Exercise 9

Tobias Raidl, 11717659

2023-01-15

Contents


```r
library(ROCit)
```

```
## Warning: Paket 'ROCit' wurde unter R Version 4.3.2 erstellt
```

```r
data(Loan)

df = Loan
set.seed(11717659)
sample <- sample(c(TRUE, FALSE), nrow(df), replace = TRUE, prob = c(0.7, 0.3))
train <- df[sample, ]
test <- df[!sample, ]
summary(train$Status)
```

```
##  CO  FP
##  96 542
```

a)

Compute an initial tree $T0$ (see help(rpart) or lecture notes).

```r
library(rpart)
```

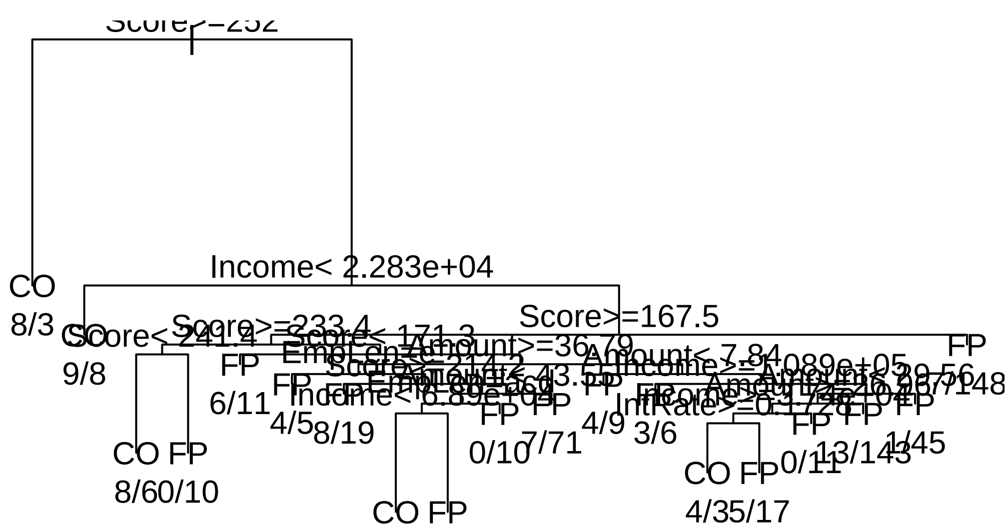
```
## Warning: Paket 'rpart' wurde unter R Version 4.3.2 erstellt
```

```r
t0 = rpart(Status ~ ., data = train, method = "class", cp = 0.001, xval = 20)
par(mfrow = c(1, 2), xpd = NA)
```

b)

Visualize the tree with the function plot() and text(), and interpret the results.

```r
plot(t0)
text(t0, use.n = TRUE)
```



c)

Predict the class variable for the test set (see help(predict.rpart) or lecture notes). Show the confusion table and report the balanced accuracy.

```r
get_balanced_accuracy = function(gt, pred) {
    # (sensitivity/specificity) / 2
    conf_mat = table(gt, pred)
    fn = conf_mat[2, 1]
    fp = conf_mat[1, 2]
    tp = conf_mat[1, 1]
    tn = conf_mat[2, 2]
    sensitivity = tp/(tp + fn)
    specificity = tn/(tn + fp)
    return(list(val = 0.5 * (sensitivity + specificity), conf_mat = conf_mat))
}

t0.pred = predict(t0, test, type = "class")
cat(paste("balanced accuracy:", get_balanced_accuracy(test$Status, t0.pred)))
```

```
## balanced accuracy: 0.545177045177045 balanced accuracy: c(6, 22, 29, 205)
```

