```
# Authors: Grzegorz Protaziuk, Robert Bembenik
# Script EDAMI lab5

#Subject: Classification
#1. Classification with "C5.0" library
#2. Classification with rpart library
#3. Random forest


library(gmodels) #results analysis
library(Hmisc) #results analysis
library(caret)
library(rpart) # rpart() - decision tree classifier
library(rpart.plot)
library(e1071)
library(C50) # C5 classifer
library(randomForest)
library(datasets)

#################################################################
#    data preprocessing                                          #
#################################################################

download.file('http://archive.ics.uci.edu/ml/machine-learning-
databases/car/car.data', 'car.data')

#date reading
cars = read.csv("car.data", header = FALSE,
                col.names = c('buying', 'maint', 'doors', 'persons',
'lug_boot','safety', "category"),
                stringsAsFactors = TRUE )

summary(cars)

set.seed(7777)

#creating training and test datasets
sam <- sample(2, nrow(cars), replace=TRUE, prob=c(0.7, 0.3))
sam
carTrain1 <- cars[sam==1,]
carTest1 <- cars[sam==2,]

#class distribution in sets
prop.table(table(carTrain1$category))
prop.table(table(carTest1$category))

#creating training and test datasets
?createDataPartition
idTrainData <- unlist(createDataPartition(cars$category,p=0.7))
#str(idTrainData)

carTrain <-cars[idTrainData,]
carTest <-cars[-idTrainData,]

#class distribution in sets
prop.table(table(carTrain$category))
prop.table(table(carTest$category))
```

```
table(carTrain$category)
table(carTest$category)

#################################################################
# 1.  C5.0 classifier                                           #
#################################################################

str(carTrain)

?C5.0
#model building - a decision tree
car_C50 <- C5.0(carTrain[,-7], carTrain$category)
summary(car_C50)
plot(car_C50)

#quality of classification for training data
car_c50_trainPred <- predict(car_C50, carTrain)

?CrossTable
CrossTable(car_c50_trainPred, carTrain$category, prop.chisq = FALSE,prop.c =
FALSE,
           prop.r = FALSE, dnn = c('predicted class', 'actual class'))

?confusionMatrix
confusionMatrix(car_c50_trainPred, carTrain$category, mode="everything")

#quality of classification for test data
car_c50_testPred <- predict(car_C50, carTest)
CrossTable(car_c50_testPred, carTest$category, prop.chisq = FALSE,prop.c =
FALSE,
           prop.r = FALSE, dnn = c('predicted class', 'actual class'))

confusionMatrix(car_c50_testPred, carTest$category, mode="everything")

#model building - rules
car_C50R <- C5.0(carTrain[,-7], carTrain$category,  rules = TRUE)
summary(car_C50R)

#quality of classification for test data
car_c50_testPred <- predict(car_C50R, carTest)
CrossTable(car_c50_testPred, carTest$category, prop.chisq = FALSE,prop.c =
FALSE,
           prop.r = FALSE, dnn = c('predicted class', 'actual class'))

confusionMatrix(car_c50_testPred, carTest$category, mode="everything")

#Ensemble classifier (boosting)
download.file('https://staff.elka.pw.edu.pl/~rbembeni/dane/churnTrain.csv','c
hurnTrain.csv')
churnTrain <- read.csv("churnTrain.csv",sep = ";", header = TRUE)
download.file('https://staff.elka.pw.edu.pl/~rbembeni/dane/churnTest.csv','ch
urnTest.csv')
churnTest <- read.csv("churnTest.csv",sep = ";", header = TRUE)

churnTrain <- as.data.frame(unclass(churnTrain),                       #
Convert all columns to factor
```

```
                      stringsAsFactors = TRUE)
churnTest <- as.data.frame(unclass(churnTest),                    # Convert
all columns to factor
                      stringsAsFactors = TRUE)

summary(churnTrain$churn)
summary(churnTest$churn)
summary(churnTrain)
str(churnTrain)
#tree
churn_C50 <- C5.0(churnTrain[, -20], churnTrain$churn)
churn_C50_testPred = predict(churn_C50, churnTest)
confusionMatrix(churn_C50_testPred, churnTest$churn, mode="everything")

#ensemble tree
churn_C50B <- C5.0(churnTrain[, -20], churnTrain$churn,trials = 10)
churn_C50B_testPred = predict(churn_C50B, churnTest)
confusionMatrix(churn_C50B_testPred, churnTest$churn, mode="everything")
summary(churn_C50B)

