Statistical Learning Assignment 3 Group 3

```
library(MASS)
library(rpart)
set.seed(1234)
options(scipen = 999)
```

Exercise 1

We fit a classification tree to the `ica` dataset. We choose a loose stopping criterion, e.g. we set the complexity parameter (cp), which describes the minimum improvement in the model needed at each node, very low.

```
data("icu", package = "aplore3")

library(rpart)

head(icu, n=3)
```

```
##   id   sta age gender  race      ser can crn inf cpr sys hra pre      type fra
## 1  4  Died  87 Female White Surgical  No  No Yes  No  80  96  No Emergency Yes
## 2  8 Lived  27 Female White  Medical  No  No Yes  No 142  88  No Emergency  No
## 3 12 Lived  59   Male White  Medical  No  No  No  No 112  80 Yes Emergency  No
##     po2      ph   pco   bic    cre     loc
## 1 <= 60  < 7.25  > 45 >= 18 <= 2.0 Nothing
## 2  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
## 3  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
```

```
tree <- rpart(sta ~ ., data = icu, method = "class",
              parms = list(split = "gini"),
              control = list(cp = 1e-10))
options(digits = 4, width = 60)
printcp(tree)
```

```
##
## Classification tree:
## rpart(formula = sta ~ ., data = icu, method = "class", parms = list(split = "gini"),
##     control = list(cp = 0.0000000001))
##
## Variables actually used in tree construction:
## [1] age id  loc sys
##
## Root node error: 40/200 = 0.2
##
## n= 200
##
##             CP nsplit rel error xerror xstd
## 1 0.2750000000      0      1.00   1.00 0.14
## 2 0.0750000000      1      0.73   0.73 0.12
## 3 0.0250000000      2      0.65   0.70 0.12
```

```
## 4 0.0000000001      4      0.60   0.85 0.13
```

```
plotcp(tree)
library("partykit")
```

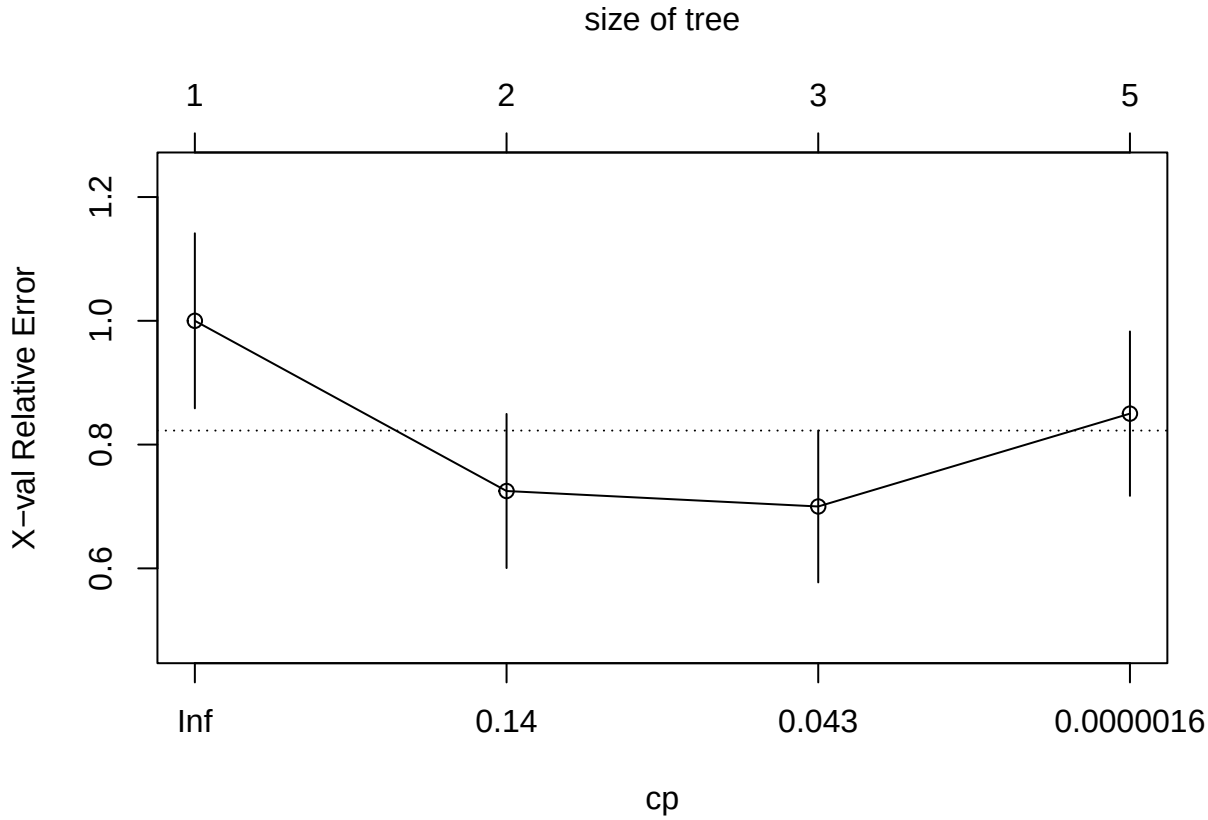
```
## Warning: Paket 'partykit' wurde unter R Version 4.1.3
## erstellt
```

```
## Lade nötiges Paket: grid
```

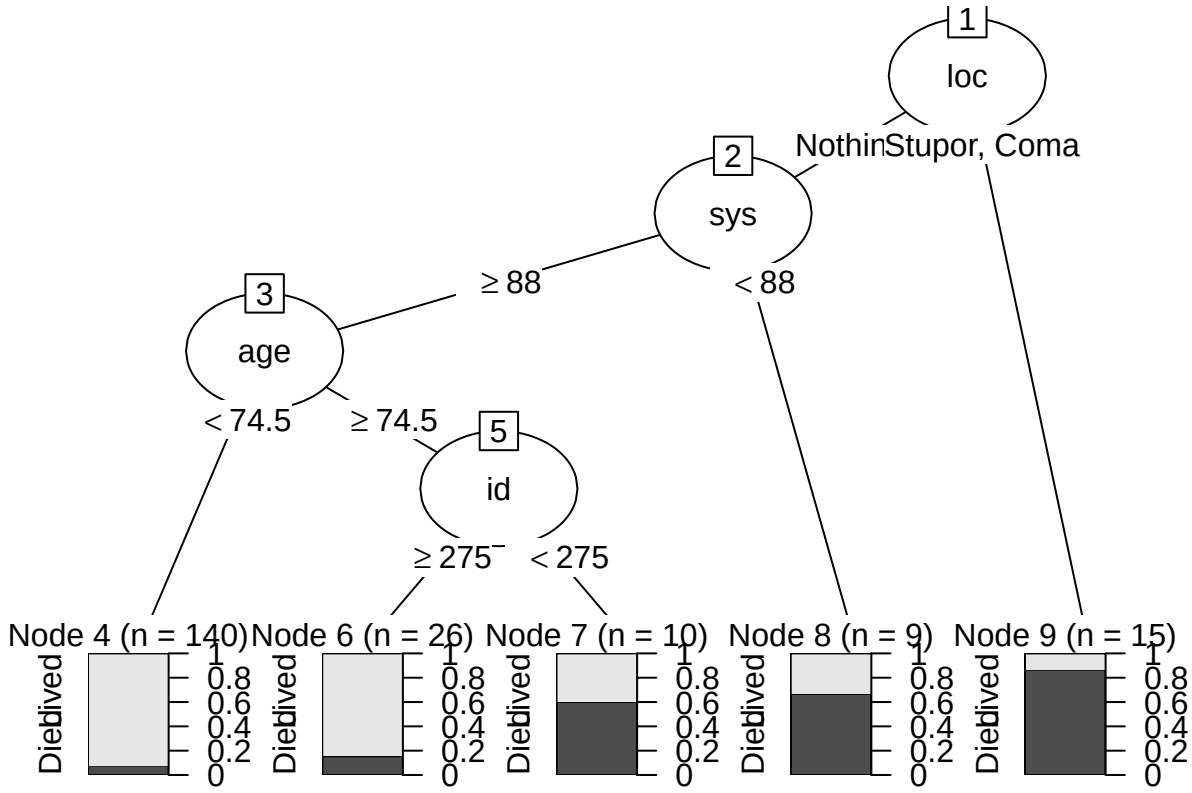
```
## Lade nötiges Paket: libcoin
```

```
## Warning: Paket 'libcoin' wurde unter R Version 4.1.3
## erstellt
```

```
## Lade nötiges Paket: mvtnorm
```




```
tree_ <- partykit::as.party(tree)
plot(tree_)
```

