R for Classification Trees et al.
Dolores Romero Morales

The aim of this homework is to revise classification trees using a credit scoring (creditgerman.txt). This file can be found on the course directory. This file has twenty explanatory variables. The last column is the class membership. There are two classes: '1' (Good) and '2' (Bad). If you want to learn more about the dataset, please go to the well-known UCI Machine Learning repository, and find the german credit dataset using the search tool (https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)).

Recall that we will need to install the corresponding R packages.
Note1: To build Classification Trees we will use the R package 'tree'. Follow the two steps below to complete the installation and uploading of the package:

```
install.packages('tree',dependencies=TRUE)
library('tree')
```

Step 1. Read the creditgerman.txt file into a data frame and get the dimension of the dataset.
Answer:

```
mycredit <- read.table(file.choose(),header=T, stringsAsFactors=TRUE)
dim(mycredit)
```

Note: It would be great that you familiarize with the explanatory variables you have. Please note that you have many categorical variables. Please read about those in https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data).

Step 2. For your convenience, convert the target variable into a categorical one. Note that you can do this with the function 'as.factor()'.
Answer:

```
mycredit$classmembership <- as.factor(mycredit$classmembership)
```

Note: If you do not perform this step, the values of the response classmembership, '1' and '2', will be understood as numeric. However, these are simply the names of the two classes '1' (Good) and '2' (Bad).

Step 3. Get a summary report for all the explanatory variables and the class split, i.e., the number of observations in each class. Recall that the class membership is given by the last column.
Answer:

```
dim(mycredit)
classmembershiptable <- table(mycredit$classmembership)
goodinmycredit <- classmembershiptable [1]/nrow(mycredit)
badinmycredit <- classmembershiptable [2]/nrow(mycredit)
```

Note: Note this is an imbalance dataset with a 70%-30% class split.

Step 4. Reshuffle the dataset and take a subsample of 750 observations. Call this the *minicredit*. Call the remaining dataset the *testcredit*.
Answer:

```
set.seed(1)
reshuffle <- mycredit[sample(nrow(mycredit)),]
minicredit <- reshuffle[1:750,]
```

```
testcredit <- reshuffle[751: nrow(reshuffle),]
```

Step 5. Get a summary report for all the explanatory variables and the class split, i.e., the number of observations in each class for both the *minicredit* and *testcredit*.

Answer:

```
dim(minicredit)
classmembershiptablemini <- table(minicredit$classmembership)
goodinminicredit <- classmembershiptablemini [1]/nrow(minicredit)
badinminicredit <- classmembershiptablemini [2]/nrow(minicredit)

dim(testcredit)
classmembershiptabletest <- table(testcredit$classmembership)
goodintestcredit <- classmembershiptabletest [1]/nrow(testcredit)
badintestcredit <- classmembershiptabletest [2]/nrow(testcredit)
```

Note: Note both the minicredit as well as the testcredit have roughly speaking a 70%-30% class split.

Step 6. Using the *minicredit* subsample build a tree using the crossvalidation function.

Answer:

```
set.seed(100)
mytree <- tree(classmembership~.,minicredit)
plot(mytree)
text(mytree)
summary(mytree)
set.seed(1000)
mycrossval <- cv.tree(mytree,FUN=prune.tree)
mybestsize <- mycrossval$size[which(mycrossval$dev==min(mycrossval$dev))]
myprunedtree <- prune.tree(mytree,best =mybestsize)
```

Step 7. Plot the tree and discuss the contribution of the categorical explanatory variables and of the numerical ones.

Answer:

```
plot(myprunedtree)
text(myprunedtree)
```

Note1: You are required here to discuss the contribution of the categorical explanatory variables and of the numerical ones. For this, you can print the rules that define the tree. In these rules, you can see that one categorical variable intervenes as well as one numeric variable.

```
Classification tree:
snip.tree(tree = mytree, nodes = c(2L, 7L, 6L))
Variables actually used in tree construction:
[1] "Statusof"        "Durationinmonth"
Number of terminal nodes:  3
Residual mean deviance:  1.071 = 800.1 / 747
Misclassification error rate: 0.2627 = 197 / 750

node), split, n, deviance, yval, (yprob)
      * denotes terminal node
```