#TASK: Build C.50 classifier for the titanic dataset
# 1. Create training and test data sets
# 2. Build a classifier
# 3. Determine quality of the classifier
# 4. Determine if Johny and Sylvia survived according to the classifier.


download.file('https://staff.elka.pw.edu.pl/~rbembeni/dane/titanic.csv','tita
nic.csv')
titanic <- read.csv("titanic.csv", stringsAsFactors = TRUE)
titanic <- titanic[,-1]

Johny <- titanic[1,]
Johny$gender <- "male"
Johny$age <- 8
Johny$class <- "1st"
Johny$embarked <- "Southampton"
Johny$country <- "Norway"
Johny$fare = 25
Johny$sibsp <- 0
Johny$parch <- 0

Sylvia <-Johny
Sylvia$gender <- "female"
Sylvia$age <- 20
Sylvia$class <- "3rd"
Sylvia$embarked <- "Cherbourg"
Sylvia$country <- "England"
Sylvia$fare = 25
Sylvia$sibsp <- 0
Sylvia$parch <- 0




#####################################################################3333333
#dane iris (https://archive.ics.uci.edu/ml/datasets/Iris)
```

```
data(iris)
#str(iris)
#View(iris)

idTrainData1 <- unlist(createDataPartition(iris$Species,p=0.7))
#str(idTrainData)

irisTrain <-iris[idTrainData1,]
irisTest <-iris[-idTrainData1,]

table(irisTest$Species)

#setting the class attribute
irisFormula <-  Species ~ Sepal.Length + Sepal.Width + Petal.Length +
Petal.Width

################################################################
# 2.  rpart - recursive partitioning trees                      #
################################################################

?rpart
# tree building
iris_rpart <- rpart(irisFormula,  method="class", data=irisTrain)
print(iris_rpart)

#  CP - complexity parameter: serves as a penalty to control the size of the
tree;
#the greater the CP value, the fewer the number of splits there are
#  rel error represents the average deviance of the current tree divided
#by the average deviance of the null tree
#  xerror value represents the relative error estimated by a 10-fold
classification
#  xstd stands for the standard error of the relative error

summary(iris_rpart)

?rpart.plot
#plot(iris_rpart, main="Classification for Iris")
#text(iris_rpart, use.n=TRUE, all=TRUE, cex=.7)
rpart.plot(iris_rpart, main="Classification for Iris")

?prp
prp(iris_rpart, faclen = 0, cex = NULL, extra = 1, main="Classification for
Iris")


#training data classification - confusion matrix
iris_rpat_trainPred = predict(iris_rpart,irisTrain,type = "class")
table(iris_rpat_trainPred, irisTrain$Species)
confusionMatrix(iris_rpat_trainPred, irisTrain$Species, mode="everything")

#test data classification - confusion matrix
iris_rpat_testPred = predict(iris_rpart,irisTest,type = "class")
table(iris_rpat_testPred, irisTest$Species)

confusionMatrix(iris_rpat_testPred, irisTest$Species, mode="everything")
```

```
#Loss matrix - a row the actual class, a column - predicted class
#(The loss matrix must have zeros on the diagonal and positive off-diagonal
elements)
lossM=matrix(c(0,1,1,1,0,8,1,1,0), byrow=TRUE, nrow=3)
lossM
iris_rpartLM <-  rpart(irisFormula,  method="class", data=irisTrain, parms =
list(loss = lossM ))

#training data classification - confusion matrix
iris_rpatLM_trainPred = predict(iris_rpartLM,irisTrain,type = "class")

confusionMatrix(iris_rpatLM_trainPred, irisTrain$Species, mode="everything")

table(iris_rpatLM_trainPred, irisTrain$Species)
CrossTable(irisTrain$Species, iris_rpatLM_trainPred, prop.chisq =
