Task 8

for Advanced Methods for Regression and Classification

Teodor Chakarov

03.01.2023

Contents

Loading and splitting the data

```
library(rpart)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

We will load the bank dataset as for exercise 5.

```
df <- read.csv2("bank.csv")
df$y <- as.factor(df$y)

str(df)
```

```
## 'data.frame':    4521 obs. of  17 variables:
##  $ age      : int  30 33 35 30 59 35 36 39 41 43 ...
##  $ job      : chr  "unemployed" "services" "management" "management" ...
##  $ marital  : chr  "married" "married" "single" "married" ...
##  $ education: chr  "primary" "secondary" "tertiary" "tertiary" ...
##  $ default  : chr  "no" "no" "no" "no" ...
```

```
##  $ balance  : int  1787 4789 1350 1476 0 747 307 147 221 -88 ...
##  $ housing  : chr  "no" "yes" "yes" "yes" ...
##  $ loan     : chr  "no" "yes" "no" "yes" ...
##  $ contact  : chr  "cellular" "cellular" "cellular" "unknown" ...
##  $ day      : int  19 11 16 3 5 23 14 6 14 17 ...
##  $ month    : chr  "oct" "may" "apr" "jun" ...
##  $ duration : int  79 220 185 199 226 141 341 151 57 313 ...
##  $ campaign : int  1 1 1 4 1 2 1 2 2 1 ...
##  $ pdays    : int  -1 339 330 -1 -1 176 330 -1 -1 147 ...
##  $ previous : int  0 4 1 0 0 3 2 0 0 2 ...
##  $ poutcome : chr  "unknown" "failure" "failure" "unknown" ...
##  $ y        : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
```

We will take 3500 random samples for training set and the rest are for the test.

```
set.seed(1234555)
row_Count <- floor(round(nrow(df)*2.0/3))
train_Data <- sample(seq_len(nrow(df)), size = 3500)

train <- df[train_Data, ]
test <- df[-train_Data, ]
```

```
nrow(train)
```

```
## [1] 3500
```

```
nrow(test)
```

```
## [1] 1021
```

Lest's see the distribution of the classes on the two sets.

```
table(train$y)
```

```
##
##   no  yes
## 3100  400
```

```
table(test$y)
```

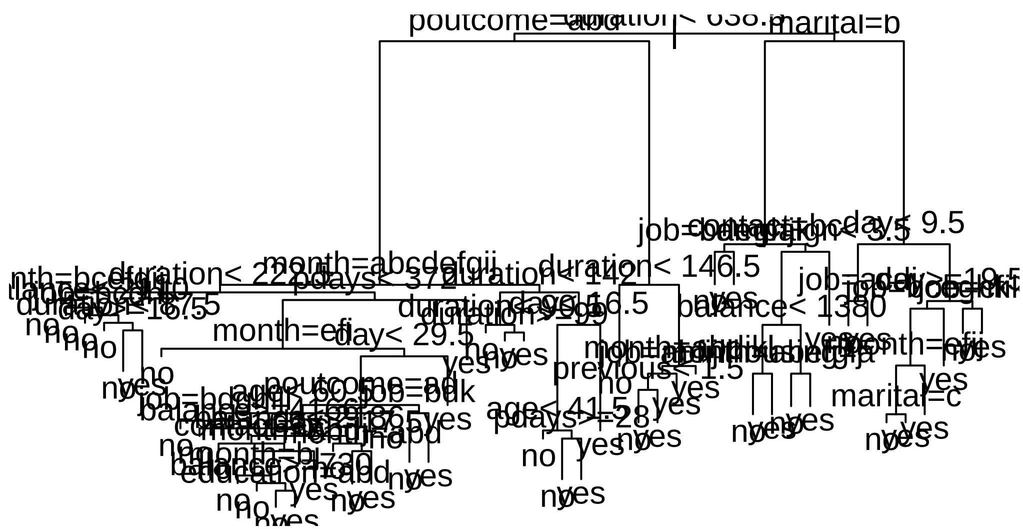
```
##
##  no yes
## 900 121
```

Decision tree

For T0, we will put cp=0 because we want the most complex tree to see how our data is being split.