1) root 750 905.9 1 (0.7080 0.2920)
  2) Statusof: A13,A14 355 281.3 1 (0.8648 0.1352) *
  3) Statusof: A11,A12 395 540.5 1 (0.5671 0.4329)
    6) Durationinmonth < 22.5 221 280.4 1 (0.6697 0.3303) *
    7) Durationinmonth > 22.5 174 238.4 2 (0.4368 0.5632) *

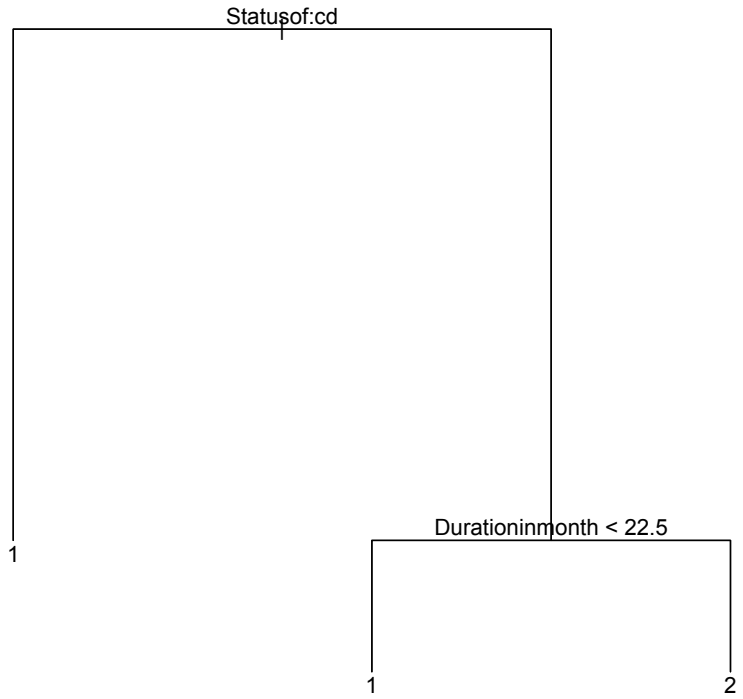

Figure 1 Tree plot obtained with the package tree after pruning

Note2: It is important to note one thing. The tree may suffer from the class imbalance. There are different ways of addressing this (undersampling, oversampling, asymmetric cost of misclassification, etc.). The easiest is to use the rpart package (an alternative package to build trees), which supports asymmetric cost of misclassification. These costs become parameters of the tree building function ('parms = list(loss = myloss)'). The default matrix would be

myloss <- matrix(c(0, 1, 1, 0), nrow = 2, byrow = TRUE)

but you can change it to any other matrix, where you keep the diagonal equal to zero, such as,

myloss <- matrix(c(0, 100, 1, 0), nrow = 2, byrow = TRUE)

The error associated with '100' will be more penalized than the error associated with '1'. This is what you would do if you want the smaller class to be protected (errors in the smaller class count more). By changing the '100' to other values you can generate different trees, with more or less protection of the smaller class.

Step 8. Use the tree to test the model in *testcredit* and report the classification matrix as well as the classification accuracy.

Answer:

```
myprediction <- predict(myprunedtree, testcredit, type='class')
classificationtable <- table(myprediction,testcredit[,21])
acctesttree <- sum(diag(classificationtable))/sum(classificationtable)
```

Step 9. Compare the accuracy results on the *testcredit* with those of Logistic Regression, SVM with different kernels, K-NN and Random Forests.

Answer:

This will become a future homework, but you should be able to start already using past PC Workshops answers.

Note1: Please note that the FNN() package you have seen in class uses as default the Euclidean distance. Therefore, it assumes your data is numeric or integer. This is not the case for the creditgerman dataset. These are the alternatives you have:

OPTION 1: You can use a different K-NN package, that can deal with categorical as well as continuous variables, see, for instance, knncat(). Please note that this package will try to find an alternative encoding to the categorical variables to the one given by the dummy variables (zero-one encoding). It also gives an alternative representation to the quantitative variables.