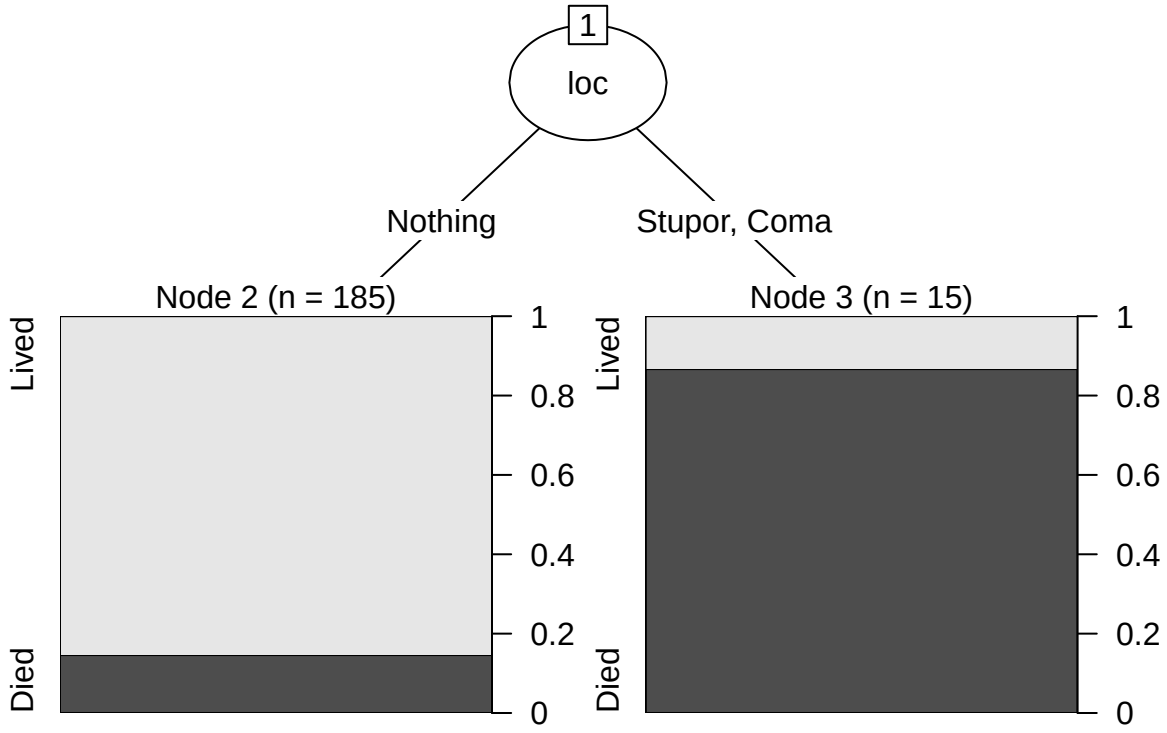


We can see that the splitting continues until a tree size of five nodes is reached (four splits). With such a loose stopping criterion, there is generally a risk of overfitting, however in this example the tree does not get extremely complex. There is basically no improvement after the fifth node. Still, to avoid any overfitting, we prune the tree. As a new complexity parameter we use the first entry where the `xerror` is smaller than the minimum error plus one standard deviation. This criterion is already met after one split.

```
imin <- which.min(tree$cptable[, "xerror"])
select <- which(
  tree$cptable[, "xerror"] <
    sum(tree$cptable[imin, c("xerror", "xstd")]))[1]
ptree <- prune(tree, cp = tree$cptable[select, "CP"])
ptree
```

```
## n= 200
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 200 40 Lived (0.8000 0.2000)
##   2) loc=Nothing 185 27 Lived (0.8541 0.1459) *
##   3) loc=Stupor,Coma 15  2 Died (0.1333 0.8667) *
```

```
ptree <- partykit::as.party(ptree)
plot(ptree)
```



We can see that the data is split once w.r.t. the level of consciousness at ICU admission (`loc`) into "Nothing" and "Stupor, Coma". The survival probability for the first case lies above 80%, while for the second case it is below 20%.

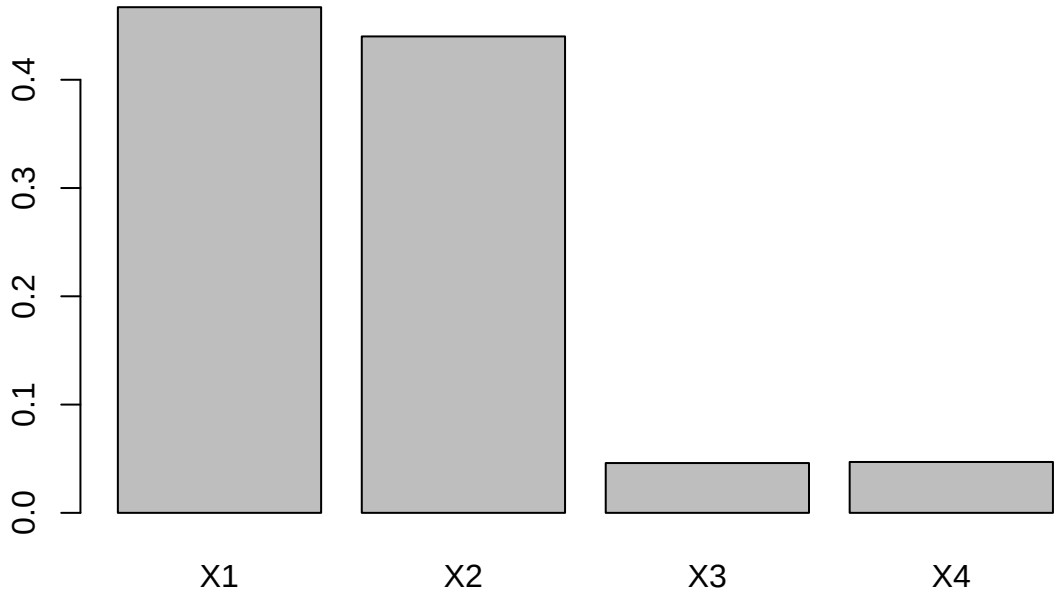
Exercise 2

```
X1 <- runif(100)
X2 <- rnorm(100)
X3 <- rbinom(100,1,0.5)
X4 <- rbinom(100,1,0.1)
X <- cbind(X1,X2,X3,X4)

stump <- function(x=X){
  y <- rnorm(100)
  data <- data.frame(cbind(y,x))
  tree <- rpart(y ~ ., data = data, method = "anova",
                control = list(maxdepth = 1, cp =-1))
  i <- 1 * grepl("X1",labels(tree)[2]) +
    2 * grepl("X2",labels(tree)[2]) +
    3 * grepl("X3",labels(tree)[2]) +
    4 * grepl("X4",labels(tree)[2])
  variable <- paste0("X",i)
  variable
}
reps <- replicate(1000, stump())
```

```
(rel.freq <- table(factor(reps,levels=c("X1","X2","X3","X4")))/length(reps))
```

```
##
##    X1    X2    X3    X4
## 0.467 0.440 0.046 0.047
```

```
barplot(rel.freq)
```



The probability of including the variables as a split is different for each of them. It is highest for X_1. The difference in probabilities results from the difference in distributions. The sample of length 100 from the standard normal distribution is best captured by splitting a uniformly distributed sample and computing the respective means, followed by the standard normal distribution. This is because these two distributions are more flexible when it comes to splitting.

Exercise 3

Y We assume the following data generating process

$$y = x + \epsilon,$$

where $x \sim N(0, 1)$ and $\epsilon \sim N(0, 0.1)$.