```r
tree1 <- rpart(y~., data = train, method = "class", cp=0, xval=20)

plot(tree1)
text(tree1)
```



As we see on the plot, we have a lot of splits for our data and that makes the tree complex and deep.

```r
predict_tree1 <- predict(tree1, test, type="class")

(TAB <- table(test$y,predict_tree1))
```

```
##      predict_tree1
##        no yes
##   no  857  43
##   yes  78  43
```

```r
1-sum(diag(TAB))/sum(TAB)
```

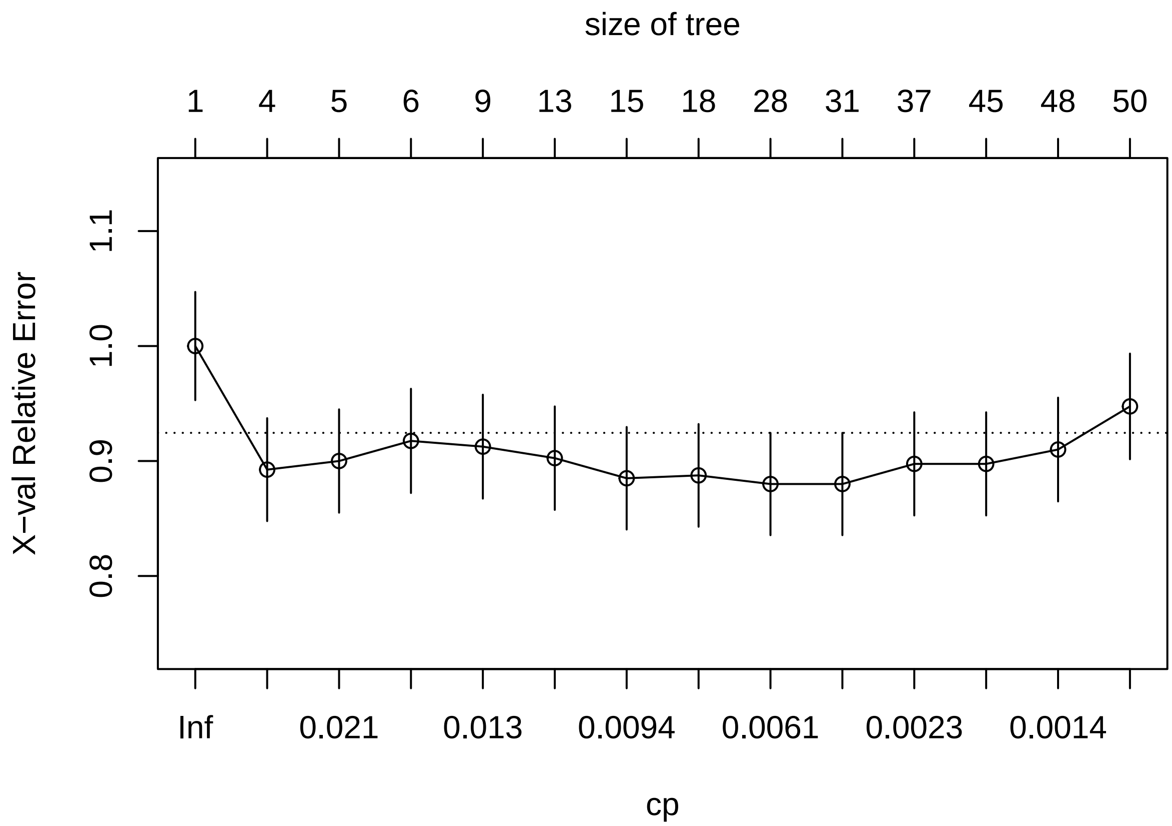
```
## [1] 0.1185113
```

As for miss-classification error we have 0.11. It is not big but for our YES class we have small number of true positives.

