Exercise 5

Business Analytics and Data Science WS18/19

Introduction

In this exercise we will deepen our understanding of classification methods. We will start with looking into a logistic regression (remember the difference from the linear regression) and then explore a very popular machine learning model called "Decision tree", that can be used for classification as well as regression problems. Later in the course, we will see them again as part of the very popular random forest ensemble model.

Logistic regression

Logistic regression is one of the standard methods in predictive analytics, modeling the outcome of a categoric variable by a linear combination of independent variables transformed in a way to predict outcome probabilities between 0 and 1.

1. Load the data set saved at https://stats.idre.ucla.edu/stat/data/binary.csv.
2. Analyze the structure of the data with the appropriate functions. Which variable is a binary one? Are there any missing values? Let's call for a contingency table to check how the binary variable is represented among the ranks.
3. Get a feeling for the data and check for plausability with some easy visualizations. A student's grade point average (GPA) indicates their performance in school, while the Graduate Record Examination (GRE) (definitely not endorsed) is one standardized test. A high score should be indicative of university admission. Plot the distribution of GPA scores for students that have been admitted and those that weren't as a **boxplot()**. This calls for another **formula** argument.
4. We haven't learnt about the preparation of the data yet but we will do just one little trick: convert the **rank** variable from numeric to factor (use *as.factor* function). We want to do the same with the target variable. But watch your numbers! by convention, the target class 1 should become the second factor level (meaning admission).
5. Let's see if GPA scores might be a good predictor of GRE results. Check the correlation of the two. At least we won't have to worry about collinearity.
6. For a little more insight, create a scatterplot of the two variables. Add a red regression line predicting y using x.

```r
# Prepare the data
uni <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv",header=TRUE, sep=",")
str(uni)
```

```
## 'data.frame':    400 obs. of  4 variables:
##  $ admit: int  0 1 1 1 0 1 1 0 1 0 ...
##  $ gre  : int  380 660 800 640 520 760 560 400 540 700 ...
##  $ gpa  : num  3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...
##  $ rank : int  3 3 1 4 4 2 1 2 3 2 ...
```

```r
summary(uni)
```

```
##      admit             gre             gpa             rank      
##  Min.   :0.0000   Min.   :220.0   Min.   :2.260   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:520.0   1st Qu.:3.130   1st Qu.:2.000  
##  Median :0.0000   Median :580.0   Median :3.395   Median :2.000  
##  Mean   :0.3175   Mean   :587.7   Mean   :3.390   Mean   :2.485  
##  3rd Qu.:1.0000   3rd Qu.:660.0   3rd Qu.:3.670   3rd Qu.:3.000  
##  Max.   :1.0000   Max.   :800.0   Max.   :4.000   Max.   :4.000  
```

```r
class(uni$admit)
```

```
## [1] "integer"
```

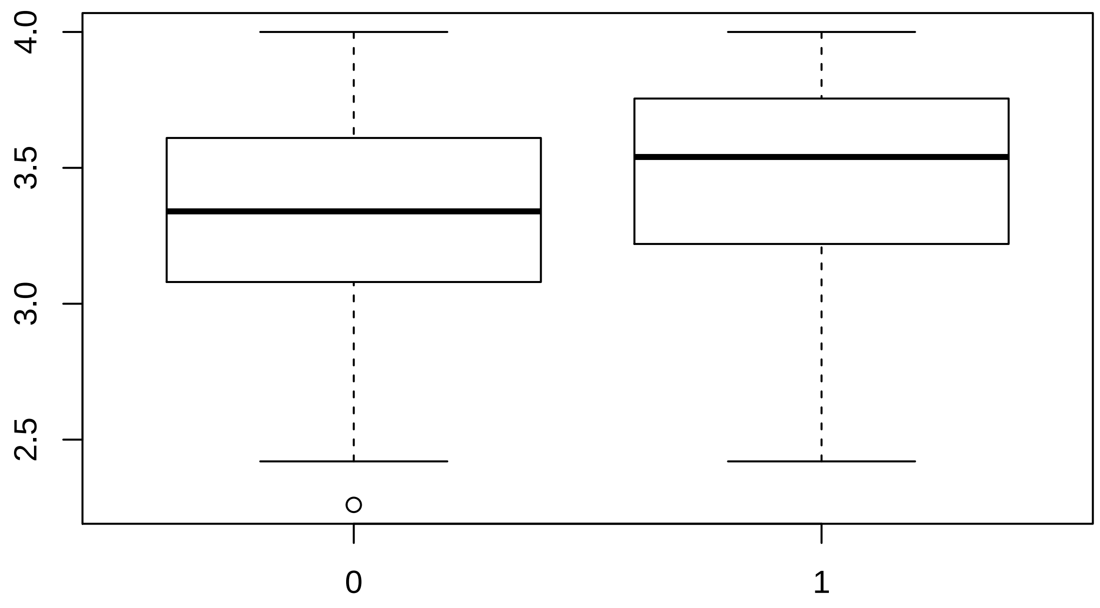
```r
table(uni$admit)
```

```
##
##   0   1
## 273 127
```

