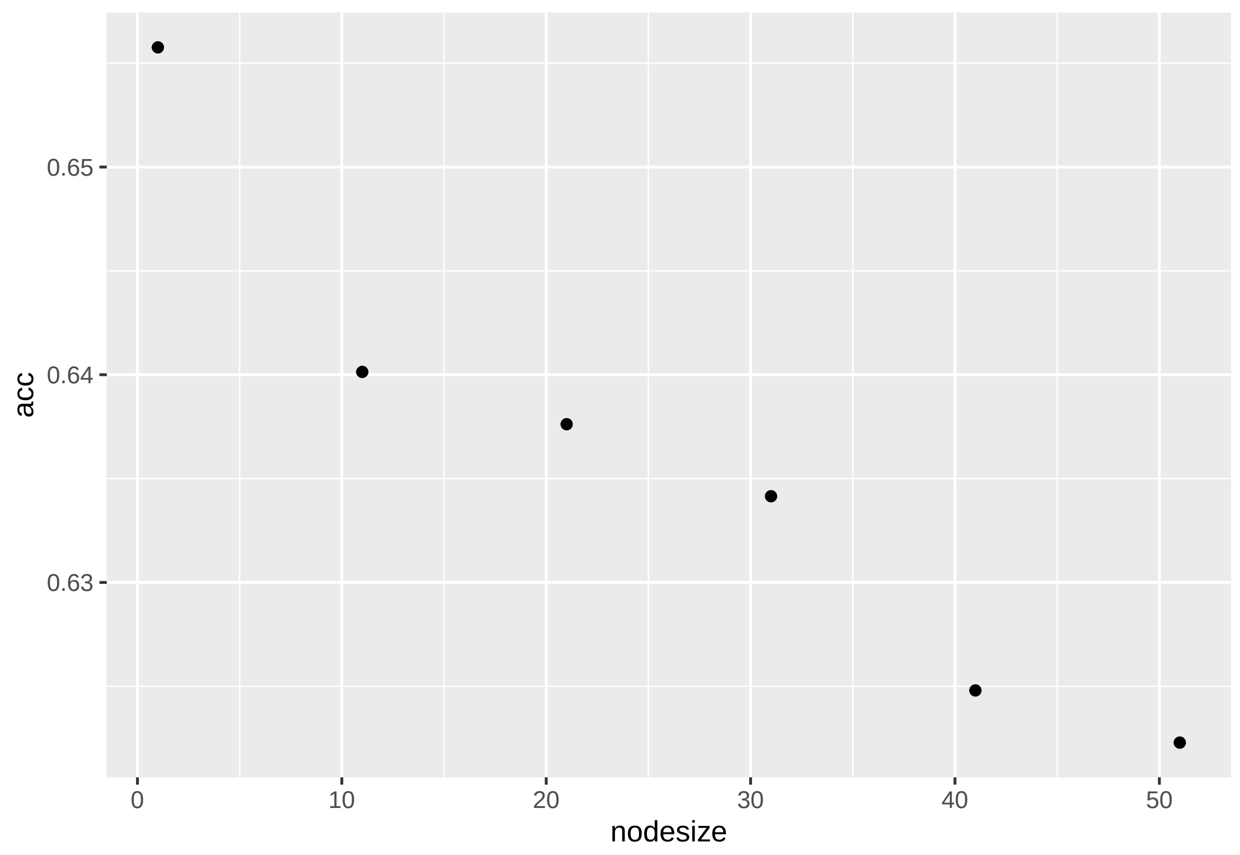
```
##           Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  0  0  0  0  0  0
##          5  1  7 90 43  5  0
##          6  1  4 43 64 19  2
##          7  0  0  4 21 16  2
##          8  0  0  0  0  0  0
```

```r
ct_acc <- confusionMatrix(y_hat_rpart, rw_test$quality)$overall["Accuracy"]

acc_results <- bind_rows(acc_results,
                         data_frame(Method="Classification Tree",
                                    Accuracy = ct_acc))
acc_results %>% knitr::kable()
```