Next, we set up a function to

- draw a sample from this data generation process
- fit a linear regression and determine the prediction error
- fit a regression tree using cost-complexity pruning to select a suitable tree

- determine tree size and prediction error

```
generate_y <- function(rep){
    x <- rnorm(rep, 0, 1)
    epsilon <- rnorm(rep, 0, sqrt(0.1))
    y <- numeric(rep)
    for (i in 1:rep){
    y[i] <- x[i] + epsilon[i]           #gives us the 100 observations
    }
    y_hat_model <- lm(y ~ x)
    y_hat <- predict(y_hat_model)
    error <- 1/rep * sum((y_hat-y)^2)   #prediction error of linear regression


    regtree <- rpart(y ~ x, method = "anova", control = list(cp = 0.0001)) #regression tree
    imin <- which.min(regtree$cptable[, "xerror"])

    select <- which(
    regtree$cptable[, "xerror"] <
    sum(regtree$cptable[imin, c("xerror", "xstd")]))[1]

    pregtree <- prune(regtree, cp = regtree$cptable[select, "CP"]) #cost-complexity pruning
    y_hat_tree <- predict(pregtree)
    error_tree <- 1/rep * sum((y_hat_tree-y)^2) #prediction error of regression tree

    return(list(cbind(x, y, y_hat, y_hat_tree), error, error_tree, max(pregtree$cptable[,"nsplit"])))
    #nsplit shows tree size

}
```

Next, we draw the first 100 observations and have a look at the prediction errors and tree size. Then we repeat this process 100 times.

```
outcome <- generate_y(100)
df_outcome <- as.data.frame(outcome[1])
as.numeric(outcome[2:3]) #prediction errors (larger for regression tree)
```

```
## [1] 0.09565 0.08939
```

```
as.numeric(outcome[4]) #number of nodes
```

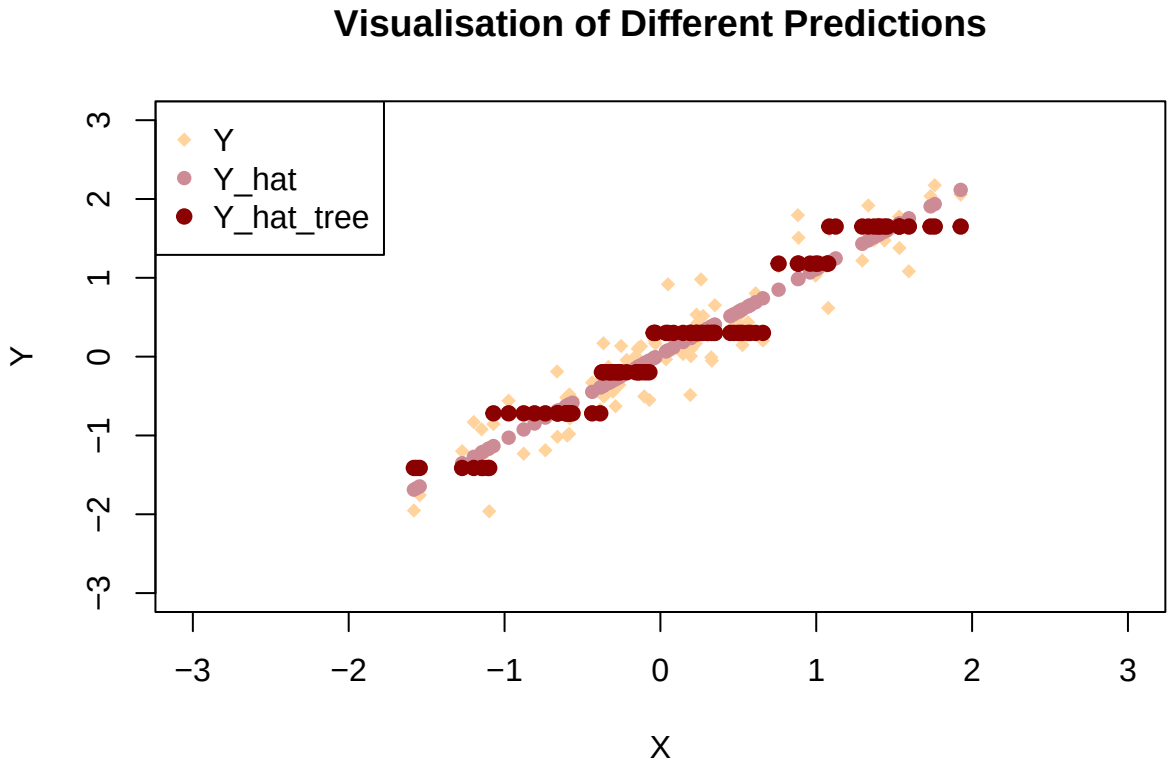
```
## [1] 5
```

```
outcome_99 <- replicate(99, generate_y(100)[2:4])
errors_linear <- outcome_99[1,]
errors_tree <- outcome_99[2,]
nodes <- outcome_99[3,]
```

We visualize one data set together with the fitted predictions using the linear model as well as the tree.

```
plot(x = 1,     #empty plot
     type = "n",
     xlim = c(-3, 3),
     ylim = c(-3, 3),
     pch = 16,
     xlab = "X",
     ylab = "Y",
```

```
     main = "Visualisation of Different Predictions")
points(x = df_outcome$x,
       y = df_outcome$y,
       pch = 18,
       col = "burlywood1")
points(x = df_outcome$x,
       y = df_outcome$y_hat,
       pch = 16,
       col = "lightpink3")
points(x = df_outcome$x,
       y = df_outcome$y_hat_tree,
       pch = 19,
       col = "darkred")
legend("topleft", legend = c("Y", "Y_hat", "Y_hat_tree"), pch = c(18, 16, 19),
       col = c("burlywood1", "lightpink3", "darkred"))
```



In the following, we summarize the results across the 100 repetitions regarding prediction error of the linear model and the fitted tree as well as the tree size.

```
summary(unlist(errors_linear))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0658  0.0871  0.0953  0.0967  0.1083  0.1377
```

```
summary(unlist(errors_tree))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.071   0.105   0.124   0.125   0.139   0.216
```

```
summary(unlist(nodes))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     3.0     4.0     4.0     4.5     5.0     6.0
```

We see that the regression tree results in higher prediction errors. Moreover, the stepwise estimation of the regression tree visible in the plot indicates problems of this estimation method for simple linear relationships. The smaller the steps, the closer it would resemble the real relationship in this case. The regression tree method would probably be more suitable for complex relationships.

Exercise 4

We assume the following data generating process

$$Y = X + \epsilon,$$

where $X \sim N(0, 1)$ and $\epsilon \sim N(0, 1)$

and generate a training data set of size 30.

```
set.seed(123)
X <- rnorm(30)
epsilon <- rnorm(30)

Y <- X + epsilon
training <- as.data.frame(cbind(Y, X))
```

With this training data set, we fit

- a regression tree,
- a linear model with linear effects of X
- and the null model predicting Y through the observed mean in the training set.

```
regtree_train <- rpart(Y ~ X, data = training, method = "anova", control = list(cp = 0.0001))
lm_train <- lm(Y ~ X, data = training)
mean_train <- lm(Y ~ 1, data = training)

regtree_train$cptable
```

```
##       CP nsplit rel error xerror   xstd
## 1 0.4441      0    1.0000 1.1097 0.2538
## 2 0.0001      1    0.5559 0.6787 0.1739
```

```
summary(lm_train)
```

```
##
## Call:
## lm(formula = Y ~ X, data = training)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.609 -0.506 -0.215  0.693  2.012
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)    0.172      0.153    1.12      0.27
## X              0.866      0.159    5.45 0.0000081 ***
```

```
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.839 on 28 degrees of freedom
## Multiple R-squared:  0.515,  Adjusted R-squared:  0.497
## F-statistic: 29.7 on 1 and 28 DF,  p-value: 0.0000081
```

```
summary(mean_train)
```

```
##
## Call:
## lm(formula = Y ~ 1, data = training)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.564 -0.955 -0.151  0.765  2.323
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.131      0.216    0.61     0.55
##
## Residual standard error: 1.18 on 29 degrees of freedom
```

```
mean(Y) #same as Y~1
```

```
## [1] 0.1312
```

Then we determine the mean squared error of the three models on a test sample of size 10,000

```
X_test <- rnorm(10000)
Y_test <- X_test + rnorm(10000)
test <- as.data.frame(cbind(Y_test, X_test))
colnames(test) <- c("Y", "X")

regtree_test <- predict(regtree_train, newdata = test)
lm_test <- predict(lm_train, newdata = test)
mean_test <- predict(mean_train, newdata = test)

MSE <- function(Y_hat){
    error <- 1/10000 * sum((Y_hat-Y_test)^2)
    return(error)
}

MSE(regtree_test)
```

```
## [1] 1.376
```

```
MSE(lm_test)
```

```
## [1] 1.055
```

```
MSE(mean_test)
```

```
## [1] 1.999
```