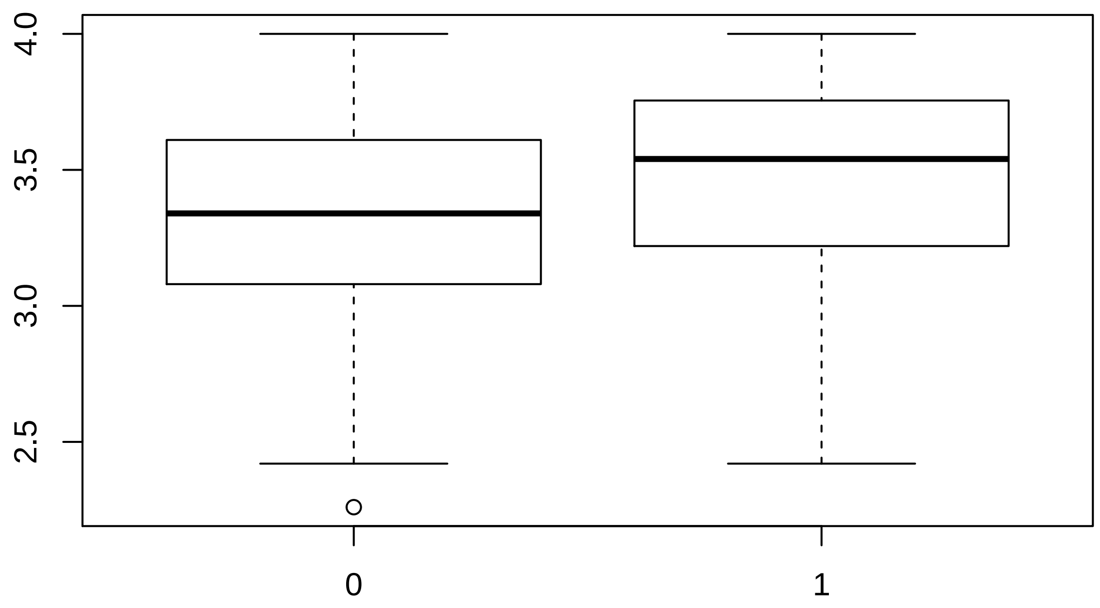
```r
xtabs(~admit + rank, data = uni)
```

```
##      rank
## admit  1  2  3  4
##     0 28 97 93 55
##     1 33 54 28 12
```

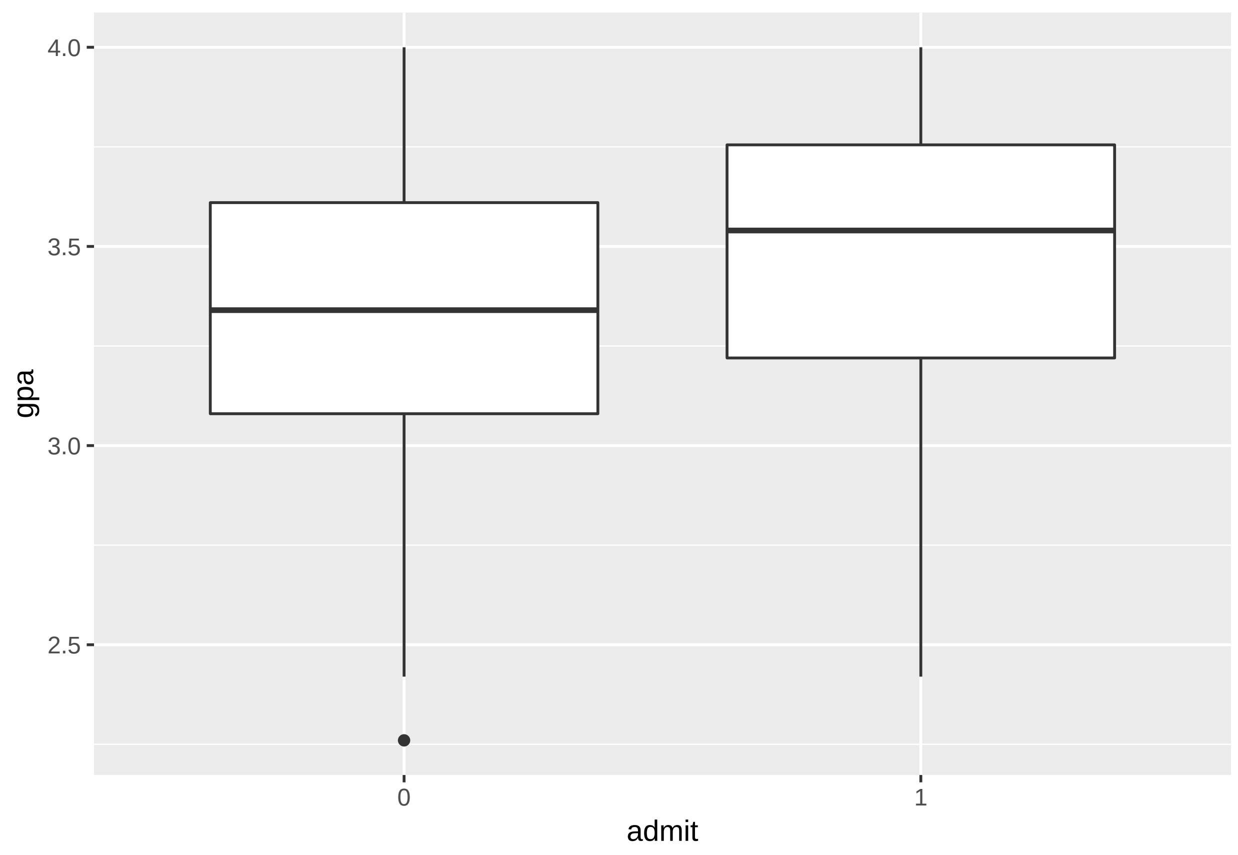
```r
#seems that it is ok, no NA. Rank 1 is clearly the "best" with the highest admission ratio

uni$rank <- factor(uni$rank, levels = c(4,3,2,1))
uni$admit <- factor(uni$admit)

#par(mfrow=c(2,2))
# Boxplot of GPA grouped by variable admit (1=yes/0=no)
boxplot(gpa ~ admit, data = uni)
```



```r
library(ggplot2)
```