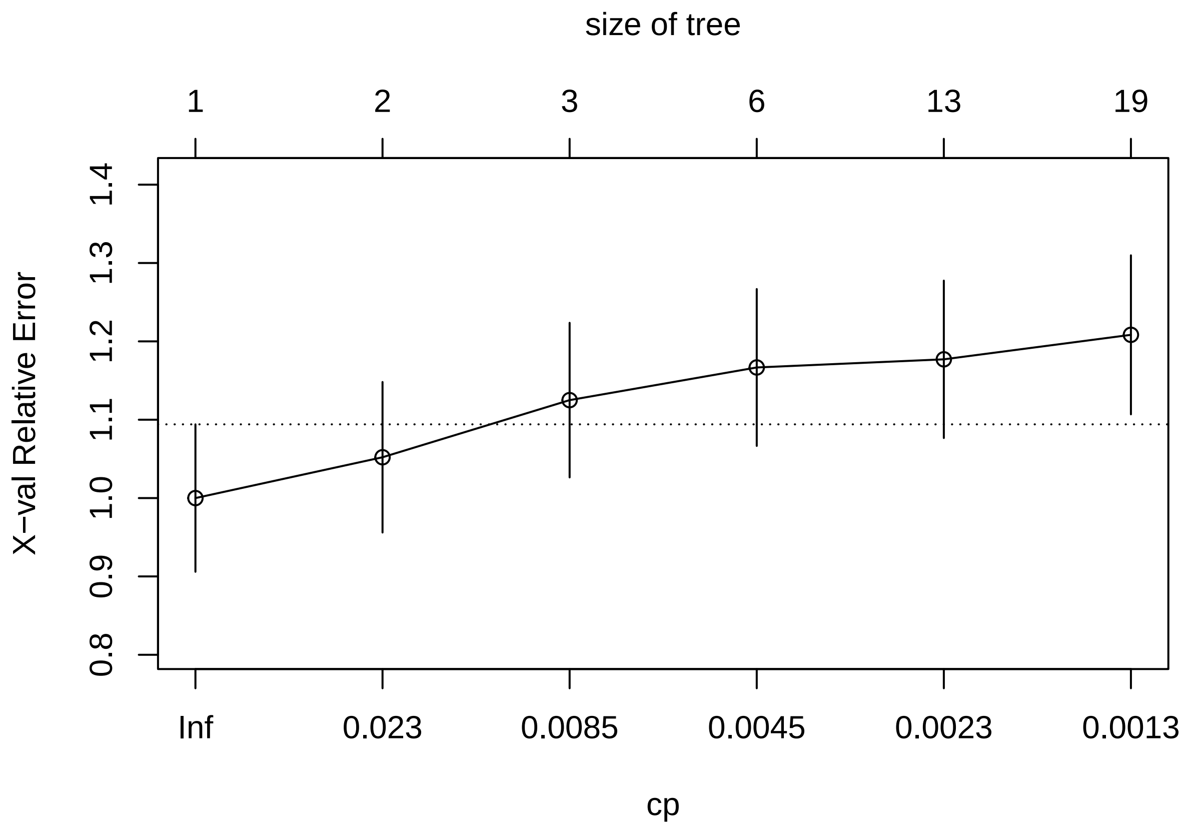
d)

Show and interpret results of cross-validation obtained by using printcp() und plotcp(). What is the optimal tree complexity?

```r
printcp(t0)
```

```
##
## Classification tree:
## rpart(formula = Status ~ ., data = train, method = "class", cp = 0.001,
##     xval = 20)
##
## Variables actually used in tree construction:
## [1] Amount  EmpLen  Income  IntRate Score
##
## Root node error: 96/638 = 0.15047
##
## n= 638
##
##          CP nsplit rel error xerror     xstd
## 1 0.0520833      0   1.00000 1.0000 0.094071
## 2 0.0104167      1   0.94792 1.0521 0.096043
## 3 0.0069444      2   0.93750 1.1250 0.098666
## 4 0.0029762      5   0.91667 1.1667 0.100097
## 5 0.0017361     12   0.89583 1.1771 0.100447
## 6 0.0010000     18   0.88542 1.2083 0.101481
```

```r
plotcp(t0, upper = "size")
```

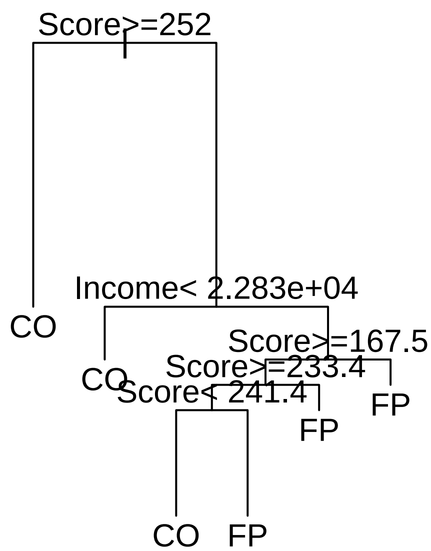


The optimal tree complexity is 0.0003

e)

Prune the tree T_0 to the optimal complexity using prune(). Visualize und interpret the results.

```r
t1 = prune(t0, cp = 0.003)
par(mfrow = c(1, 2), xpd = NA)
plot(t1)
text(t1)
```



f)