The mean squared error is highest for the null model and lowest for the linear model. Now, we add to the

training and test data 20 covariates Z_1,...,Z_20 with

$$Z_i \sim \sqrt{0.9}X + \epsilon_{Z_i},$$

where $\epsilon_{Z_i} \sim N(0, 0.11)$.

```
Z <- numeric(20)
covariates <- function(Z){
    X <-  rnorm(20)
    for(i in 1:20){
    Z[i] <- sqrt(0.9) * X[i] + rnorm(1, 0, sqrt(0.1))
    }
    return(Z)
}

test_added <- as.data.frame(t(replicate(10000, covariates(Z))))
train_added <- as.data.frame(t(replicate(30, covariates(Z))))

#merge datasets
training <- cbind(training, train_added)
test <- cbind(test, test_added)
```

Now we use the trainig data to estimate

- a regression tree,
- a linear model with linear effects of X and all Z
- and the null model predicting Y through the observed mean in the training set.

Moreover, we estimate a linear model potentially including linear effects for X and all Z variables, but use model selection with the AIC to select a suitable model starting from the null model.

```
regtree_train2 <- rpart(Y ~ ., data = training, method = "anova", control = list(cp = 0.0001))
lm_train2 <- lm(Y ~ ., data = training)
mean_train2 <- lm(Y ~ 1, data = training)

regtree_train2$cptable
```

```
##       CP nsplit rel error xerror   xstd
## 1 0.4441      0    1.0000 1.0682 0.2495
## 2 0.0001      1    0.5559 0.6036 0.1487
```

```
summary(lm_train2)
```

```
##
## Call:
## lm(formula = Y ~ ., data = training)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6403 -0.1390  0.0246  0.2219  0.6085
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.1683     0.3020    0.56  0.59258
## X             1.2414     0.2240    5.54  0.00055 ***
## V1            0.7511     0.4421    1.70  0.12774
```

```
## V2           0.5873     0.2395    2.45  0.03980 *
## V3          -1.0357     0.4150   -2.50  0.03718 *
## V4           0.4877     0.1939    2.51  0.03609 *
## V5          -0.1829     0.2701   -0.68  0.51751
## V6           0.5247     0.2290    2.29  0.05118 .
## V7           0.7259     0.2820    2.57  0.03292 *
## V8           0.2358     0.1898    1.24  0.24934
## V9           0.3693     0.1901    1.94  0.08797 .
## V10         -0.0533     0.1597   -0.33  0.74695
## V11          0.5731     0.2572    2.23  0.05645 .
## V12         -0.1643     0.1994   -0.82  0.43388
## V13         -0.2059     0.2016   -1.02  0.33690
## V14         -0.8248     0.2288   -3.61  0.00693 **
## V15         -0.4526     0.3208   -1.41  0.19595
## V16          0.9350     0.4371    2.14  0.06485 .
## V17          0.3092     0.2252    1.37  0.20714
## V18         -0.5761     0.3679   -1.57  0.15602
## V19         -0.1124     0.2128   -0.53  0.61181
## V20         -0.8652     0.3101   -2.79  0.02356 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6 on 8 degrees of freedom
## Multiple R-squared:  0.929,  Adjusted R-squared:  0.743
## F-statistic:    5 on 21 and 8 DF,  p-value: 0.0124
```

```
summary(mean_train2)
```

```
##
## Call:
## lm(formula = Y ~ 1, data = training)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.564 -0.955 -0.151  0.765  2.323
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.131      0.216    0.61     0.55
##
## Residual standard error: 1.18 on 29 degrees of freedom
```

```
mean(Y)
```

```
## [1] 0.1312
```

```
lm_AIC <- stepAIC(mean_train2, #our start point
        direction = "both", #in order to avoid multicollinearity,
                            #allow algorithm to go back and exclude variables
                            #(We double-checked that "forward" would yield the same result)
        scope = list(upper=lm_train2, lower=mean_train2), k = 2, trace = FALSE) #k=2 gives the AIC
#Algorithm stops when <none> (i.e. leaving model as it is) yields the lowest AIC.

summary(lm_AIC) #6 regressors are included
```

```
##
## Call:
## lm(formula = Y ~ X + V9 + V20 + V13, data = training)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6167 -0.4167 -0.0941  0.2396  1.3063
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)    0.113      0.157    0.72     0.477
## X              0.898      0.145    6.20 0.0000018 ***
## V9             0.358      0.164    2.18     0.039 *
## V20           -0.377      0.151   -2.50     0.019 *
## V13            0.292      0.123    2.37     0.026 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.718 on 25 degrees of freedom
## Multiple R-squared:  0.683,  Adjusted R-squared:  0.632
## F-statistic: 13.5 on 4 and 25 DF,  p-value: 0.00000551
```

Again, we determine the mean squared error of the models on the test sample of size 10,000.

```
regtree_test2 <- predict(regtree_train2, newdata = test)
lm_test2 <- predict(lm_train2, newdata = test)
mean_test2 <- predict(mean_train2, newdata = test)
lm_AIC_test <- predict(lm_AIC, newdata = test)

MSE(regtree_test2)
```

```
## [1] 1.376
```

```
MSE(lm_test2)
```

```
## [1] 7.746
```

```
MSE(mean_test2)
```

```
## [1] 1.999
```

```
MSE(lm_AIC_test)
```

```
## [1] 1.376
```

Now that we have a non-linear relationship between the dependent and independent variables, a simple linear fit including all variables gives us the highest error whereas the regression tree performs best. Also, regression trees apparently deal better with the given multicollinearity problem. Also, the AIC method performs quite well by taking into account both model fit and the number of predictors.

Exercise 5

We assume data with the following data generation process:

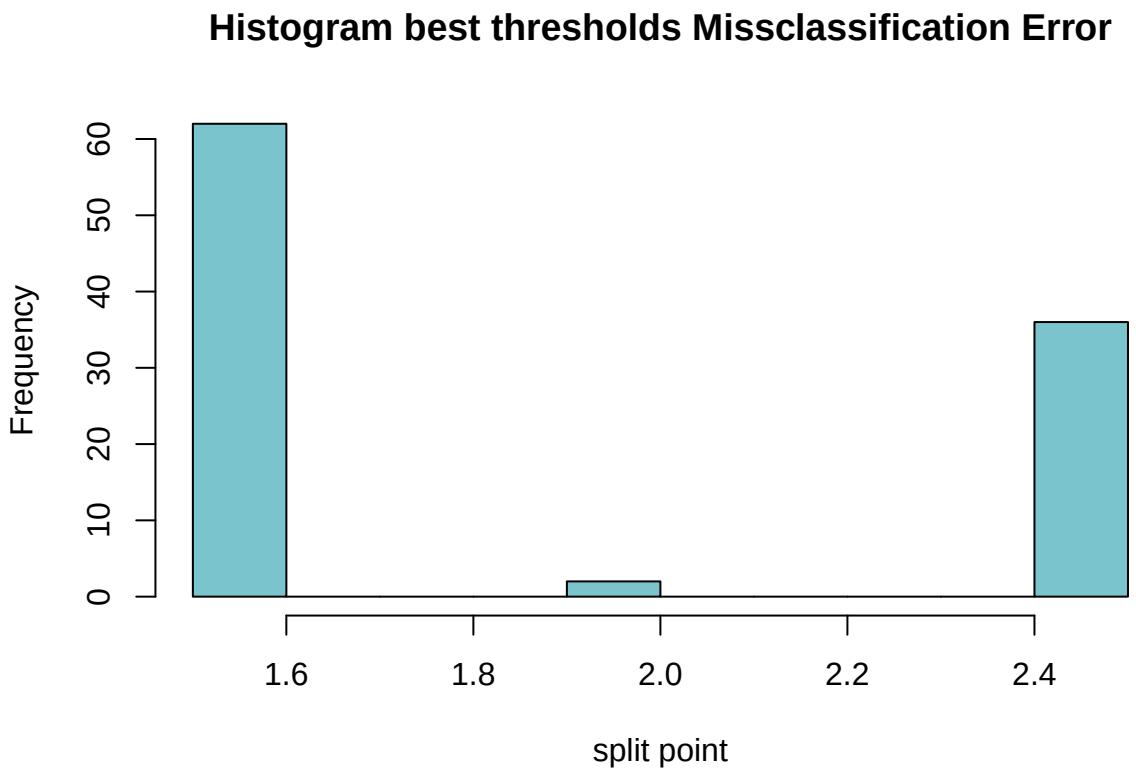
$$x = y + \epsilon,$$

where y is a categorical variable with values 1, 2, 3, which occur with equal probability and $\epsilon \sim \mathcal{N}(0, 0.2)$ independent.