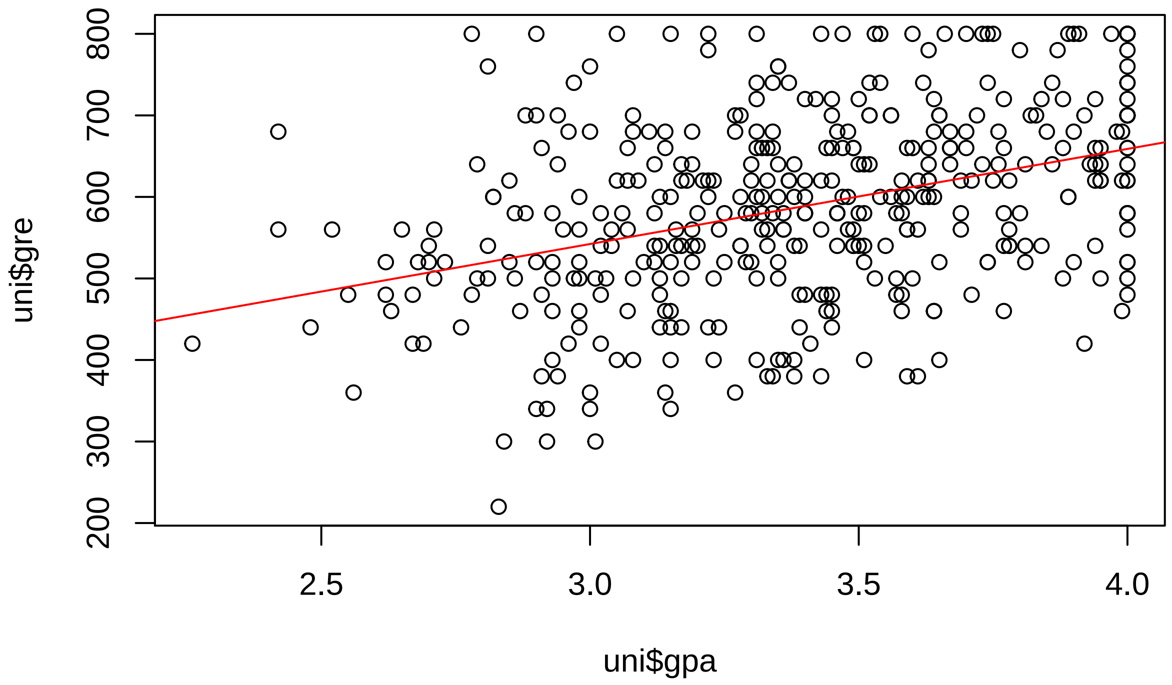
```r
ggplot(uni, aes(y=gpa, x= admit)) + geom_boxplot()
```



```
# Correlation of GPA and GRE scores
cor(uni$gpa, uni$gre)
```

```
## [1] 0.3842659
```

```
# Scatterplot with regression line
# Note: The swirly brackets encase a block of code. This is
#       necessary only for Rmarkdown to make sure that the plot
#       and regression line are drawn 'at the same time'
{
plot(uni$gpa, uni$gre)
regressionLine <- lm(gre ~ gpa, data = uni)
abline(regressionLine, col = 'red')
}
```