Predict the class variable for the test set, show the confusion table, and report the balanced accuracy. Do we observe any improvement?

```
t1.pred = predict(t1, test, type = "class")
bal_acc = get_balanced_accuracy(test$Status, t1.pred)
bal_acc$conf_mat
```

```
##     pred
## gt    CO  FP
##   CO   5  30
##   FP  11 216
```

```
cat(paste("balanced accuracy:", bal_acc$val))
```

```
## balanced accuracy: 0.595274390243902
```

We can observe an improvement in balanced accuracy by ~5%

g)

A simple way to improve the balanced accuracy could be to make use of the argument weights within rpart(). Try it out and report the results.

```r
CO_weight = 1/nrow(subset(train, Status == "CO"))/nrow(train)
FP_weight = 1/nrow(subset(train, Status == "FP"))/nrow(train)

weights <- ifelse(train$Status == "CO", CO_weight, FP_weight)

t3 = rpart(Status ~ ., data = train, method = "class", cp = 0.003, xval = 20, weights =
↪  weights)
t3.pred = predict(t3, test, type = "class")
bal_acc = get_balanced_accuracy(test$Status, t3.pred)
bal_acc$conf_mat
```

```
##     pred
## gt    CO  FP
##   CO  16  19
##   FP  74 153
```

```r
cat(paste("balanced accuracy:", bal_acc$val))
```

```
## balanced accuracy: 0.533656330749354
```

Results are worse.

2

a)

Use Random Forests to classify the training data and predict the class variable for the test data. Report the resulting cofusion table and the balanced accuracy.

```r
library(randomForest)
```

```
## Warning: Paket 'randomForest' wurde unter R Version 4.3.2 erstellt

## randomForest 4.7-1.1

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

```r
set.seed(11717659)
rf0 = randomForest(Status ~ ., data = train)
rf0.pred = predict(rf0, test)
bal_acc = get_balanced_accuracy(test$Status, rf0.pred)
bal_acc$conf_mat
```

```
##     pred
## gt    CO  FP
##   CO   1  34
##   FP   6 221
```

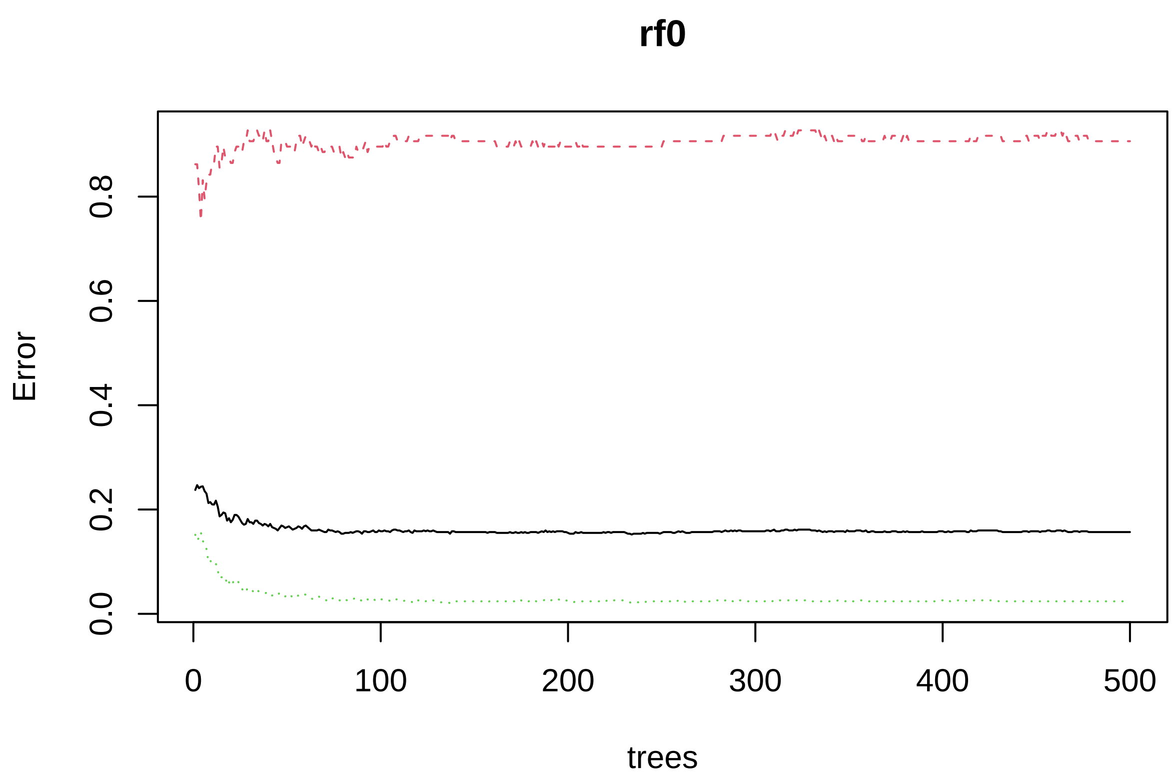
```r
cat(paste("balanced accuracy:", bal_acc$val))
```