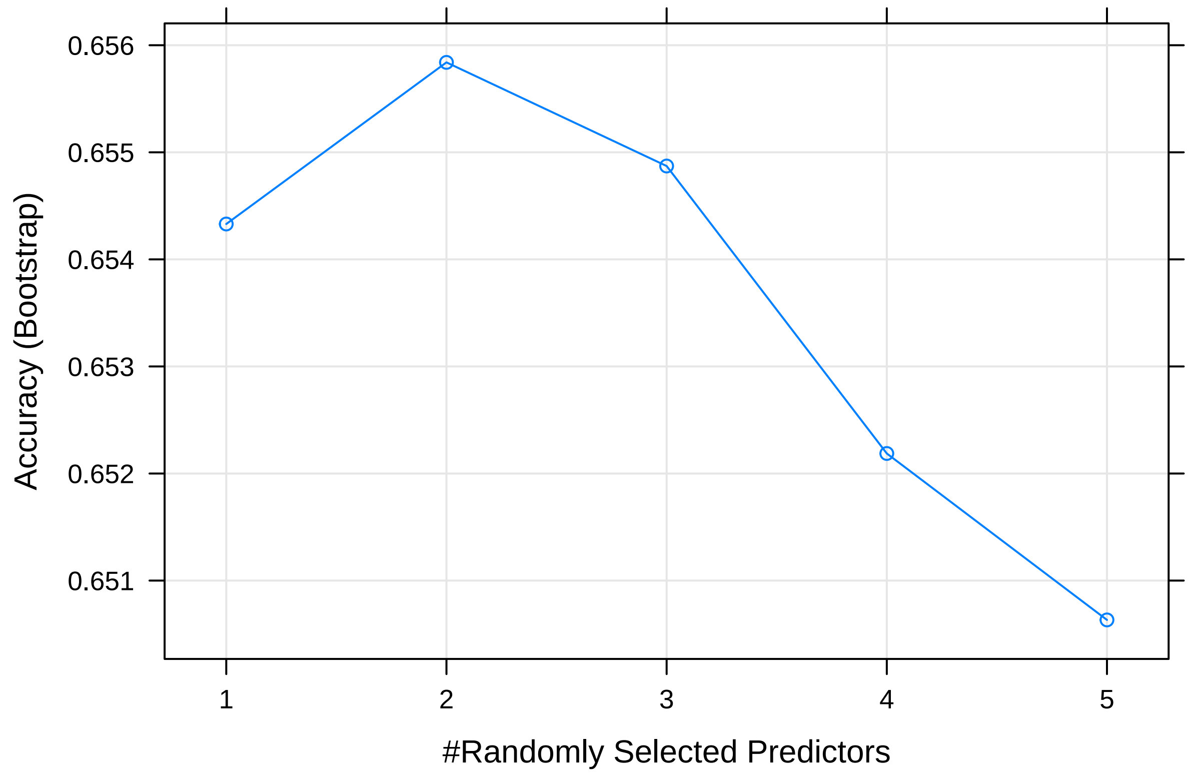
Method	Accuracy
Cutoffs	0.18944
KNN	0.50932
KNN CV	0.55280
Classification Tree	0.52795

The Random Forest should find now the best fitting

```r
set.seed(1, sample.kind="Rounding") # `set.seed(1)` for R version <= 3.5
nodesize <- seq(1, 51, 10)
acc <- sapply(nodesize, function(ns){
  train(quality ~ .,
        method = "rf",
        data = rw_train,
        tuneGrid = data.frame(mtry = 2),
        nodesize = ns)$results$Accuracy
})
qplot(nodesize, acc)
```



```r
set.seed(1, sample.kind="Rounding") # `set.seed(1)` for R version <= 3.5
rw_rf <- train(quality ~ .,
               method = "rf",
               tuneGrid = data.frame(mtry = seq(1:5)),
               nodesize = nodesize[which.max(acc)],
               data = rw_train)
plot(rw_rf)
```



```r
# rw_rf$bestTune
# rw_rf$finalModel

y_hat_rf <- predict(rw_rf, rw_test, type = "raw")
confusionMatrix(y_hat_rf, rw_test$quality)$table
```

```
##             Reference
## Prediction    3   4   5   6   7   8
##            3   0   0   0   0   0   0
##            4   1   0   0   1   0   0
##            5   1   8 114  30   1   0
##            6   0   3  22  88  13   2
##            7   0   0   1   9  26   2
##            8   0   0   0   0   0   0
```

```r
rf_acc <- confusionMatrix(y_hat_rf, rw_test$quality)$overall["Accuracy"]

acc_results <- bind_rows(acc_results,
                         data_frame(Method="Random Forest",
                                    Accuracy = rf_acc))
acc_results %>% knitr::kable()
```

Method	Accuracy
Cutoffs	0.18944
KNN	0.50932
KNN CV	0.55280
Classification Tree	0.52795

Method	Accuracy
Random Forest	0.70807

The performance of all algorithms is not good, the reason is probably the unbalancing of data.