7. Run a logistic regression to predict the dependent variable **admit** based on all other variables in the data set. Use the **glm()** generalized model function to create the model and save the trained model as **lr**. Hint 1: We have already discussed how to specify "all other variables" but you can check with the help on **formula**. Hint 2: Specify that you need a logistic model with option **family**. Check the help for **family** to find out which specification to use if you are unsure.
8. Which variables have a significant impact on the value of **admit**? Make sure you know how to interpret your coefficients. The logistic regression coefficients give the change in the log odds of the outcome for a one unit increase in the predictor variable.
9. Use the trained model **lr** to predict for each observation in the **uni** data frame the probability that the applicant will be addmitted. Use function **predict()** and store the predictions as anew column of your dataset called **prediction**.
10. Evaluate the model by computing the prediction accuracy of the model. This is defined as the ratio of correct predictions (predicted *yes* if observed *yes*, predicted *no* if observed *no*) to the overall number of predictions.
11. Last, compute the Brier score **brier.lr** of your model to evaluate not only accuracy but also the *calibration* of your model. The Brier score is the mean square error (MSE) between the actual values of **admit** (1/0) and the predicted class probabilities **pred.lr**. How well does your model perform?

```
# The glm function with a logit link offers the necessary functionality.
lr <- glm(admit ~rank, data = uni, family = binomial(link="logit"))

# Given that glm() supports many types of models including logistic
# regression, you need to specify which model you need.
# The third parameter in the above function call does just that.

# Print a summary of the model
summary(lr)
```

```
##
## Call:
## glm(formula = admit ~ rank, family = binomial(link = "logit"),
##     data = uni)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2479  -0.9408  -0.7255   1.1085   1.8546
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.5224     0.3186  -4.778 1.77e-06 ***
## rank3         0.3220     0.3847   0.837  0.40252
## rank2         0.9367     0.3610   2.595  0.00947 **
## rank1         1.6867     0.4093   4.121 3.77e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 474.97  on 396  degrees of freedom
## AIC: 482.97
##
## Number of Fisher Scoring iterations: 4
```

```r
# Note how rank4 is left out and becomes the base level (i.e. all others 0)

# As for interpretation
betas <- lr$coefficients
betas
```

```
## (Intercept)       rank3       rank2       rank1
##  -1.5224265   0.3220316   0.9366996   1.6867296
```

```r
# The regression part gives a prediction on the odds of our target class 'admission'
# In other words, prob(admission)/prob(rejection)
# If the odds are >1, then the chance to be sucessful is higher than that of rejection
xtabs(~admit + rank, data = uni)
```

```
##      rank
## admit  4  3  2  1
##     0 55 93 97 28
##     1 12 28 54 33
```

```r
xtabs(~admit + rank, data = uni)[2,] / xtabs(~admit + rank, data = uni)[1,]
```

```
##         4         3         2         1
## 0.2181818 0.3010753 0.5567010 1.1785714
```

```r
# And for comparison as calculated by our logistic regression
betas
```

```
## (Intercept)       rank3       rank2       rank1
##  -1.5224265   0.3220316   0.9366996   1.6867296
```

```
#Let's try to interpret it, remember logit tries to create linear relationships
# Baseline (rank 4): Intercept + 0 + 0 + 0
exp(betas[1])
```

```
## (Intercept)
##   0.2181818
```

```
# Rank 1: exp(Intercept + 0 + 0 + 1.68)
exp(betas[[1]]+betas[[4]])
```

```
## [1] 1.178571
```

```
# Which are the odds based on our tabulation

# We can now calculate odds ratio: having attended a rank 1 instead of a rank 4 school decreases admiss
1.178571/0.2181818
```

```
## [1] 5.401784
```

```
# which turns out is equivalent to our effect for rank 1
exp(betas[4])
```

```
##    rank1
## 5.401786
```

```
# What does that mean in terms of probability?
prob_rank4 = 1/(1+exp(- betas[[1]]))
prob_rank1 = 1/(1+exp(- (betas[[1]]+betas[[4]])))
prob_rank1/prob_rank4
```

```
## [1] 3.020492
```

```
# Not 5 times the probability!


#####

# Now let's include all variables
lr <- glm(admit ~., data = uni, family = binomial(link="logit"))

summary(lr)
```

```
##
## Call:
## glm(formula = admit ~ ., family = binomial(link = "logit"), data = uni)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6268  -0.8662  -0.6388   1.1490   2.0790
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.541443   1.138072  -4.869 1.12e-06 ***
## gre          0.002264   0.001094   2.070 0.038465 *
## gpa          0.804038   0.331819   2.423 0.015388 *
## rank3        0.211260   0.392857   0.538 0.590748
## rank2        0.876021   0.366735   2.389 0.016908 *
## rank1        1.551464   0.417832   3.713 0.000205 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 458.52  on 394  degrees of freedom
## AIC: 470.52
##
## Number of Fisher Scoring iterations: 4
```

```
#For every one unit change in gre, the log odds of admission (versus non-admission) increases by 0.002.
#For a one unit increase in gpa, the log odds of being admitted to graduate school increases by 0.804.
#The indicator variables for rank have a slightly different interpretation. For example, having attende

# Compute model predictions. For simplicity, we use the same data
# that we used to build the model (i.e., resubstitution estimate)
uni$prediction <- predict(lr, newdata = uni, type="response" )
# The last parameter says that the predicitons should be on the same
# scale as the response (i.e., dependent variable). Try calling the
# predict function without this parameter and see what happends.
head(predict(lr, newdata = uni))
```

```
##          1          2          3          4          5          6
## -1.5671256 -0.8848442  1.0377118 -1.5273305 -2.0081113 -0.5323458
```

```
# Start with an intuitive measure of model performance. For binary outcomes,
# the accuracy of a model describes how often the predicted class matches
# the observed outcome.
# Infer the predicted class from the predicted class probabilities
# We chose the default threshold of 0.5. Is that a good idea? Probably not.

uni$prediction_class <- ifelse(uni$prediction > 0.5, "1", "0")

#Set the vector that will contain your performance values
accuracy <- vector()
accuracy["Model"]<- sum(uni$prediction_class== uni$admit) / length(uni$prediction_class)


# Do you think your accuracy is good?
# Difficult to tell without a benchmark.
# There are two common naive benchmarks depending on if we test on the predicted classes or predicted p
# First, a simple benchmark for discrete  class prediction is to 'predict' the most frequent  outcome f
# Function rep repeats a value for a number of times, here the number of observations
#Always remember what you are predicting (probabilities)

baseline_probability <- sum(uni$admit == "1")/nrow(uni)

class.benchmark <- rep(baseline_probability, nrow(uni))

#Now we make a little custom function. where we can feed in predicted probabilities
Accuracy <- function(prediction, class, threshold =0.5){
  predClass <-  ifelse(prediction > threshold, 1, 0)
  acc <- sum(predClass == class) / length(class)
  return(acc)
```

```
}

accuracy["Benchmark"] <- Accuracy(prediction = class.benchmark, uni$admit, threshold = 0.5)

# Second, when prediting probabilities, we can make things more realistic by predicting the event with
class.random <- sample(c(0,1), size = nrow(uni), replace = TRUE, prob = c(1-baseline_probability, baseli
accuracy["Random"] <- Accuracy(prediction = class.random, class = uni$admit,threshold = 0.5)
accuracy
```

```
##     Model Benchmark    Random
##    0.7100    0.6825    0.5700
```

```
#What if we change the threshold??
```