```r
printcp(tree1)
```

```
##
## Classification tree:
## rpart(formula = y ~ ., data = train, method = "class", cp = 0,
##     xval = 20)
##
## Variables actually used in tree construction:
##  [1] age       balance   campaign  contact   day       duration  education
##  [8] housing   job       marital   month     pdays     poutcome  previous
##
## Root node error: 400/3500 = 0.11429
##
## n= 3500
##
##           CP nsplit rel error xerror     xstd
## 1  0.0458333      0    1.0000 1.0000 0.047056
## 2  0.0250000      3    0.8625 0.8925 0.044762
## 3  0.0175000      4    0.8375 0.9000 0.044929
## 4  0.0141667      5    0.8200 0.9175 0.045313
## 5  0.0112500      8    0.7775 0.9125 0.045203
## 6  0.0100000     12    0.7275 0.9025 0.044984
## 7  0.0087500     14    0.7075 0.8850 0.044595
## 8  0.0075000     17    0.6750 0.8875 0.044651
## 9  0.0050000     27    0.5975 0.8800 0.044483
## 10 0.0025000     30    0.5825 0.8800 0.044483
## 11 0.0021875     36    0.5675 0.8975 0.044873
## 12 0.0016667     44    0.5500 0.8975 0.044873
## 13 0.0012500     47    0.5450 0.9100 0.045149
## 14 0.0000000     49    0.5425 0.9475 0.045959
```

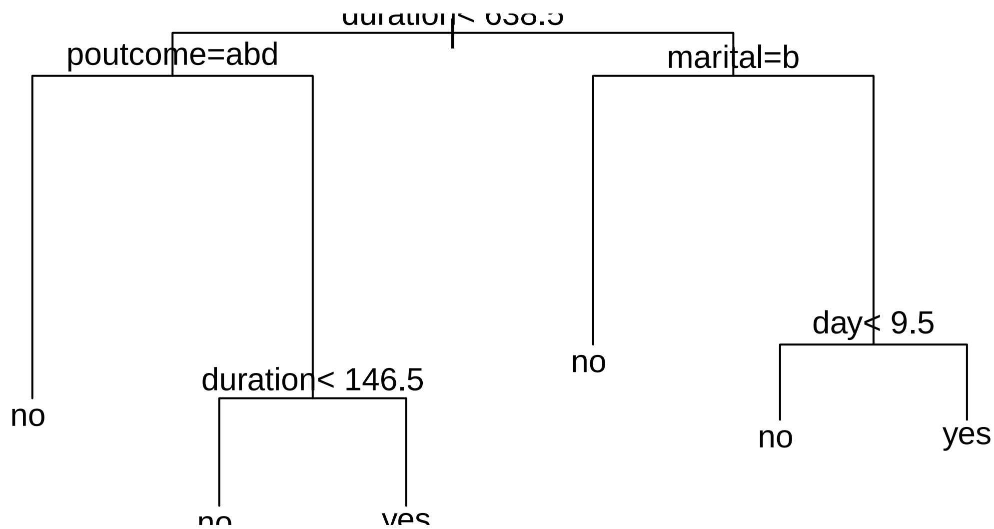
```r
plotcp(tree1)
```



For the cross validation we can see how our error changes with changing the cp complexity parameter. I will take cp=0.016 as value.

```
pruned <- prune(tree1, cp=0.016)
```

```
plot(pruned)
text(pruned)
```



We can see how our tree has less splits and it is way more simple than the previous one.

```
predict_prined <- predict(pruned, test, type="class")

(TAB <- table(test$y,predict_prined))
```

```
##      predict_prined
##         no yes
##   no   883  17
##   yes   90  31
```

```
1-sum(diag(TAB))/sum(TAB)
```

```
## [1] 0.1047992
```

And as result we see that we have less miss-classification error but we have more falsely predicted NO classes. The miss-classification error reduces because NO is the majority class and that's why this error is miss-leading. In order for our model to classify better the minor class we can try to oversample it or to apply cost function based on our YES class.

Random Forest

Basic model