Further transformation: balancing

To balance a dataset we have basically two options, undersampling or oversampling. We don't have enough data to apply undersampling of majority classes so let's see what will happen with the oversampling of the minority ones.

```r
y <- rw$quality
x <- rw[ ,1:11]
usrw <- upSample(x, y, list=FALSE, yname = "quality")

usrw %>%
  group_by(quality) %>%
  summarize(Q = n()) %>%
  knitr::kable()
```

quality	Q
3	681
4	681
5	681
6	681
7	681
8	681

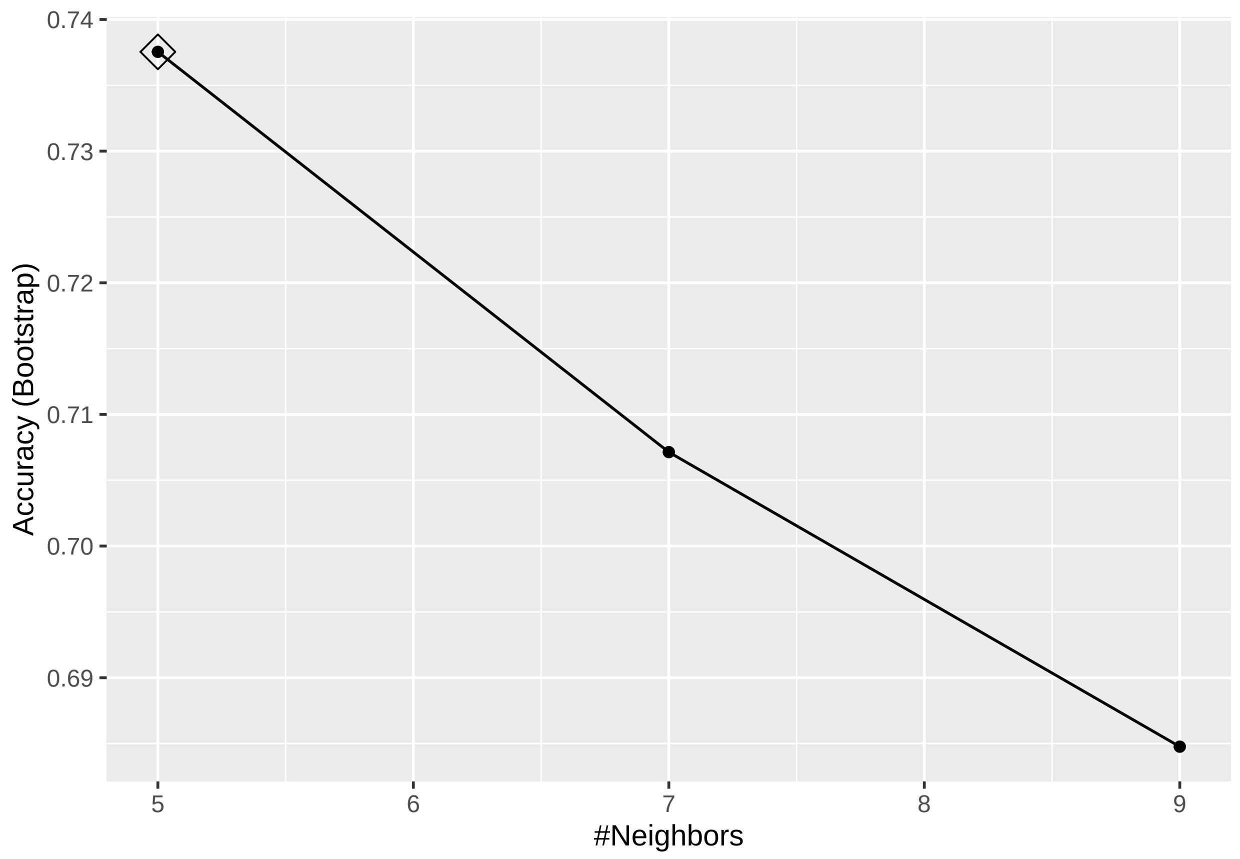
Now the dataset is perfectly balanced, we can build new train set and test set

```r
set.seed(1968, sample.kind="Rounding") # `set.seed(1)` for R version <= 3.5
tst_idx <- createDataPartition(y = usrw$quality,
                               times = 1,
                               p = 0.2,
                               list = FALSE)
usrw_train <- usrw[-tst_idx, ]
usrw_test <- usrw[tst_idx, ]
```