Decision Trees

Decisions trees are among the most basic machine learning algorithms used for classification and regression. Trees are very interpretable and can, at least in principle, accomodate missing values in new observations to be predicted.

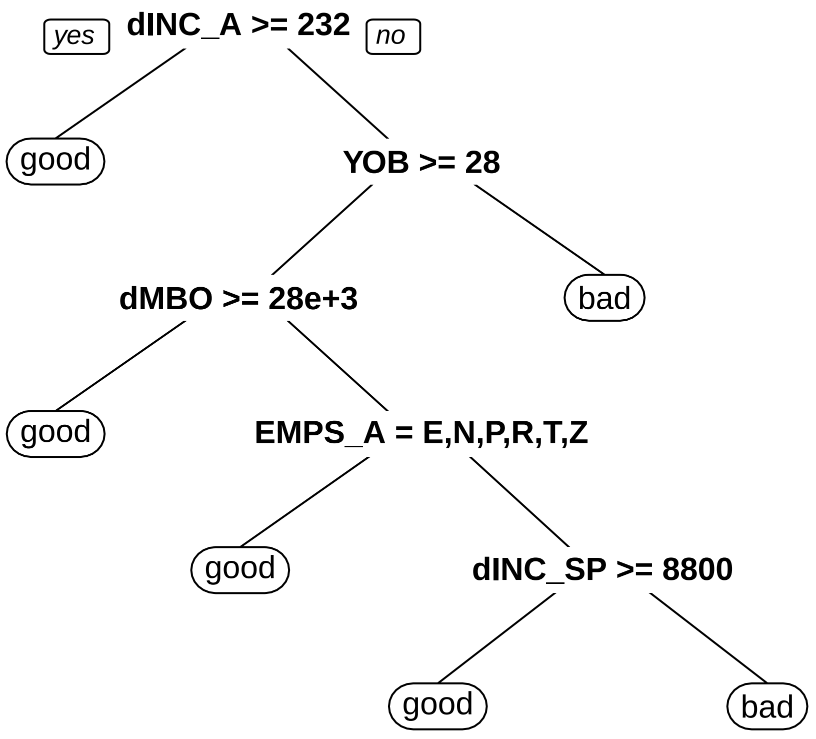
Classification and regression trees work by partitioning the data into smaller, more homogeneous groups. Based on some measure of homogeneity, e.g. the Gini index/impurity (in the two class case $p_1(1-p_1)+p_2(1-p_2)$) or overall sums of squared errors (SSE), the algorithm looks for the variable and split for this variable that most increases homogeneity in the resulting partitions. This splitting process continues within the newly created partions until a pre-defined maximum depth or minimum number of observations in each node is reached. Predictions can then be calculated based on the category probabilities in the terminal nodes (for classification) or a model trained on each subgroup (for regression).

1. Load the loan data set using your custom cleaning function.
2. Use function **rpart()** in the package with the same name to build a decision tree on the data with the default options.
3. As before, predict the the default probability (i.e. BAD == 1) and compare the performance using the brier score.
4. Use package **rpart.plot** to visualize your decision tree. Have a look at the options to optimize it, e.g. to look at how many observations fall into each node.
5. What credit risk does a 55-year old man without income (no response on employment) with three childen and no outstanding mortgage whose wife earns $60.000 a year pose according to the model tree?