```
rf <- randomForest(y ~., data = train)
rf
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 10.57%
## Confusion matrix:
##       no yes class.error
## no  3022  78  0.02516129
## yes  292 108  0.73000000
```

```
predict_rf <- predict(rf, test, type="class")

(TAB <- table(test$y,predict_rf))
```

```
##      predict_rf
##         no yes
##   no   885  15
##   yes   88  33
```

```
1-sum(diag(TAB))/sum(TAB)
```

```
## [1] 0.1008815
```

We can see for the first Random Forest model we get 0.10 miss-classification error. A little bit less than our descision tree model from above.

Importance

```
rf1 <- randomForest(y ~., data = train, importance=TRUE)
rf1
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = train, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
```

```
##         OOB estimate of  error rate: 10.51%
## Confusion matrix:
##       no yes class.error
## no  3023  77  0.02483871
## yes  291 109  0.72750000
```

```
predict_rf1 <- predict(rf1, test, type="class")

(TAB <- table(test$y,predict_rf1))
```

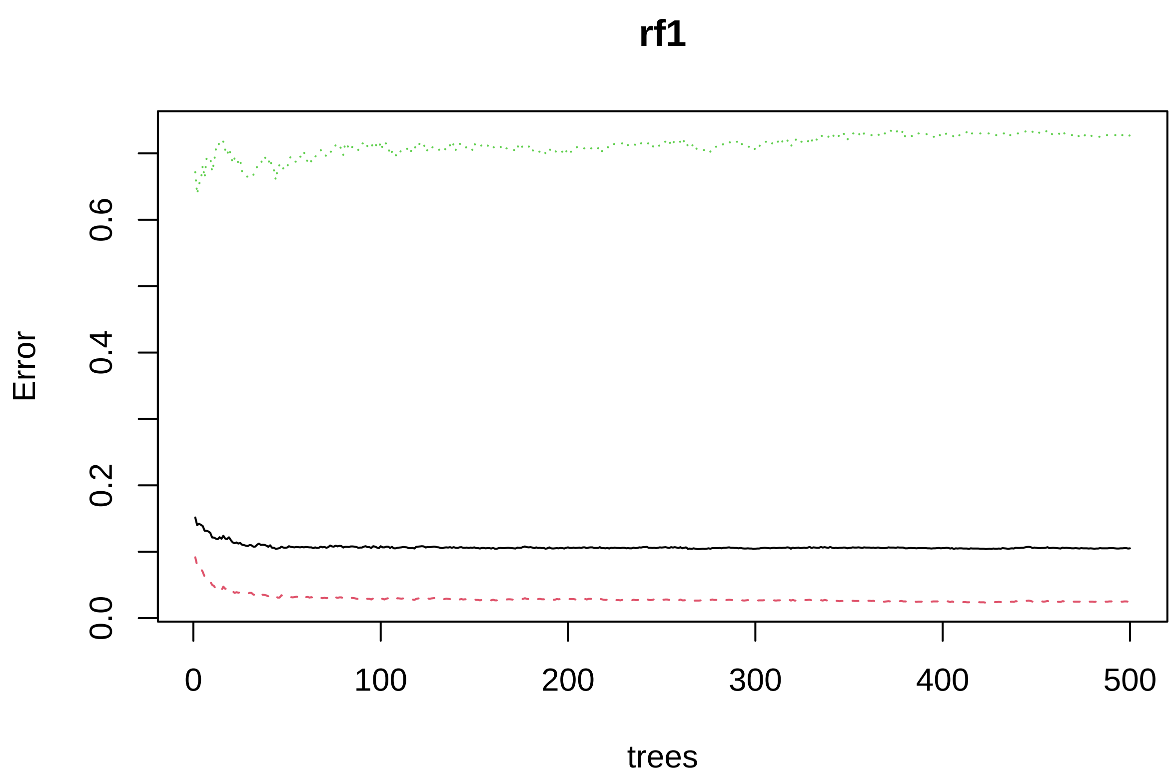
```
##      predict_rf1
##        no yes
##   no  887  13
##   yes  86  35
```