```
## balanced accuracy: 0.504761904761905
```

b)

Plot the result object with plot() and interpret the plot

```r
plot(rf0)
```



Plot the error rates of the randomForest I think the red line corresponds to the error for class CO and the green line to the error of FP, for the different number of trees.

c)

Try to improve the balanced accuracy with different strategies: – Modify the parameter sampsize in the randomForest() function. What is it doing? – Modify the parameter classwt in the randomForest() function. What is it doing? – Modify the parameter cutoff in the randomForest() function. What is it doing? Which approach leads to the overall best solution?

```r
set.seed(11717659)
rf1 = randomForest(Status ~ ., data = train, sampsize = c(50, 10), classwt = c(0.1,
    0.5), cutoff = c(0.6, 0.4), importance = TRUE)
```

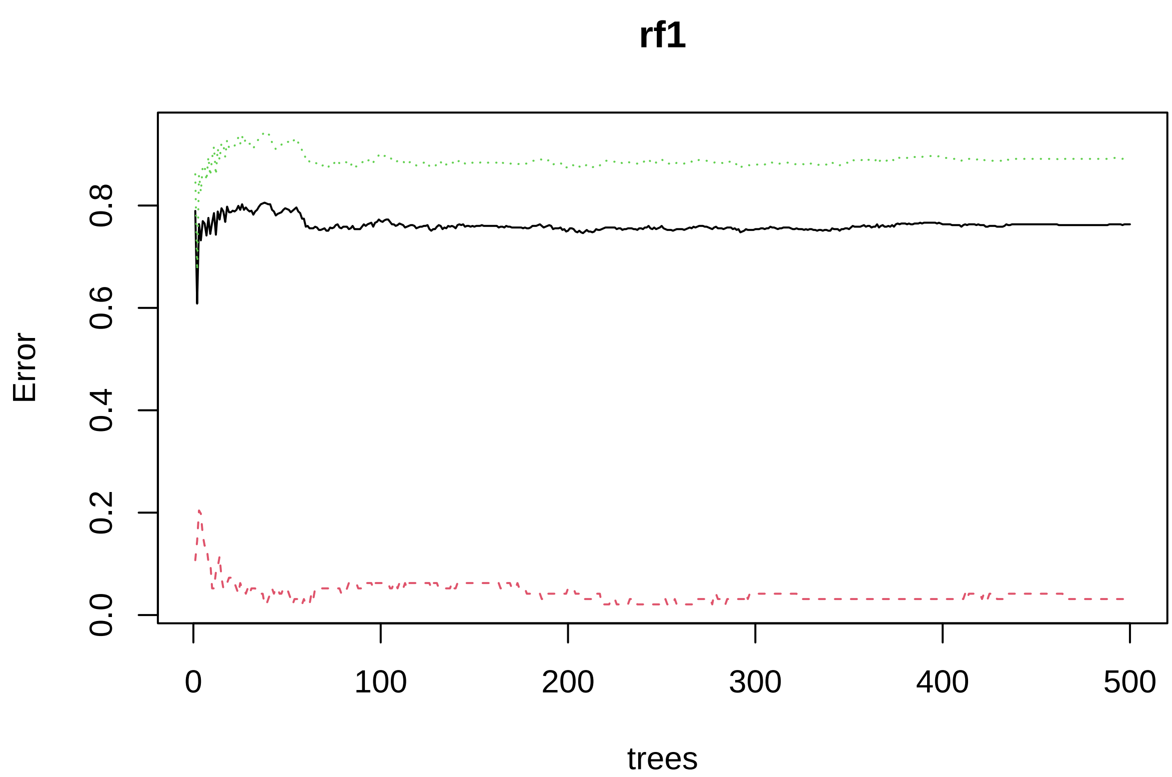
```r
rf1.pred = predict(rf1, test)
bal_acc = get_balanced_accuracy(test$Status, rf1.pred)
bal_acc$conf_mat
```

