```
#######################################################
# Part 5: DECISION TREES
# Apply Decision Trees Analysis
#
#
#######################################################

# load package FactoMineR and ggplot2
require(ggplot2)
require(rpart)

# read cleaned data set
dd = read.csv("CleanCreditScoring.csv", header=TRUE, stringsAsFactors=TRUE)

# Let's obtain a decision tree with the function 'rpart'
# using all the variables (both continuous and categorical)
ct = rpart(Status ~ ., data=dd)
# let's see how the output looks like
ct
# how to read the output?
# "node), split, n, loss, yval, (yprob)"
# node): indicates the node number
# split: indicates the split criterion
# n: indicates the number of individuals in the groupe
# loss: indicates the the number of individuals misclassified
# yval: indicates the predicted value
# (yprob): indicates the probability of belonging to each class

# it's much easier to read a tree with a graphic
plot(ct, margin=0.05, compress=TRUE, main="Decision Tree")
text(ct, use.n=TRUE, pretty=1, all=TRUE, cex=0.7)

# one of the goals is to obtain a tree in which
# the nodes are as much homogenous as possible,
# but also a tree with good prediction ability
# In order to improve our decision tree, we need to have
# 1) a train (aka learning) dataset
# 2) a test dataset
# let's keep 2/3 of the data for learning, and 1/3 for testing
n = nrow(dd)
learn = sample(1:n, size=round(0.67 * n))
nlearn = length(learn)
ntest = n - nlearn

# selection of model by crossvalidation
# first we need a maximal tree with low value of cp
# and quiprobability of classes
ct1 = rpart(Status ~ ., data = dd[learn,], method="class",
            parms = list(prior=c(0.50, 0.50), split='gini'),
            control = rpart.control(cp=0.001, xval=10, maxdepth=15))

# check results of the complexity parameter table
ct1$cptable
# we can use the function 'plotcp' to see the results
# the 'best' tree is the one with the lowest xerror
plotcp(ct1, las=2, cex.axis=0.8)
# what is the minimum XERROR?
min(ct1$cptable[,4])
min.xe = which(ct1$cptable[,4] == min(ct1$cptable[,4]))
# the optimal tree corresponds to a cp=0.003
ct1$cptable[min.xe,]

# Now that we know that the 'best' tree has cp=0.03
# we can plugin that information in the parameters
```

```
ct2 = rpart(Status ~ .,
      data = dd[learn,],
      parms = list(prior=c(0.50, 0.50), split='gini'),
      control = rpart.control(cp=0.00285, xval=0, maxdepth=15))
# plot
par(mar = c(1,1,2,0.5))
plot(ct2, margin=0.05, compress=TRUE, main="Decision Tree")
text(ct2, use.n=TRUE, pretty=1, all=TRUE, cex=0.5)
summary(ct2)

# calculate error rate in the learning sample
# (this will give a matrix)
ct2.learn = predict(ct2, data=ddtot[learn,])
# create a vector with predicted status
ct2.learnp = rep("", nlearn)
ct2.learnp[ct2.learn[,1] < 0.5] = "pred_neg"
ct2.learnp[ct2.learn[,1] >= 0.5] = "pred_pos"
# let's make a table
status_learn = table(dd$Status[learn], ct2.learnp)
# classification error
100 * sum(diag(status_learn)) / nlearn


# calculate error rate in the testing sample
# (this will give a matrix)
ct2.test = predict(ct2, newdata=ddtot[-learn,])
# create a vector with predicted status
ct2.testp = rep("", ntest)
ct2.testp[ct2.test[,1] < 0.5] = "pred_neg"
ct2.testp[ct2.test[,1] >= 0.5] = "pred_pos"
# let's make a table
status_test = table(dd$Status[-learn], ct2.testp)
# classification error
100 * sum(diag(status_test)) / ntest


# we'll repeat the same but changing the cp=0.002
ct3 = rpart(Status ~ .,
            data = dd[learn,],
            parms = list(prior=c(0.50, 0.50), split='gini'),
            control = rpart.control(cp=0.002, xval=0, maxdepth=15))

par(mar = c(1,1,2,0.5))
plot(ct3, margin=0.05, compress=TRUE, main="Decision Tree")
text(ct3, use.n=TRUE, all=TRUE, cex=0.5)

# calculate error rate in the learning sample
# (this will give a matrix)
ct3.learn = predict(ct3, data=ddtot[learn,])
# create a vector with predicted status
ct3.learnp = rep("", nlearn)
ct3.learnp[ct3.learn[,1] < 0.5] = "pred_neg"
ct3.learnp[ct3.learn[,1] >= 0.5] = "pred_pos"
# let's make a table
table(dd$Status[learn], ct3.learnp)
# classification error
100 * sum(diag(table(dd$Status[learn], ct3.learnp))) / nlearn

# calculate error rate in the testing sample
# (this will give a matrix)
ct3.test = predict(ct3, newdata=ddtot[-learn,])
# create a vector with predicted status
ct3.testp = rep("", ntest)
ct3.testp[ct3.test[,1] < 0.5] = "pred_neg"
ct3.testp[ct3.test[,1] >= 0.5] = "pred_pos"
```

```
# let's make a table
table(dd$Status[-learn], ct3.testp)
# classification error
100 * sum(diag(table(dd$Status[-learn], ct3.testp))) / ntest


# concentration curve
# the positive predictions on the test sample
pred.test = ct2.test[,1]
# the number of individuals in each value
totn = table(-pred.test) / ntest
ac_totn = 100 * cumsim(as.numeric(totn))
# ranking the predictions
rank_pred.test = rank(pred.test)
# how many positive are in each leave?
Status.test = dd$Status[-learn]
table(Status.test)
npos = table(Status.test)[1]

tapply(Status.test == "good", ran_pred.test, sum)
ac_true.pos = 100 * cumsum(rev(as.numeric(tabla))) / npos


# ROC curve
nneg = ntest - npos
ac_fals.pos = 100 * cumsum(rev())
```