```
1-sum(diag(TAB))/sum(TAB)
```

```
## [1] 0.09696376
```

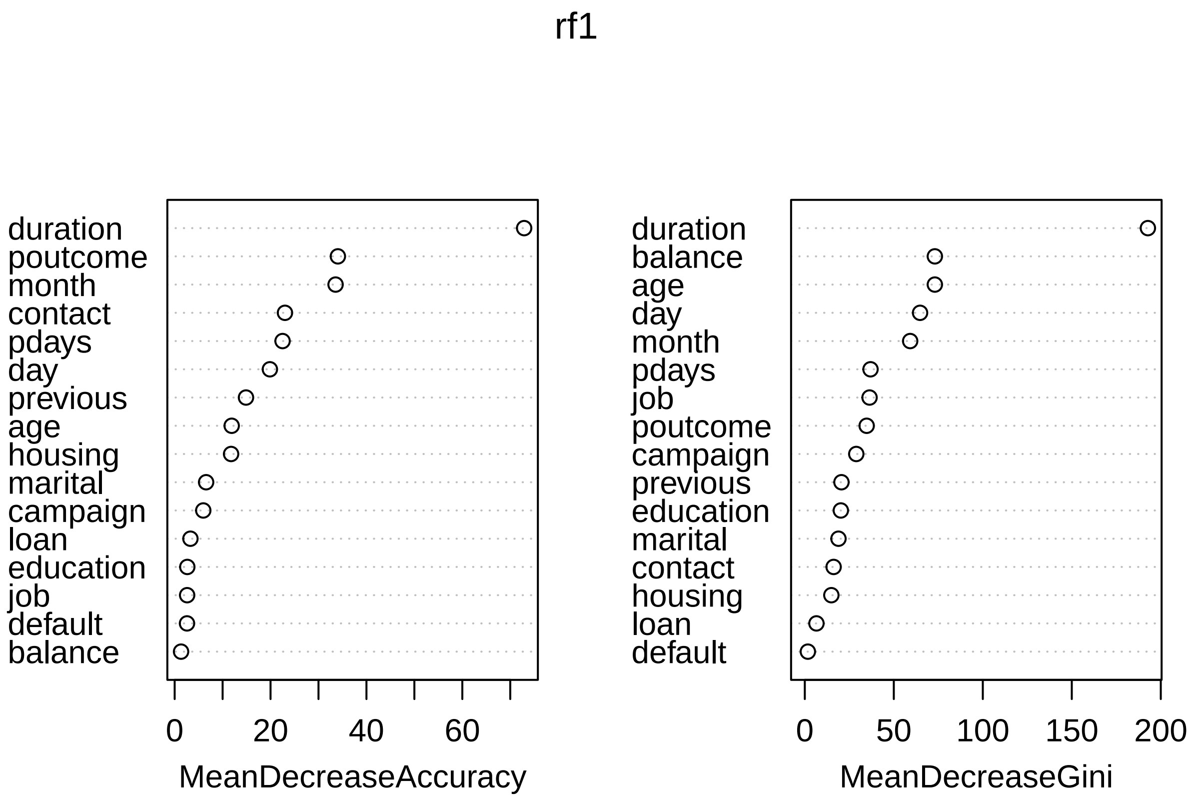
With importance=True we get even better results.

```
plot(rf1)
```



We can see the developments of the miss-classification error of both classes with each tree we build. We can see the lowest line is the class **no** and the highest is class **yes**. The middle one shows us the average error. We can see that both of them are stable over time and 500 trees should be enough.