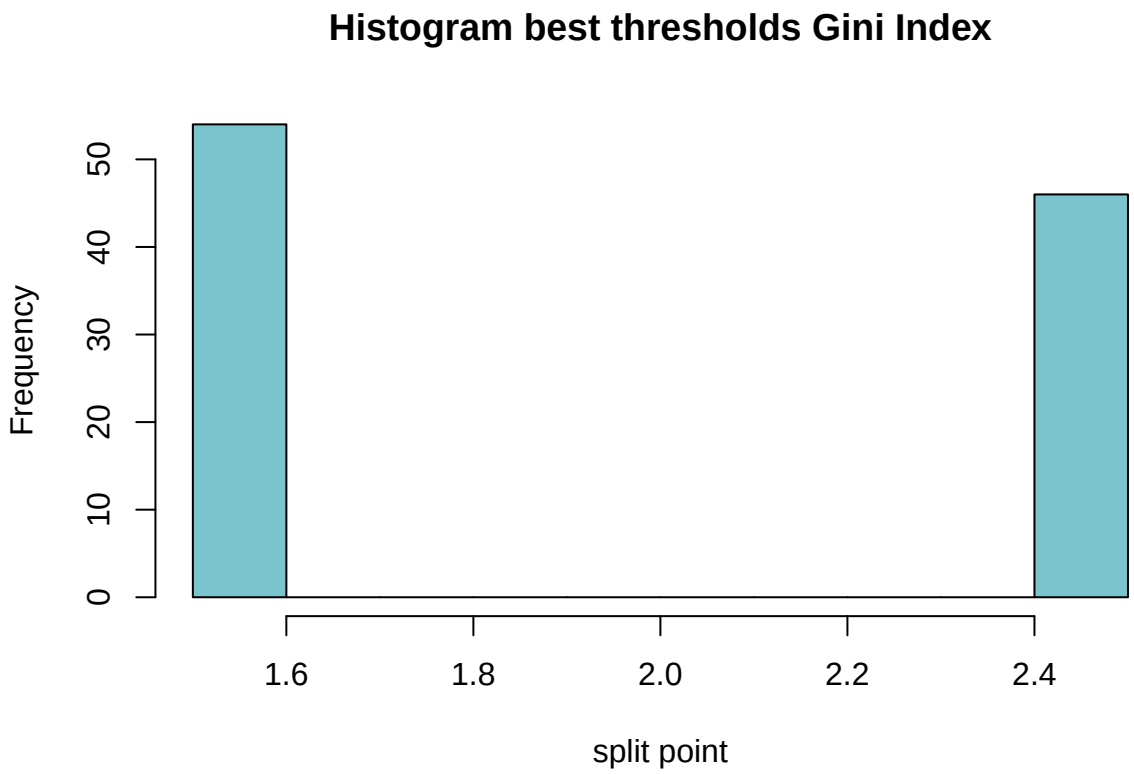
- Draw 100 data sets of size 100.
- Determine the sum of the misclassification rates, Gini indices and deviance criteria weighted with the number of observations in each subgroup for the subgroups obtained when splitting the observations using x with thresholds 1.5, 2, and 2.5 and y as dependent variable in the classification problem.
- Calculate the best threshold according to each of the three impurity measures for each of the 100 data sets. Summarize and interpret the results.

```
n <- 100
rep <- 100
splitpoints <- c(1.5,2,2.5)
best <- matrix(nrow=100,ncol=3)
ME_matrix <- matrix(nrow=100,ncol=3)
gini_matrix <- matrix(nrow=100,ncol=3)
deviance_matrix <- matrix(nrow=100,ncol=3)

for (i in 1:rep){
  y <- sample(c(1,2,3),size=n,replace=TRUE,prob = c(1/3,1/3,1/3))
  epsilon <- rnorm(100, 0,0.2)
  x <- y+epsilon
  ME_best <- c(Inf,0)
  gini_best <- c(Inf,0)
  deviance_best <- c(Inf,0)
  for(k in c(1,2,3)){
    j <- splitpoints[k]
    group1 <- y[which(x < j)]
    group2 <- y[which(x >= j)]
    p11 <- length(which(group1==1))/length(group1)
    p12 <- length(which(group1==2))/length(group1)
    p13 <- length(which(group1==3))/length(group1)
    p21 <- length(which(group2==1))/length(group2)
    p22 <- length(which(group2==2))/length(group2)
    p23 <- length(which(group2==3))/length(group2)
    class1 <- which.max(c(p11,p12,p13))
    class2 <- which.max(c(p21,p22,p23))
    ME_matrix[i,k] <- ME <- sum(group1 != class1)+sum(group2 != class2)
    if(ME_best[1] > ME){
      ME_best[1] <- ME
      ME_best[2] <-j
    }
    gini_matrix[i,k] <- gini <- length(group1)*(p11*(1-p11)+p12*(1-p12)+p13*(1-p13))+length(group2)*(p2
    if(gini_best[1] > gini ){
      gini_best[1] <- gini
      gini_best[2] <-j
    }
    #note that zero probabilities drop out before taking the log
    deviance_matrix[i,k] <- deviance <- -length(group1)*(p11*log(ifelse(p11!=0,p11,1))+p12*log(ifelse(p
    if(deviance_best[1] > deviance ){
      deviance_best[1] <- deviance
      deviance_best[2] <-j
    }

  }
  best[i,] <- c(ME_best[2],gini_best[2],deviance_best[2])
```

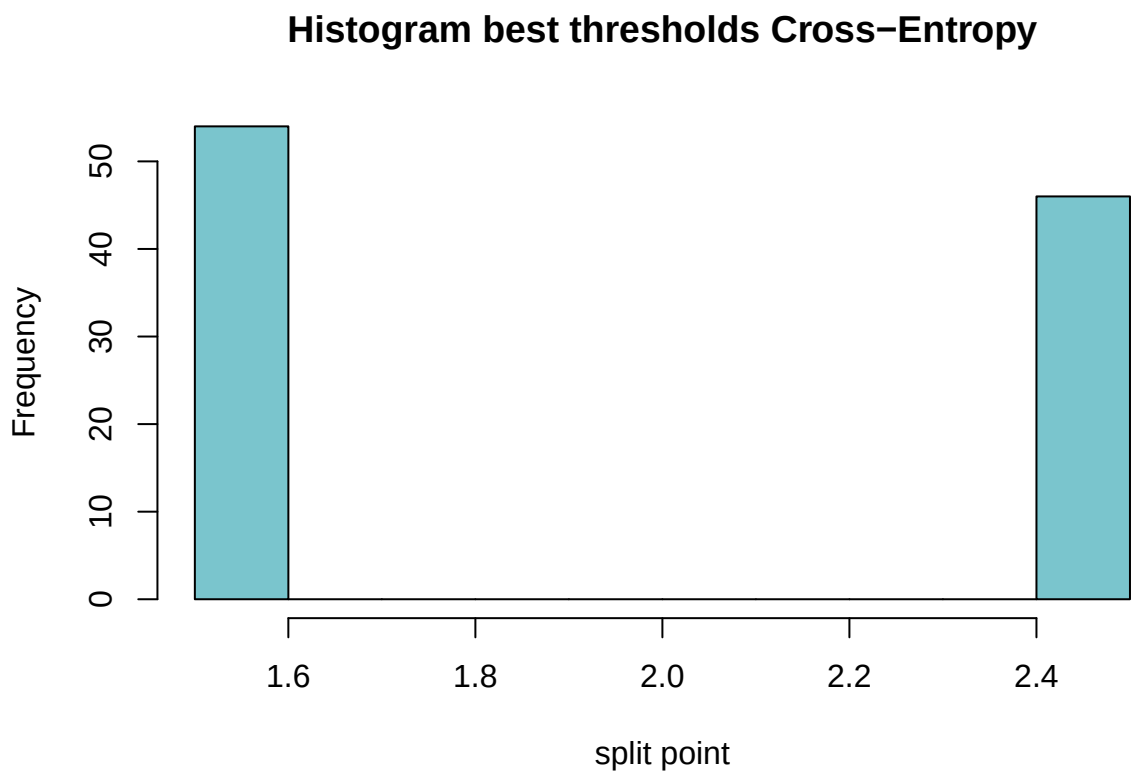
```
}
hist(best[,1], main="Histogram best thresholds Missclassification Error",xlab="split point",col="cadetbl
```



