```r
#### Lab 2 - Assignment 2 - Analysis of credit scoring
### Libraries
library(tree)
library(e1071)
library(MASS)

### Setup
data.credit = read.csv("creditscoring.csv")
n = dim(data.credit)[1]

## Task 1 - Divide data into training, validation and test as 50/25/25
# Train
set.seed(12345)
ids.train = sample(1:n, floor(n*0.5))
train = data.credit[ids.train,]

# Validation
remainder = setdiff(1:n, ids.train)
set.seed(12345)
ids.validation = sample(remainder, floor(n*0.25))
validation = data.credit[ids.validation,]

# Test
ids.test = setdiff(remainder, ids.validation)
test = data.credit[ids.test,]

#set.seed(12345)
#ids = sample(1:n, n)                                      # random
order of observations
#train = data[ids[1:floor(n*0.5)],]                        # 50% of the
data
#validation = data[ids[(floor(n*0.5)+1):floor(n*0.75)],] # 25% of data
#test = data[ids[(floor(n*0.75)+1):n],]                    # 25% of the
data

# Y-values
y.train = train$good_bad
y.validation = validation$good_bad
y.test = test$good_bad

### Functions

## Returns the confusion matrix
confusion_matrix = function(y, yfit) {
  return(table(y, yfit, dnn=c("TRUE", "PRED")))
}

mcr2 = function(y,yfit){
  return(mean(abs(y-yfit)))
}

## Returns the missclassifcationrate
mcr = function(cm){
  return(1-sum(diag(cm))/sum(cm))
}

### Implementation
## Task 2 - Decision tree, Deviance vs. Gini index
# Fit trees
tree.deviance = tree(formula=good_bad~., data=train, split="deviance")
tree.gini = tree(formula=good_bad~., data=train, split="gini")

# Plot trees
plot(tree.deviance)
text(tree.deviance, pretty=0)
plot(tree.gini)
text(tree.gini, pretty=0)

# Make predictions
yfit.train.deviance = predict(tree.deviance, train, type="class")
yfit.test.deviance = predict(tree.deviance, test, type="class")
yfit.train.gini = predict(tree.gini, train, type="class")
yfit.test.gini = predict(tree.gini, test, type="class")

# Confusion Matrixes
cm.train.deviance = confusion_matrix(y.train, yfit.train.deviance)
cm.test.deviance = confusion_matrix(y.test, yfit.test.deviance)
cm.train.gini = confusion_matrix(y.train, yfit.train.gini)
cm.test.gini = confusion_matrix(y.test, yfit.test.gini)

# Missclassification Rates
mcr.train.deviance = mcr(cm.train.deviance) # 0.212
mcr.test.deviance = mcr(cm.test.deviance) # 0.268
mcr.train.gini = mcr(cm.train.gini) # 0.238
mcr.test.gini = mcr(cm.test.gini) # 0.372
# Comment: 0.212<0.238 and 0.268<0.372, deviance performs better as a
measure of impurity

## Task 3 - Optimal Tree Depth
m = 14
deviance.train = rep(1:m)
deviance.validation = rep(1:m)

# Calculate deviance for # of leaves
for(i in 2:m){
  tree.pruned = prune.tree(tree.deviance, best=i)
  yfit.train = predict(tree.pruned, train, type="tree")
  yfit.validation = predict(tree.pruned, validation, type="tree")
  deviance.train[i] = deviance(yfit.train)
  deviance.validation[i] = deviance(yfit.validation)
}

# Plot deviance based on # of leaves
leaves = 2:14
plot(leaves, deviance.train[leaves], ylim=c(280, 580), type="l",
col="green",
     main="Deviance for given # of leaves", xlab="# of leaves",
ylab="deviance")
lines(leaves, deviance.validation[leaves], col="blue")
```

```r
# Locate best # of leaves
best = which.min(deviance.validation[leaves])+1
points(best, deviance.validation[best]) # Add to plot

# Best tree info
tree.best = prune.tree(tree.deviance, best=best)
plot(tree.best)
text(tree.best, pretty=0) # savings, duration and history used
yfit.test.best = predict(tree.best, test, type="class")
cm.test.best = confusion_matrix(y.test, yfit.test.best)
#       PRED
#TRUE    bad good
# bad     18   58
# good     6  168
mcr.test.best = mcr(cm.test.best) # 0.256
# Comment: 0.256 < 0.268, performs better than a full tree
(tree.deviance above)

## Task 4 - Naive Bayes classification
naive.bayes = naiveBayes(good_bad~., data=train)

# Make predictions
yfit.train.naive = predict(naive.bayes, train, type="class")
yfit.test.naive = predict(naive.bayes, test, type="class")

# Confusion matrixes
cm.train.naive = confusion_matrix(y.train, yfit.train.naive)
cm.test.naive = confusion_matrix(y.test, yfit.test.naive)

# Missclassification rates
mcr.train.naive = mcr(cm.train.naive) # 0.3
mcr.test.naive = mcr(cm.test.naive) # 0.316
# 0.3 > 0.256, naive bayes performs worse than the pruned tree

## Task 5 - Naive Bayes with Loss Matrix
# Loss Matrix:
#           PRED
# L = TRUE    bad good
#       bad     0   10
#      good     1    0
# Comment:
# Expensive to predict a client as good when in fact he/she is bad
# Costly but not as much to predict as bad when the client is good

# Get confidence of predictions
yfit.train.naive.raw = predict(naive.bayes, train, type="raw")
yfit.test.naive.raw = predict(naive.bayes, test, type="raw")

# Make predictions
p.good.train = yfit.train.naive.raw[,2]
p.bad.train = yfit.train.naive.raw[,1]
p.good.test = yfit.test.naive.raw[,2]
p.bad.test = yfit.test.naive.raw[,1]
# p.good * L.good.bad > p.bad * L.bad.good -> 1*p.good > 10*p.bad  ->
```

```
predict good, otherwise bad
yfit.train.naive.loss = ifelse(p.good.train>10*p.bad.train, TRUE,
FALSE)
yfit.test.naive.loss = ifelse(p.good.test>10*p.bad.test, TRUE, FALSE)

# Confusion matrixes
cm.train.naive.loss = confusion_matrix(y.train, yfit.train.naive.loss)
#       PRED
#TRUE   FALSE TRUE
#bad    137   10
#good   263   90
cm.test.naive.loss = confusion_matrix(y.test, yfit.test.naive.loss)
#       PRED
#TRUE   FALSE TRUE
#bad     71    5
#good   122   52

# Missclassification rates
mcr.train.naive.loss = mcr(cm.train.naive.loss) # 0.546
mcr.test.naive.loss = mcr(cm.test.naive.loss) # 0.508
```