Machine Learning Model: final assessment and results

KNN after oversampling

```r
usrw_knn <- train(quality ~ .,
                  method = "knn",
                  data = usrw_train)

ggplot(usrw_knn, highlight = TRUE)
```



```
usrw_knn$bestTune
```

```
##   k
## 1 5
```

```
usrw_knn$finalModel
```

```
## 5-nearest neighbor model
## Training set outcome distribution:
##
##   3   4   5   6   7   8
## 544 544 544 544 544 544
```

```
y_hat_knn <- predict(usrw_knn, usrw_test, type = "raw")
confusionMatrix(y_hat_knn, usrw_test$quality)$table
```

```
##           Reference
## Prediction   3   4   5   6   7   8
##          3 137   0   2   2   0   0
##          4   0 137  26  14   3   0
##          5   0   0  55  38   2   0
##          6   0   0  33  36   7   0
##          7   0   0  18  41 110   0
##          8   0   0   3   6  15 137
```

```
knn_acc <- confusionMatrix(y_hat_knn, usrw_test$quality)$overall["Accuracy"]

acc_results <- bind_rows(acc_results,
                         data_frame(Method="KNN (oversampling)",
                                    Accuracy = knn_acc))
acc_results %>% knitr::kable()
```

Method	Accuracy
Cutoffs	0.18944
KNN	0.50932
KNN CV	0.55280
Classification Tree	0.52795
Random Forest	0.70807
KNN (oversampling)	0.74453

Random Forest after oversampling

```
set.seed(1968, sample.kind="Rounding") # `set.seed(1)` for R version <= 3.5
usrw_rf <- train(quality ~ .,
                 method = "rf",
                 tuneGrid = data.frame(mtry = 2),
                 nodesize = 2,
                 data = usrw_train)
# plot(usrw_rf)
# usrw_rf$bestTune
# usrw_rf$finalModel

y_hat_usrf <- predict(usrw_rf, usrw_test, type = "raw")
confusionMatrix(y_hat_usrf, usrw_test$quality)$table
```

```
##             Reference
## Prediction   3   4   5   6   7   8
##          3 137   0   3   0   0   0
##          4   0 137   3   0   0   0
##          5   0   0 109  34   0   0
##          6   0   0  22  91   3   0
##          7   0   0   0  12 134   0
##          8   0   0   0   0   0 137
```

```
rf_acc <- confusionMatrix(y_hat_usrf, usrw_test$quality)$overall["Accuracy"]

acc_results <- bind_rows(acc_results,
                         data_frame(Method="Random Forest (oversampling)",
                                    Accuracy = rf_acc))
acc_results %>% knitr::kable()
```

Method	Accuracy
Cutoffs	0.18944
KNN	0.50932
KNN CV	0.55280
Classification Tree	0.52795
Random Forest	0.70807
KNN (oversampling)	0.74453
Random Forest (oversampling)	0.90633

At the end we get an accuracy greater tham 0.90 and a confusion matrix showing a perfect prediction in minority classes and an acceptable performance in the majority ones also.

Conclusion and further development

After oversampling we have now a problem of overfitting of the minority class which is quite easy to foresee because we are randomly adding observation with replacement, so we have for sure several copies of the same observation. A further improvement could be acheived doing a bit of undersampling of the original minority class or better use a syntetic method (as SMOTE) for the generation of new observations in the minority classes.