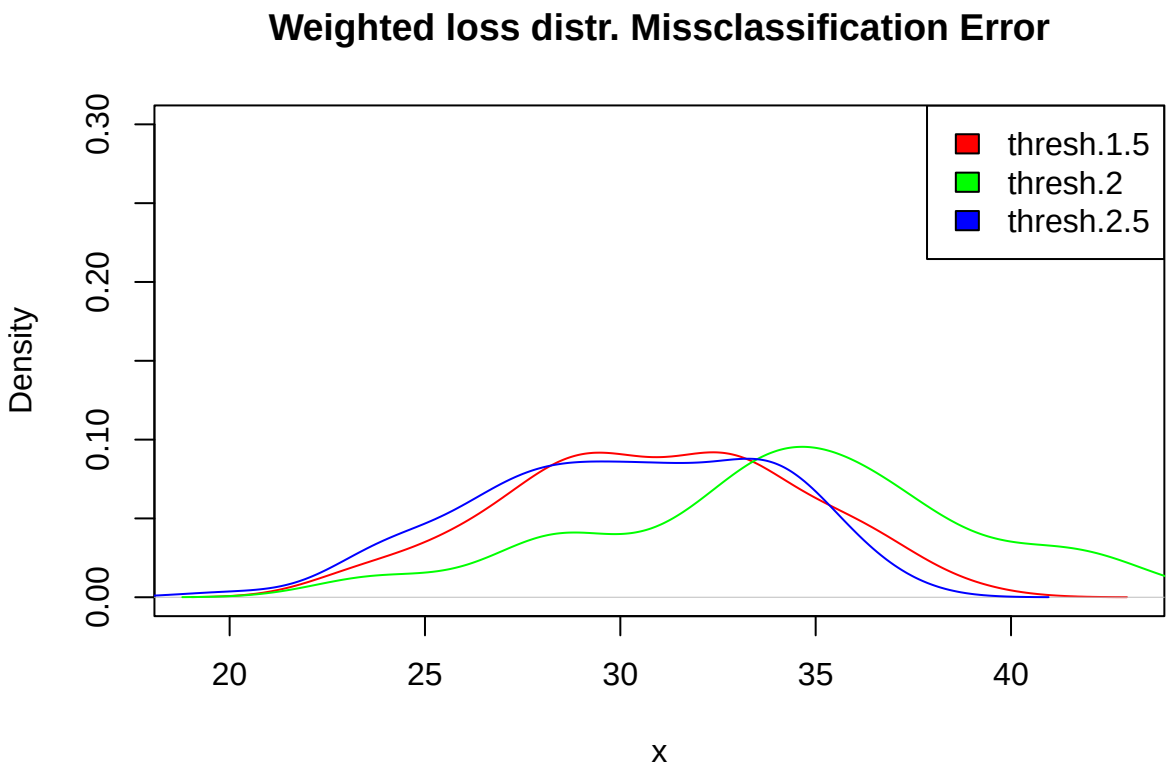

```
hist(best[,2], main="Histogram best thresholds Gini Index",xlab="split point",col="cadetblue3")
```



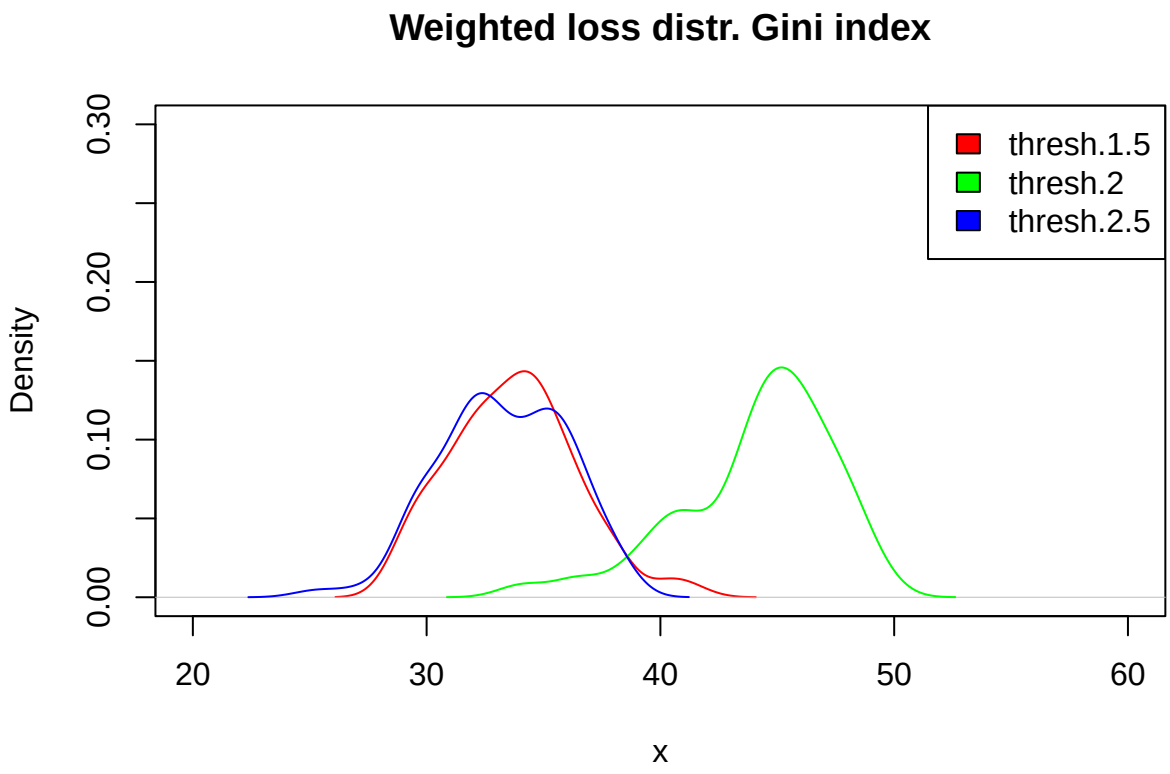

```
hist(best[,3], main="Histogram best thresholds Cross-Entropy",xlab="split point",col="cadetblue3")
```