OPTION 2: You have the option of transforming the categorical variables into dummies. You need one dummy variable for each category categorical variables have. This dataset is available in the internet where this transformation of the categorical variables has been performed. Below I have added some code for you to create the dataset yourself. Please note that, this is a less preferred option because the size of your dataset will increase dramatically when you have many categories.

```
install.packages('dummies',dependencies=TRUE)
library(dummies)
myXcredit <- mycredit[,-21]
classassignment <- mycredit[,21]
myXcreditD <- dummy.data.frame(myXcredit, sep= ".")
mycreditD <-  cbind(myXcreditD, classassignment)
```

You can have a look at the variables of the original set and the variables of the set with dummies to check that indeed you do not have categorical explanatory variables anymore. Of course, the. size of the dataset has increased from 20 explanatory variables to 61.

ORGINAL dataset

```
'data.frame':    1000 obs. of  21 variables:
 $ Statusof              : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4 4 2 4 2 ...
 $ Durationinmonth       : int  6 48 12 42 24 36 24 36 12 30 ...
 $ Credithistory         : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3 3 3 3 5 ...
 $ Purpose               : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1 8 4 2 5 1 ...
 $ Creditamount          : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
 $ Savingsaccountbonds   : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1 5 3 1 4 1 ...
 $ Presentemploym        : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3 3 5 3 4 1 ...
 $ Installmentratein     : int  4 2 2 2 3 2 3 2 2 4 ...
 $ Personalstatusandsex  : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3 3 3 3 1 4 ...
```

$ Otherdebtorsguarantors : Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1 1 1 1 ...
$ Presentres : int 4 2 3 4 4 4 4 2 4 2 ...
$ Property : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2 3 1 3 ...
$ Ageinyears : int 67 22 49 45 53 35 53 35 61 28 ...
$ Otherinstallmentplans : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3 3 3 3 ...
$ Housing : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2 1 2 2 ...
$ Numberofexistingcr : int 2 1 1 1 2 1 1 1 1 2 ...
$ Job : Factor w/ 4 levels "A171","A172",..: 3 3 2 3 3 2 3 4 2 4 ...
$ Numbprovidemaintenancefor: int 1 1 2 2 2 2 1 1 1 1 ...
$ Telephone : Factor w/ 2 levels "A191","A192": 2 1 1 1 1 2 1 2 1 1 ...
$ foreignworker : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1 1 1 ...
$ classmembership : Factor w/ 2 levels "1","2": 1 2 1 1 2 1 1 1 1 2 ...