```
varImpPlot(rf1)
```



We can see based on the two error measures. The top once are most important attributes. When we compare them we can see that they are not in the same order but duration is on both the first place. Helps us to interpret the data. For example balance attribute is on the second place in MeanDecreaseGini and the last on the MeanDecreaseAccuracy.

Improve Random Forest

Sample size

```
samp <- c(800, 1000, 1200, 1500, 1800)

for (s in samp) {
  rf2 <- randomForest(y ~., data = train, importance=TRUE, sampsize=s)

  predict_rf2 <- predict(rf2, test, type="class")

  print((TAB <- table(test$y,predict_rf2)))

  print(1-sum(diag(TAB))/sum(TAB))
}
```

```
##      predict_rf2
```

```
##        no yes
##   no  893   7
##   yes  97  24
## [1] 0.1018609
##      predict_rf2
##        no yes
##   no  891   9
##   yes  94  27
## [1] 0.1008815
##      predict_rf2
##        no yes
##   no  890  10
##   yes  93  28
## [1] 0.1008815
##      predict_rf2
##        no yes
##   no  889  11
##   yes  90  31
## [1] 0.09892262
##      predict_rf2
##        no yes
##   no  890  10
##   yes  91  30
## [1] 0.09892262
```

With that technique we couldn't manage to reduce our miss-classification error nor the True positives.

Classwt

```
wy = sum(train$y=="yes")/length(train$y)
wy
```

```
## [1] 0.1142857
```

```
wn = 1
```

```
weight <- c(0.15, 0.80, 1.5, 5, 10, 15, 20)

for (w in weight) {
  rf3 <- randomForest(y ~., data = train, importance=TRUE, classwt = c("yes"=1, "no"=w))

  predict_rf3 <- predict(rf3, test, type="class")

  print("For Weight:")
  print(w)
  print((TAB <- table(test$y,predict_rf3)))

  print(1-sum(diag(TAB))/sum(TAB))
}
```

```
## [1] "For Weight:"
```

```
## [1] 0.15
##      predict_rf3
##        no yes
##   no  894   6
##   yes 106  15
## [1] 0.1096964
## [1] "For Weight:"
## [1] 0.8
##      predict_rf3
##        no yes
##   no  892   8
##   yes  99  22
## [1] 0.1047992
## [1] "For Weight:"
## [1] 1.5
##      predict_rf3
##        no yes
##   no  891   9
##   yes 101  20
## [1] 0.1077375
## [1] "For Weight:"
## [1] 5
##      predict_rf3
##        no yes
##   no  886  14
##   yes  91  30
## [1] 0.1028404
## [1] "For Weight:"
## [1] 10
##      predict_rf3
##        no yes
##   no  886  14
##   yes  86  35
## [1] 0.09794319
## [1] "For Weight:"
## [1] 15
##      predict_rf3
##        no yes
##   no  882  18
##   yes  81  40
## [1] 0.09696376
## [1] "For Weight:"
## [1] 20
##      predict_rf3
##        no yes
##   no  883  17
##   yes  81  40
## [1] 0.09598433
```

Here we can see the True Positives raised form 35 (our initial model from the previous step) to 42 now. But our True Negatives reduced from 887 to 880. So it is a tried-off to boost the smaller class.

Cutoff technique

```
rf4 <- randomForest(y ~., data = train, importance=TRUE, cutoff=c(0.7, 0.3))

predict_rf4 <- predict(rf4, test, type="class")

print((TAB <- table(test$y,predict_rf4)))
```

```
##      predict_rf4
##        no yes
##   no  841  59
##   yes  52  69
```

```
print(1-sum(diag(TAB))/sum(TAB))
```

```
## [1] 0.1087169
```

With Cutoff we have much better results on the True Positives. From 42 to 71.The miss-classification error is a little bit higher and True negatives are a little bit less.

Strata technique

```
rf5 <- randomForest(y ~., data = train, importance=TRUE, strata=y)

predict_rf5 <- predict(rf5, test, type="class")

print((TAB <- table(test$y,predict_rf5)))
```

```
##      predict_rf5
##        no yes
##   no  885  15
##   yes  87  34
```

```
print(1-sum(diag(TAB))/sum(TAB))
```

```
## [1] 0.09990206
```

We don't have much improvement with stratifying the data. Still we have good miss-classification error but underrepresented **YES** class still suffers.

As conclusion I can say that **Classwt** and **Cutoff** techniques did the best job. We managed to score less miss-classification error for True Positives and achieving less accurate True Negatives, but those technique helps when we have unbalanced data to improve the underrepresented class.