```
##     pred
## gt    CO  FP
##   CO  35   0
##   FP 201  26
```

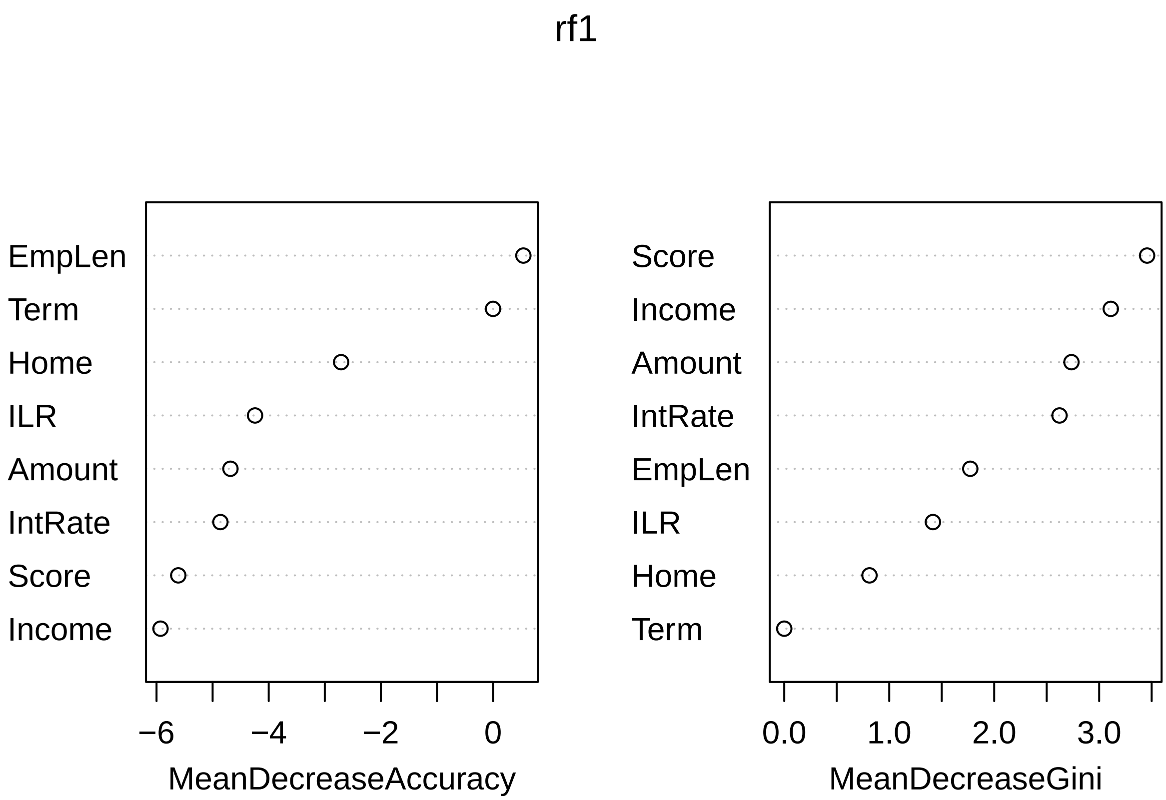
```r
cat(paste("balanced accuracy:", bal_acc$val))
```

```
## balanced accuracy: 0.574152542372881
```

```r
plot(rf1)
```




```r
varImpPlot(rf1)
```



- sampsize: Sizes of sample to draw. Parameter takes vector where each value corresponds to the numbers drawn per class. (Stratification)
- classwt: Priors of the classes.
- cutoff: A vector of length equal to number of classes. The 'winning' class for an observation is the one with the maximum ratio of proportion of votes to cutoff. Default is 1/k where k is the number of classes (i.e., majority vote wins).

I tried multiple combinations for the sampsize and [50, 10] had the best balanced accuracy. This parameter seems to be rather sensitive, as minor changes can change the balanced accuracy massively. The cutoff [0.6, 0.4] results in the best balanced accuracy. For classwt the priors for each class should be chosen and are per default 1/2 for us due to 2 classes. I changed it to [0.1, 0.5] because we have ~100 appearances of class CO and ~500 appearances of class FP.

Plot the error rates of the random forest. I think the red line corresponds to the error for class CO and the green line to the error of FP, for the different number of trees. The varImpPlot displays the importance of the variables.