Dataset with dummies

'data.frame': 1000 obs. of 62 variables:
$ StatusofA11 : int 1 0 0 1 1 0 0 0 0 0 ...
$ StatusofA12 : int 0 1 0 0 0 0 0 1 0 1 ...
$ StatusofA13 : int 0 0 0 0 0 0 0 0 0 0 ...
$ StatusofA14 : int 0 0 1 0 0 1 1 0 1 0 ...
$ Durationinmonth : int 6 48 12 42 24 36 24 36 12 30 ...
$ CredithistoryA30 : int 0 0 0 0 0 0 0 0 0 0 ...
$ CredithistoryA31 : int 0 0 0 0 0 0 0 0 0 0 ...
$ CredithistoryA32 : int 0 1 0 1 0 1 1 1 1 0 ...
$ CredithistoryA33 : int 0 0 0 0 1 0 0 0 0 0 ...
$ CredithistoryA34 : int 1 0 1 0 0 0 0 0 0 1 ...
$ PurposeA40 : int 0 0 0 0 1 0 0 0 0 1 ...
$ PurposeA41 : int 0 0 0 0 0 0 0 1 0 0 ...
$ PurposeA410 : int 0 0 0 0 0 0 0 0 0 0 ...
$ PurposeA42 : int 0 0 0 1 0 0 1 0 0 0 ...
$ PurposeA43 : int 1 1 0 0 0 0 0 0 1 0 ...
$ PurposeA44 : int 0 0 0 0 0 0 0 0 0 0 ...
$ PurposeA45 : int 0 0 0 0 0 0 0 0 0 0 ...
$ PurposeA46 : int 0 0 1 0 0 1 0 0 0 0 ...
$ PurposeA48 : int 0 0 0 0 0 0 0 0 0 0 ...
$ PurposeA49 : int 0 0 0 0 0 0 0 0 0 0 ...
$ Creditamount : int 1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
$ SavingsaccountbondsA61 : int 0 1 1 1 1 0 0 1 0 1 ...
$ SavingsaccountbondsA62 : int 0 0 0 0 0 0 0 0 0 0 ...
$ SavingsaccountbondsA63 : int 0 0 0 0 0 0 1 0 0 0 ...
$ SavingsaccountbondsA64 : int 0 0 0 0 0 0 0 0 1 0 ...
$ SavingsaccountbondsA65 : int 1 0 0 0 0 1 0 0 0 0 ...
$ PresentemploymA71 : int 0 0 0 0 0 0 0 0 0 1 ...
$ PresentemploymA72 : int 0 0 0 0 0 0 0 0 0 0 ...
$ PresentemploymA73 : int 0 1 0 0 1 1 0 1 0 0 ...
$ PresentemploymA74 : int 0 0 1 1 0 0 0 0 1 0 ...
$ PresentemploymA75 : int 1 0 0 0 0 0 1 0 0 0 ...
$ Installmentratein : int 4 2 2 2 3 2 3 2 2 4 ...
$ PersonalstatusandsexA91 : int 0 0 0 0 0 0 0 0 1 0 ...
$ PersonalstatusandsexA92 : int 0 1 0 0 0 0 0 0 0 0 ...
$ PersonalstatusandsexA93 : int 1 0 1 1 1 1 1 1 0 0 ...
$ PersonalstatusandsexA94 : int 0 0 0 0 0 0 0 0 0 1 ...

$ OtherdebtorsguarantorsA101: int 1 1 1 0 1 1 1 1 1 1 ...
$ OtherdebtorsguarantorsA102: int 0 0 0 0 0 0 0 0 0 0 ...
$ OtherdebtorsguarantorsA103: int 0 0 0 1 0 0 0 0 0 0 ...
$ Presentres : int 4 2 3 4 4 4 4 2 4 2 ...
$ PropertyA121 : int 1 1 1 0 0 0 0 0 1 0 ...
$ PropertyA122 : int 0 0 0 1 0 0 1 0 0 0 ...
$ PropertyA123 : int 0 0 0 0 0 0 0 1 0 1 ...
$ PropertyA124 : int 0 0 0 0 1 1 0 0 0 0 ...
$ Ageinyears : int 67 22 49 45 53 35 53 35 61 28 ...
$ OtherinstallmentplansA141 : int 0 0 0 0 0 0 0 0 0 0 ...
$ OtherinstallmentplansA142 : int 0 0 0 0 0 0 0 0 0 0 ...
$ OtherinstallmentplansA143 : int 1 1 1 1 1 1 1 1 1 1 ...
$ HousingA151 : int 0 0 0 0 0 0 0 1 0 0 ...
$ HousingA152 : int 1 1 1 0 0 0 1 0 1 1 ...
$ HousingA153 : int 0 0 0 1 1 1 0 0 0 0 ...
$ Numberofexistingcr : int 2 1 1 1 2 1 1 1 1 2 ...
$ JobA171 : int 0 0 0 0 0 0 0 0 0 0 ...
$ JobA172 : int 0 0 1 0 0 1 0 0 1 0 ...
$ JobA173 : int 1 1 0 1 1 0 1 0 0 0 ...
$ JobA174 : int 0 0 0 0 0 0 0 0 1 0 1 ...
$ Numbprovidemaintenancefor : int 1 1 2 2 2 2 1 1 1 1 ...
$ TelephoneA191 : int 0 1 1 1 1 0 1 0 1 1 ...
$ TelephoneA192 : int 1 0 0 0 0 1 0 1 0 0 ...
$ foreignworkerA201 : int 1 1 1 1 1 1 1 1 1 1 ...
$ foreignworkerA202 : int 0 0 0 0 0 0 0 0 0 0 ...
$ classassignment : Factor w/ 2 levels "1","2": 1 2 1 1 2 1 1 1 1 2 ...