FALSE,prop.c = FALSE, prop.r = FALSE, dnn = c('actual','predicted'))

#test data classification - confusion matrix
iris_rpatLM_testPred = predict(iris_rpartLM,irisTest,type = "class")
confusionMatrix(iris_rpatLM_testPred, irisTest$Species, mode="everything")

table(iris_rpatLM_testPred, irisTest$Species)

#changing the values of parameters

?rpart.control

rpControl = rpart.control(minbucket = 30, maxDepth = 1);
rpTree <- rpart(irisFormula,  method="class", data=irisTrain,
                control =rpControl,
                parms = list(split = "information" ))
rpart.plot(rpTree, main="Classification for Iris")

iris_rpartS = predict(rpTree,irisTrain,type = "class")
table(iris_rpartS, irisTrain$Species)


#tree pruning

#The cost complexity pruning algorithm considers the cost complexity of a
tree to be a function of
# the number of leaves in the tree and the error rate of the tree (where the
error rate is the
# percentage of tuples misclassified by the tree). It starts from the bottom
of the tree. For
# each internal node, N, it computes the cost complexity of the subtree at N,
and the cost
# complexity of the subtree at N if it were to be pruned (i.e., replaced by a
leaf node). The
# two values are compared. If pruning the subtree at node N would result in a
smaller cost
# complexity, then the subtree is pruned. Otherwise, it is kept.
# A pruning set of class-labeled tuples is used to estimate cost complexity.
This set is
# independent of the training set used to build the unpruned tree and of any
test set used
```

```
# for accuracy estimation. The algorithm generates a set of progressively
pruned trees. In
# general, the smallest decision tree that minimizes the cost complexity is
preferred.

#the minimal  cross-validation error
min(iris_rpartLM$cptable[,"xerror"])
which.min(iris_rpartLM$cptable[,"xerror"])
rpTree.cp=iris_rpartLM$cptable[3,"CP"]
rpTree.cp
?prune
iris_rpartLM_Pruned<- prune(iris_rpartLM, cp = rpTree.cp)


#iris_rpartLM_Pruned <- prune(iris_rpartLM, cp =
rpTree$cptable[which.min(rpTree$cptable[,"xerror"]),"CP"])

rpart.plot(iris_rpartLM, main="Classification for Iris")
rpart.plot(iris_rpartLM_Pruned, main="Classification for Iris - pruned")

################################################################
#  3. randomForest                                             #
################################################################

?randomForest
car_Forest = randomForest(category~., data = carTrain, importance = TRUE,
nodesize = 10, mtry = 4, ntree = 300 )

#nodesize = minimal number of objects in a node
#mtry - the number of randomly selected attributes for searching the best
test split in nodes
#ntree -  number of trees in a forest
#importance - calculation of attriubte importance

summary(carTrain)
print(car_Forest)
plot(car_Forest)
legend("top", colnames(car_Forest$err.rate),col=1:5,cex=0.8,fill=1:5)

?importance
round(importance(car_Forest, type = 2),2)
round(importance(car_Forest, type = 1),2)


car_Forest_testPred = predict (car_Forest, newdata = carTest[-7])

confusionMatrix(car_Forest_testPred, carTest$category, mode = "everything")


#looking for the best values of parameters by means of K-fold validation
?trainControl
trControl <- trainControl(method = "cv", number = 10, search = "grid")

#arguments
#- method = "cv": The method used to resample the dataset.