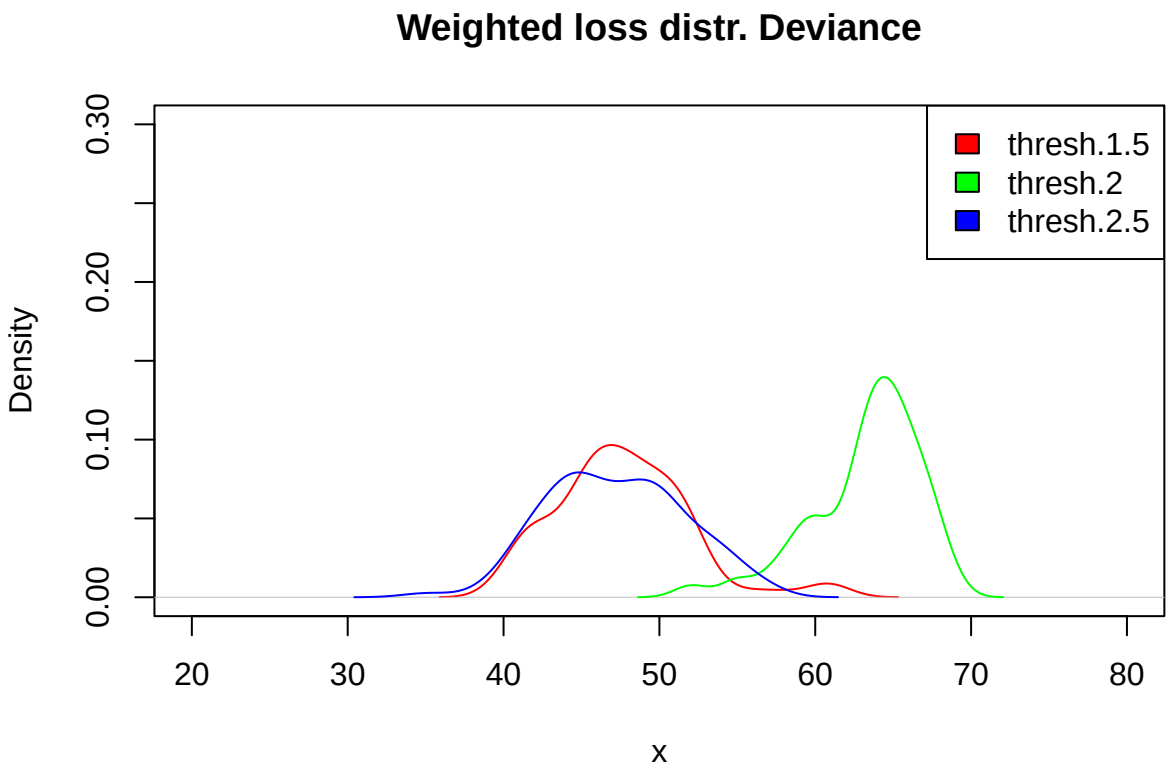

```
plot(density(ME_matrix[,1]), col='red', ylim=c(0,0.3), main='Weighted loss distr. Missclassification Er
lines(density(ME_matrix[,2]), col='green')
lines(density(ME_matrix[,3]), col='blue')

#add legend
legend('topright', c('thresh.1.5', 'thresh.2', 'thresh.2.5'), fill=c('red', 'green','blue'))
```




```
plot(density(gini_matrix[,1]), col='red', ylim=c(0,0.3), xlim=c(20,60),main='Weighted loss distr. Gini
lines(density(gini_matrix[,2]), col='green')
lines(density(gini_matrix[,3]), col='blue')

#add legend
legend('topright', c('thresh.1.5', 'thresh.2', 'thresh.2.5'), fill=c('red', 'green','blue'))
```




```
plot(density(deviance_matrix[,1]), col='red', ylim=c(0,0.3),xlim=c(20,80), main='Weighted loss distr. D
lines(density(deviance_matrix[,2]), col='green')
lines(density(deviance_matrix[,3]), col='blue')

#add legend
legend('topright', c('thresh.1.5', 'thresh.2', 'thresh.2.5'), fill=c('red', 'green','blue'))
```



Note that cross-entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate, which also prefers the threshold 2 in some of the cases. The first threshold 1.5 mostly produces a pure group 1 of only ones and the threshold 2.5 produces a pure group 2 of only threes, when classifying observations to the majority class in the node. The missclassification rate however, prefers the threshold 2 in some cases, even though the splits producing a pure node are probably preferable. Both the Gini index and cross-entropy are lower for the second split.

##Exercise 6

Assume that the true data generation process is given by:

$$Pr(Y = 1|x_1, x_2) = \pi(x_1, x_2),$$

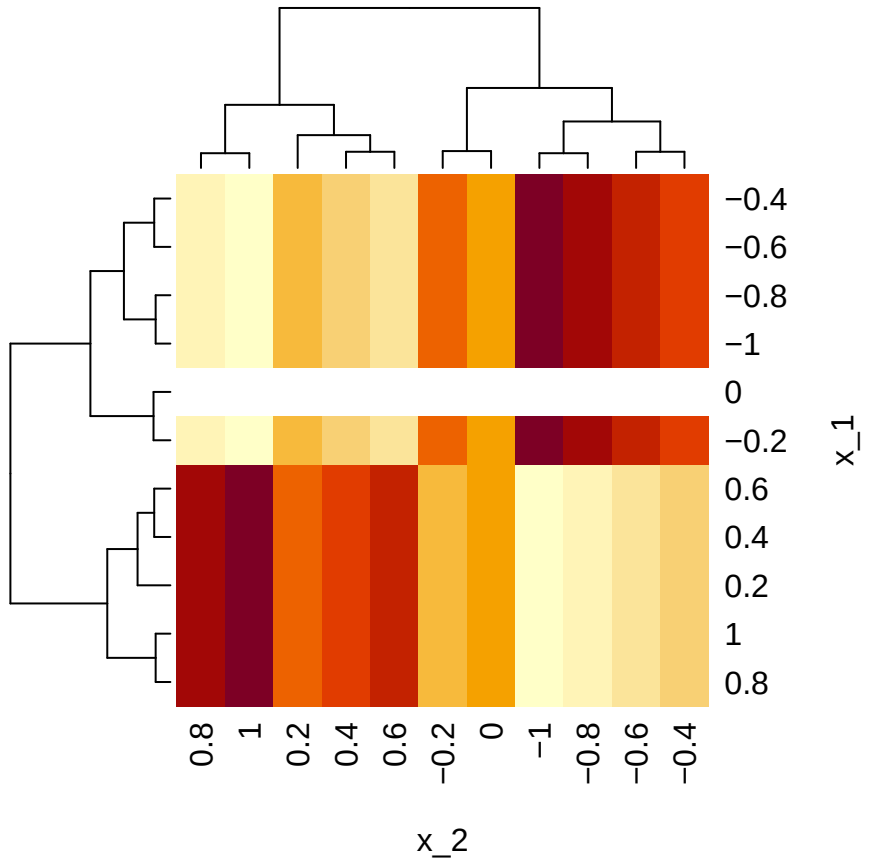
$$logit(\pi) = x_1 \cdot x_2,$$

where $x_i \sim U(1, 1)$ for $i = 1, 2$ and independent.

- Visualize the function $\pi(x_1, x_2)$ in dependence of x_1 and x_2, e.g., using an image plot.
- Draw a data set of size 1000 from the data generation process.
- Fit a logistic regression model using main and interaction effects.
- Fit a classification tree with maximum depth 5 using as impurity measures: - Gini. - Entropy / deviance. Compare the two trees.
- Visualize the fitted regression functions for the logistic regression and each of the two trees and compare them.

```
#task 1
sigmoid <- function(x){
  1/(1+exp(-x))
```

```
}

x_1_plot <- seq(-1,1,0.2)
x_2_plot <- seq(-1,1,0.2)

logit_plot <- sigmoid(outer(x_1_plot,x_2_plot))
colnames(logit_plot) <- x_1_plot
rownames(logit_plot) <- x_2_plot
heatmap(logit_plot,ylab="x_1",xlab="x_2")
```



We observe that the highest value ≈ 0.73 is obtained when $(x_1, x_2) = (1, 1)$ or $(x_1, x_2) = (-1, -1)$ and that the lowest value ≈ 0.27 is obtained when $(x_1, x_2) = (-1, 1)$ or vice versa. In general the range of the sigmoid function is $[0, 1]$

```
#task 2
x_1 <- runif(1000,-1,1)
x_2 <- runif(1000,-1,1)

#test <- expand.grid(x_1_plot,x_2_plot)
sample <- numeric(1000)
for( i in 1:1000){
  sample[i] <- as.numeric(names(sort(table(rbinom(1000,1,sigmoid(x_1[i]*x_2[i]))), decreasing = TRUE)[1]
}
#task 3
log_model <- glm(sample ~ x_1+x_2, family = 'binomial')
summary(log_model)
```

```
##
## Call:
## glm(formula = sample ~ x_1 + x_2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
##  -1.23   -1.15   -1.10    1.18    1.33
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.0584     0.0636   -0.92    0.358
## x_1          -0.1870     0.1099   -1.70    0.089 .
## x_2          -0.1413     0.1110   -1.27    0.203
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1385.7  on 999  degrees of freedom
## Residual deviance: 1381.3  on 997  degrees of freedom
## AIC: 1387
##
## Number of Fisher Scoring iterations: 3
```

```
data <- as.data.frame(cbind(sample,x_1,x_1))
log_model_interact <- glm(sample ~ x_1+x_2+x_1*x_2, data=data,family = 'binomial')
summary(log_model_interact)
```

```
##
## Call:
## glm(formula = sample ~ x_1 + x_2 + x_1 * x_2, family = "binomial",
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5499  -0.1292   0.0000   0.0855   2.6372
##
## Coefficients:
##             Estimate Std. Error z value            Pr(>|z|)
## (Intercept)    0.101      0.128    0.79                0.43
## x_1            0.329      0.344    0.96                0.34
## x_2           -0.044      0.327   -0.13                0.89
## x_1:x_2       27.544      2.376   11.59 <0.0000000000000002
##
## (Intercept)
## x_1
## x_2
## x_1:x_2     ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 1385.72  on 999  degrees of freedom
## Residual deviance:  387.69  on 996  degrees of freedom
## AIC: 395.7
##
## Number of Fisher Scoring iterations: 8
```