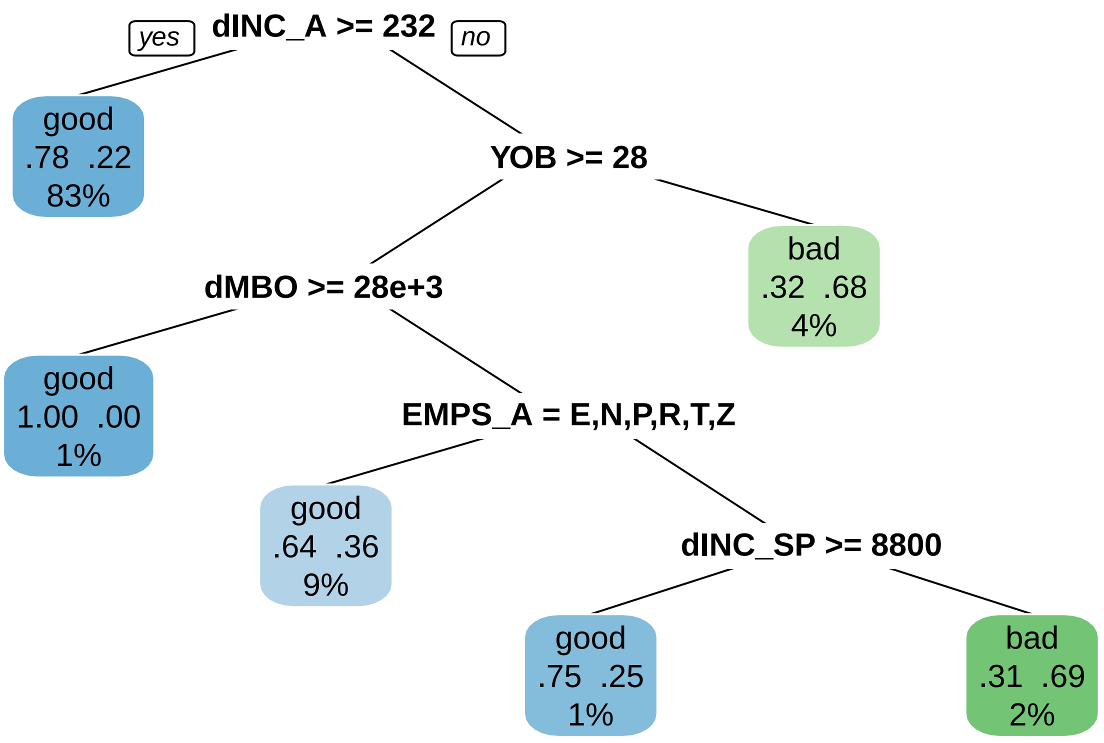
```
# Link to the R script storing your function(s)
source("BADS-HelperFunctions.R")
loans <- get.loan.dataset()
# If the previous line did not work on your machine, the most likely
# cause is that something with the file directories went wrong.
# Check whether the data is available in your working directory.

if(!require("rpart")) install.packages("rpart"); library("rpart") # Try to load rpart, if it doesn't ex

### Training and prediction
dt <- rpart(BAD ~ ., data = loans, method = "class") # create decision tree classifier
pred.dt <-predict(dt, newdata = loans, type = "prob")[, 2] # calculate predictions (in-sample)

# Calculate accuracy of the model
Accuracy <- function(prediction, class, threshold = 0.5){
  # Predict the second factor level if the predicted prob. is higher than
  # a threshold
  predClass <-  ifelse(prediction > threshold, levels(class)[2], levels(class)[1])
  # The accuracy is the ratio of predictions that are equal to
  # the actual observations
  acc <- sum(predClass == class) / length(class)
  return(acc)
}
levels(loans$BAD)
```

```
## [1] "good" "bad"
```

```
acc.dt <- Accuracy(pred.dt, loans$BAD, threshold = 0.25)

### Producing a nice chart of the tree
# (for further improvements see http://blog.revolutionanalytics.com/2013/06/plotting-classification-and
# In order to visualize your decision trees, "rpart.plot" is a handy package. Initially, install the pa
if(!require("rpart.plot")) install.packages("rpart.plot"); library("rpart.plot")

# Visualize the results from "dtree" using the prp() fucntion.
prp(dt)
```



```
prp(dt, extra = 104, border.col = 0, box.palette="auto") # Print the percentage of observations and cla
```



```
#-----------------------------------------------------------------------------------------------------

# Answer question 5: no, yes (YOB == 61), no, yes -> Good risk (36% default risk)

#### Prediction for missing values with surrogate splits ####
# For every split in the tree, a surrogate or replacement split is saved, which gives approximately the
# the surrogate split is used in its place.
summary(dt)
```

```
## Call:
## rpart(formula = BAD ~ ., data = loans, method = "class")
##   n= 1225
##
##           CP nsplit rel error    xerror       xstd
## 1 0.02786378      0 1.0000000 1.0000000 0.04774567
## 2 0.01083591      2 0.9442724 0.9721362 0.04731005
## 3 0.01000000      5 0.9102167 1.0247678 0.04811851
##
## Variable importance
##      dINC_A      EMPS_A         YOB     dINC_SP        dMBO       dHVAL
##          42          26          14           7           7           1
## YOB_missing      dOUTCC        nDEP
##           1           1           1
##
## Node number 1: 1225 observations,    complexity param=0.02786378
##   predicted class=good  expected loss=0.2636735  P(node) =1
##     class counts:   902   323
##    probabilities: 0.736 0.264
##   left son=2 (1019 obs) right son=3 (206 obs)
##   Primary splits:
##       dINC_A < 232   to the right,   improve=20.279080, (0 missing)
##       EMPS_A splits as  LLLLLRLRLRR, improve=14.593490, (0 missing)
##       YOB    < 27.5  to the right,   improve=13.659570, (0 missing)
##       dOUTM  < 22    to the right,   improve= 6.955314, (0 missing)
##       RES    splits as  LRLLL,       improve= 5.921887, (0 missing)
##   Surrogate splits:
##       EMPS_A      splits as  LLLLLLRRLRR, agree=0.874, adj=0.252, (0 split)
##       YOB         < 12.5  to the right,   agree=0.839, adj=0.044, (0 split)
##       YOB_missing < 0.5   to the left,    agree=0.834, adj=0.015, (0 split)
##       dINC_SP     < 35000 to the left,    agree=0.833, adj=0.005, (0 split)
##
## Node number 2: 1019 observations
##   predicted class=good  expected loss=0.2227674  P(node) =0.8318367
##     class counts:   792   227
##    probabilities: 0.777 0.223
##
## Node number 3: 206 observations,    complexity param=0.02786378
##   predicted class=good  expected loss=0.4660194  P(node) =0.1681633
##     class counts:   110    96
##    probabilities: 0.534 0.466
##   left son=6 (156 obs) right son=7 (50 obs)
##   Primary splits:
##       YOB     < 27.5  to the right,   improve=6.046323, (0 missing)
##       EMPS_A  splits as  -LLLLRLRRLL, improve=4.693286, (0 missing)
##       dMBO    < 28464 to the right,   improve=3.615181, (0 missing)
##       dINC_SP < 16000 to the right,   improve=3.270193, (0 missing)
##       dHVAL   < 47552 to the right,   improve=1.487537, (0 missing)
##   Surrogate splits:
##       EMPS_A splits as  -LLLLRLLLLL, agree=0.932, adj=0.72, (0 split)
##       dHVAL  < 51696 to the left,    agree=0.767, adj=0.04, (0 split)
##
## Node number 6: 156 observations,    complexity param=0.01083591
##   