#- number = n: Number of folders to create
```

```
#- search = "grid": Use the search grid method. For randomized method, use
"grid"

?train
tuneGrid <- expand.grid(mtry = c(1:6))
tuneGrid

names(getModelInfo())
modelLookup("C5.0")
#getModelInfo("rpart")

car_Frestores_mtry <- train(category~.,  data = carTrain,
                            method = "rf",
                            metric = "Accuracy",
                            tuneGrid = tuneGrid,
                            trControl = trControl,
                            importance = TRUE,    # randomForest function
parameter
                            nodesize = 10,        # randomForest function
parameter
                            ntree = 250)          ## randomForest function
parameter
print(car_Frestores_mtry)


starT <- proc.time()
treesModels <- list()
for (nbTree in c(5,10,25, 50, 100, 250, 500))
{
  car_F_maxtrees <- train(category~.,  data = carTrain,
                          method = "rf",
                          metric = "Accuracy",
                          tuneGrid = tuneGrid,
                          trControl = trControl,
                          importance = TRUE,
                          nodesize = 10,
                          ntree = nbTree)
  key <- toString(nbTree)
  treesModels[[key]] <- car_F_maxtrees
}

endT <- proc.time()
print(endT - starT)


?resamples
results_tree <- resamples(treesModels)
summary(results_tree)

#the final model
car_Forest2 = randomForest(category~., data = carTrain, importance = TRUE,
mtry = 6, ntree = 250, nodesize = 10)
print(car_Forest2)
plot(car_Forest2)
legend("top", colnames(car_Forest2$err.rate),col=1:5,cex=0.8,fill=1:5)

car_Forest2_testPred = predict (car_Forest, newdata = carTest[-7])
```

```
confusionMatrix(car_Forest2_testPred, carTest$category, mode = "everything")

varImpPlot(car_Forest2)

################################################
#Comparision of classifiers
###############################################
car_rpart <- rpart(category~., data=carTrain)
car_rpart_testPred = predict(car_rpart, carTest, type = "class")

classfiers = c('C50', 'rpart',  'rForest')
accurracy = c( mean(car_c50_testPred == carTest$category),
               mean(car_rpart_testPred == carTest$category),
               mean(car_Forest2_testPred == carTest$category))

res <- data.frame(classfiers, accurracy)
View(res)


#######################################
#Datasets for the task

#Wines
download.file('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/winequality-red.csv', 'wine_red.csv');
download.file('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/winequality-white.csv', 'wine_white.csv');
wineRed_ds = read.table("wine_red.csv", header = TRUE, sep=";", na.strings=
"*")

#creating a new attribute with 3 values(classses) based on
# the orignal class atribute - quality
wineRed_ds$Q2 = lapply(wineRed_ds[,12], function (x)
{
  if(x >6)  { "A"}
  else if(x >4)  {"B"}
  else { "C"}
})

wineRed_ds$Q2 = unlist(wineRed_ds$Q2);
wineRed_ds$Q2 = as.factor(wineRed_ds$Q2)

#cars
#download.file('http://archive.ics.uci.edu/ml/machine-learning-
databases/car/car.data', 'car.data')
#cars_ds = read.csv("E:\\dydaktyka\\als\\lab\\datasets\\car.data", header =
FALSE,
#  col.names = c('buying', 'maint', 'doors', 'persons', 'lug_boot','safety',
"category") )

#abalone
download.file('http://archive.ics.uci.edu/ml/machine-learning-
databases/abalone/abalone.data', 'abalone.data')
abalone  = read.table("abalone.data", header = FALSE, sep=",", na.strings=
"*")
colnames(abalone) <- c('Sex', 'Length','Diameter','Height','Whole',
'Shucked', 'Viscera','Shell','Rings')
```

```
abalone$Age = lapply(abalone[,'Rings'], function (x)
{
  if(x >10)  { "Old"}
  else if(x >8)  {"Middle"}
  else { "Young"}
})
abalone$Age = unlist(abalone$Age);
abalone$Age = as.factor(abalone$Age)
#abalone$Rings <- NULL
download.file('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/winequality-white.csv', 'wine_white.csv');
wineRed_ds = read.table("wine_red.csv", header = TRUE, sep=";", na.strings=
"*")
```