predicted class=good  expected loss=0.3974359  P(node) =0.1273469
##     class counts:    94    62
##    probabilities: 0.603 0.397
##   left son=12 (8 obs) right son=13 (148 obs)
##   Primary splits:
##       dMBO    < 28464 to the right,   improve=2.663895, (0 missing)
##       YOB     < 68.5  to the left,    improve=1.960373, (0 missing)
##       dINC_SP < 16000 to the right,   improve=1.890551, (0 missing)
##       EMPS_A  splits as  -LLLLLLRRLL, improve=1.884615, (0 missing)
##       dOUTM   < 172   to the left,    improve=1.464926, (0 missing)
##   Surrogate splits:
##       dINC_SP < 21500 to the right, agree=0.955, adj=0.125, (0 split)
##
## Node number 7: 50 observations
##   predicted class=bad   expected loss=0.32  P(node) =0.04081633
##     class counts:    16    34
##    probabilities: 0.320 0.680
##
```

```
## Node number 12: 8 observations
##   predicted class=good  expected loss=0  P(node) =0.006530612
##     class counts:     8     0
##    probabilities: 1.000 0.000
##
## Node number 13: 148 observations,    complexity param=0.01083591
##   predicted class=good  expected loss=0.4189189  P(node) =0.1208163
##     class counts:    86    62
##    probabilities: 0.581 0.419
##   left son=26 (111 obs) right son=27 (37 obs)
##   Primary splits:
##       EMPS_A splits as  -LRLLLLRRRL, improve=3.0450450, (0 missing)
##       dOUTM  < 172   to the left,    improve=2.5719110, (0 missing)
##       nKIDS  < 1.5   to the left,    improve=2.5155930, (0 missing)
##       YOB    < 63.5  to the right,   improve=1.7463620, (0 missing)
##       dOUTL  < 82    to the right,   improve=0.7415041, (0 missing)
##   Surrogate splits:
##       dINC_SP < 6029  to the left,  agree=0.804, adj=0.216, (0 split)
##       dMBO    < 2466  to the left,  agree=0.791, adj=0.162, (0 split)
##       dHVAL   < 1696  to the left,  agree=0.784, adj=0.135, (0 split)
##       nDEP    < 0.5   to the left,  agree=0.770, adj=0.081, (0 split)
##       dOUTCC  < 70    to the left,  agree=0.770, adj=0.081, (0 split)
##
## Node number 26: 111 observations
##   predicted class=good  expected loss=0.3603604  P(node) =0.09061224
##     class counts:    71    40
##    probabilities: 0.640 0.360
##
## Node number 27: 37 observations,    complexity param=0.01083591
##   predicted class=bad   expected loss=0.4054054  P(node) =0.03020408
##     class counts:    15    22
##    probabilities: 0.405 0.595
##   left son=54 (8 obs) right son=55 (29 obs)
##   Primary splits:
##       dINC_SP < 8800  to the right, improve=2.4240450, (0 missing)
##       dHVAL   < 29696 to the left,  improve=1.6050790, (0 missing)
##       nKIDS   < 1.5   to the left,  improve=1.1563560, (0 missing)
##       YOB     < 58.5  to the left,  improve=0.8578378, (0 missing)
##       dOUTM   < 100   to the left,  improve=0.4930103, (0 missing)
##
## Node number 54: 8 observations
##   predicted class=good  expected loss=0.25  P(node) =0.006530612
##     class counts:     6     2
##    probabilities: 0.750 0.250
##
## Node number 55: 29 observations
##   predicted class=bad   expected loss=0.3103448  P(node) =0.02367347
##     class counts:     9    20
##    probabilities: 0.310 0.690
```

```r
# Create some example data, consisting of the last two lines x2
new.obs <- loans[rep(40:50, each = 2), ]
# Add some mising values to the example data
new.obs[seq(2,nrow(new.obs),2), "dINC_A"] <- NA
```

```
print(new.obs)
```

```
##      YOB nKIDS nDEP PHON dINC_SP EMPS_A dINC_A RES dHVAL  dMBO dOUTM dOUTL
## 40    45     0    0    1   30000      W      0   O 21248 14464     0     0
## 40.1  45     0    0    1   30000      W     NA   O 21248 14464     0     0
## 41    68     0    0    1       0      T   6189   P     0     0     0     0
## 41.1  68     0    0    1       0      T     NA   P     0     0     0     0
## 42    54     2    0    1       0      P  55068   O 43392     4     0     0
## 42.1  54     2    0    1       0      P     NA   O 43392     4     0     0
## 43    45     4    0    1       0      E  42000   O 28928 34464   988     0
## 43.1  45     4    0    1       0      E     NA   O 28928 34464   988     0
## 44    13     0    0    0     850      R  37500   U     0     0   520     0
## 44.1  13     0    0    0     850      R     NA   U     0     0   520     0
## 45    58     0    0    1       0      P  40500   O 26464 64000   520     0
## 45.1  58     0    0    1       0      P     NA   O 26464 64000   520     0
## 46    66     0    0    1       0      P  21750   P     0     0     0     0
## 46.1  66     0    0    1       0      P     NA   P     0     0     0     0
## 47    59     1    0    1   16000      V  15000   O 24928 54464   960   180
## 47.1  59     1    0    1   16000      V     NA   O 24928 54464   960   180
## 48    68     0    0    1       0      T      0   P 48928  6464     0     0
## 48.1  68     0    0    1       0      T     NA   P 48928  6464     0     0
## 49    64     0    0    1       0      R      0   O 34464     4     0     0
## 49.1  64     0    0    1       0      R     NA   O 34464     4     0     0
## 50    45     4    0    1   20000      W      0   F     0     0     0     0
## 50.1  45     4    0    1   20000      W     NA   F     0     0     0     0
##      dOUTHP dOUTCC  BAD YOB_missing
## 40        0      0 good           0
## 40.1      0      0 good           0
## 41        0      0 good           0
## 41.1      0      0 good           0
## 42        0      0 good           0
## 42.1      0      0 good           0
## 43        0      0 good           0
## 43.1      0      0 good           0
## 44        0      0 good           0
## 44.1      0      0 good           0
## 45        0      0 good           0
## 45.1      0      0 good           0
## 46        0     60  bad           0
## 46.1      0     60  bad           0
## 47        0      0 good           0
## 47.1      0      0 good           0
## 48        0      0 good           0
## 48.1      0      0 good           0
## 49        0      0 good           1
## 49.1      0      0 good           1
## 50        0      0 good           0
## 50.1      0      0 good           0
```

```
# Predict for the observations with and without missing values
new.obs[["prediction"]] <- predict(dt, newdata = new.obs, type = "prob")[, 2]
print(new.obs)
```

```
##      YOB nKIDS nDEP PHON dINC_SP EMPS_A dINC_A RES dHVAL  dMBO dOUTM dOUTL
## 40    45     0    0    1   30000       W       0    O 21248 14464      0      0
## 40.1  45     0    0    1   30000       W      NA    O 21248 14464      0      0
## 41    68     0    0    1       0       T    6189    P     0     0      0      0
## 41.1  68     0    0    1       0       T      NA    P     0     0      0      0
## 42    54     2    0    1       0       P   55068    O 43392     4      0      0
## 42.1  54     2    0    1       0       P      NA    O 43392     4      0      0
## 43    45     4    0    1       0       E   42000    O 28928 34464    988      0
## 43.1  45     4    0    1       0       E      NA    O 28928 34464    988      0
## 44    13     0    0    0     850       R   37500    U     0     0    520      0
## 44.1  13     0    0    0     850       R      NA    U     0     0    520      0
## 45    58     0    0    1       0       P   40500    O 26464 64000    520      0
## 45.1  58     0    0    1       0       P      NA    O 26464 64000    520      0
## 46    66     0    0    1       0       P   21750    P     0     0      0      0
## 46.1  66     0    0    1       0       P      NA    P     0     0      0      0
## 47    59     1    0    1   16000       V   15000    O 24928 54464    960    180
## 47.1  59     1    0    1   16000       V      NA    O 24928 54464    960    180
## 48    68     0    0    1       0       T       0    P 48928  6464      0      0
## 48.1  68     0    0    1       0       T      NA    P 48928  6464      0      0
## 49    64     0    0    1       0       R       0    O 34464     4      0      0
## 49.1  64     0    0    1       0       R      NA    O 34464     4      0      0
## 50    45     4    0    1   20000       W       0    F     0     0      0      0
## 50.1  45     4    0    1   20000       W      NA    F     0     0      0      0
##      dOUTHP dOUTCC  BAD YOB_missing prediction
## 40        0      0 good           0  0.2500000
## 40.1      0      0 good           0  0.2500000
## 41        0      0 good           0  0.2227674
## 41.1      0      0 good           0  0.3603604
## 42        0      0 good           0  0.2227674
## 42.1      0      0 good           0  0.2227674
## 43        0      0 good           0  0.2227674
## 43.1      0      0 good           0  0.2227674
## 44        0      0 good           0  0.2227674
## 44.1      0      0 good           0  0.2227674
## 45        0      0 good           0  0.2227674
## 45.1      0      0 good           0  0.2227674
## 46        0     60  bad           0  0.2227674
## 46.1      0     60  bad           0  0.2227674
## 47        0      0 good           0  0.2227674
## 47.1      0      0 good           0  0.2227674
## 48        0      0 good           0  0.3603604
## 48.1      0      0 good           0  0.3603604
## 49        0      0 good           1  0.3603604
## 49.1      0      0 good           1  0.2227674
## 50        0      0 good           0  0.2500000
## 50.1      0      0 good           0  0.2500000
# Even with the variable for the first and most important split missing, the model gives predictions an
```