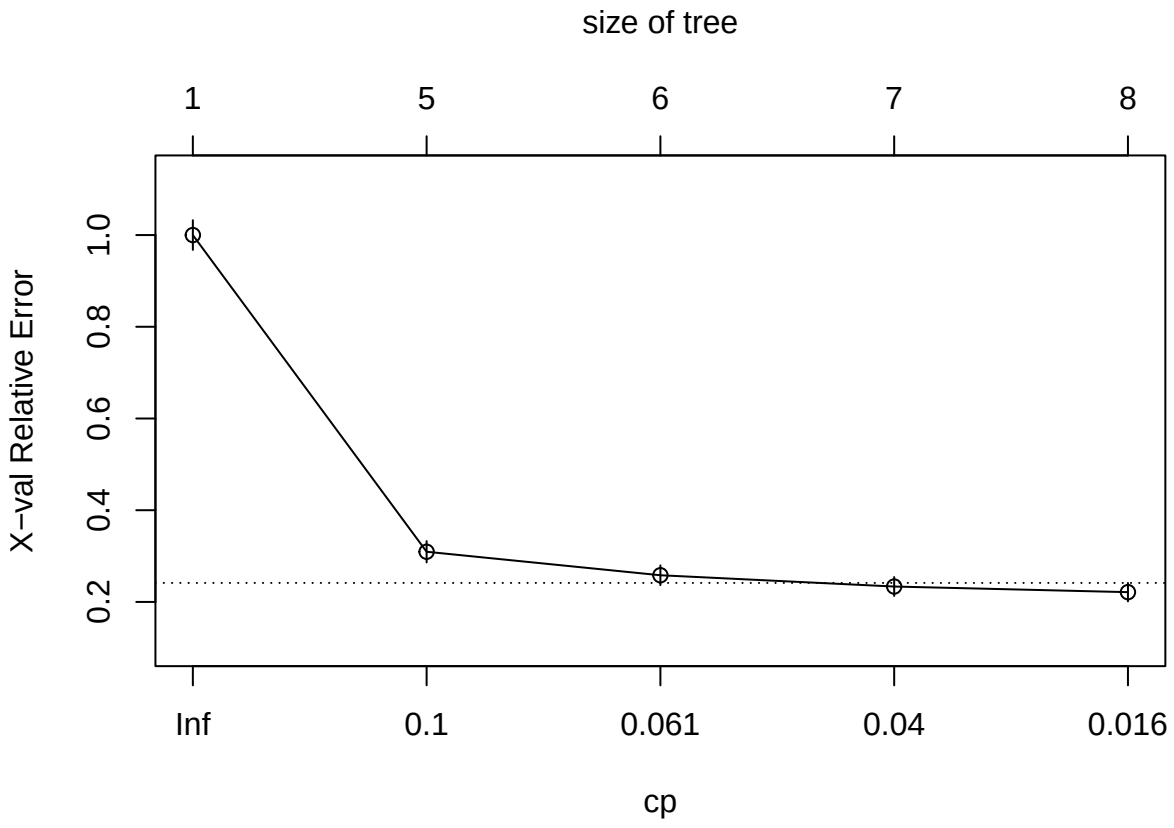
```
slope <- coef(log_model_interact)[2]/(-coef(log_model_interact)[3])
intercept <- coef(log_model_interact)[1]/(-coef(log_model_interact)[3])


#task 4
library("rpart")
library("plot3D")
```

```
## Warning: Paket 'plot3D' wurde unter R Version 4.1.3 erstellt
```

```
tree_gini <- rpart(sample ~ x_1+x_2,
  method = "class", parms = list(split = "gini"),
  control = list(maxdepth=5))

printcp(tree_gini)
```

```
##
## Classification tree:
## rpart(formula = sample ~ x_1 + x_2, method = "class", parms = list(split = "gini"),
##     control = list(maxdepth = 5))
##
## Variables actually used in tree construction:
## [1] x_1 x_2
##
## Root node error: 488/1000 = 0.49
##
## n= 1000
##
##      CP nsplit rel error xerror  xstd
## 1 0.165      0      1.00   1.00 0.032
## 2 0.064      4      0.34   0.31 0.023
## 3 0.059      5      0.28   0.26 0.022
## 4 0.027      6      0.22   0.23 0.021
## 5 0.010      7      0.19   0.22 0.020
```

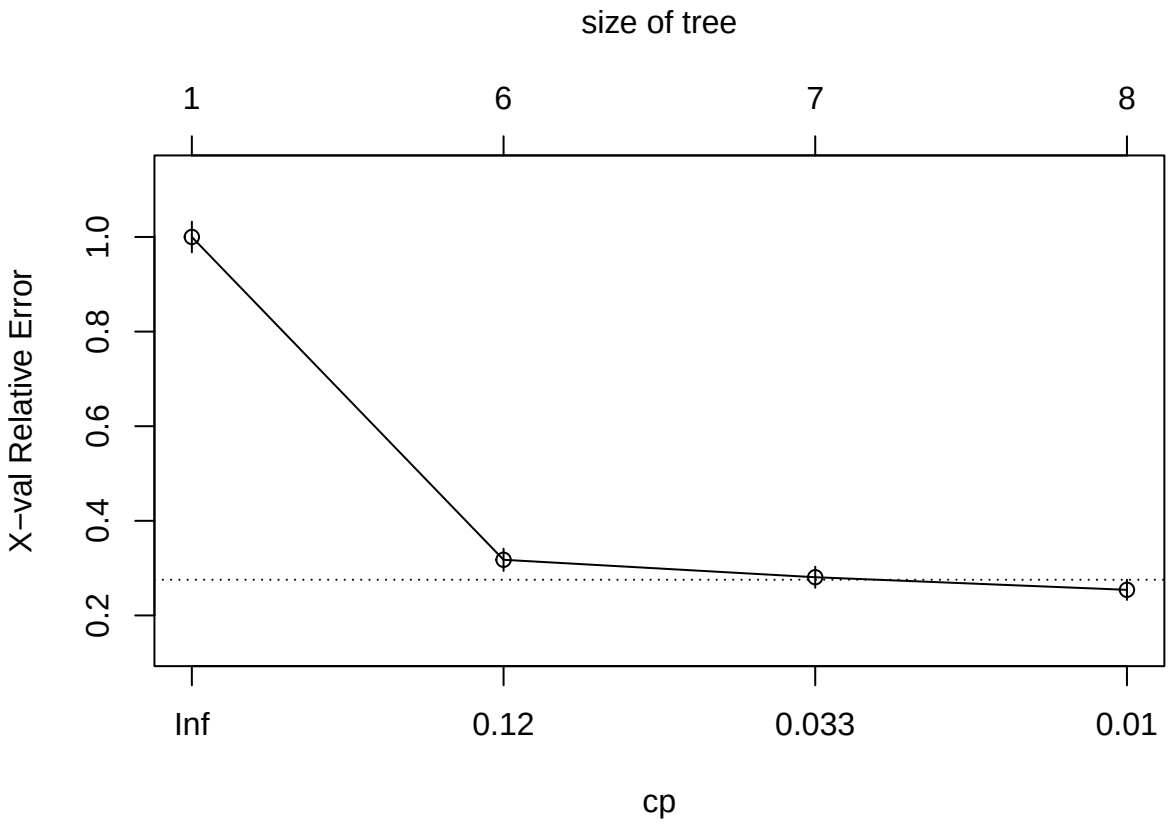
```
plotcp(tree_gini)
```




```
tree_entropy <- rpart(sample ~ x_1+x_2,
  method = "class", parms = list(split = "information"),
  control = list(maxdepth=5))

printcp(tree_entropy)
```

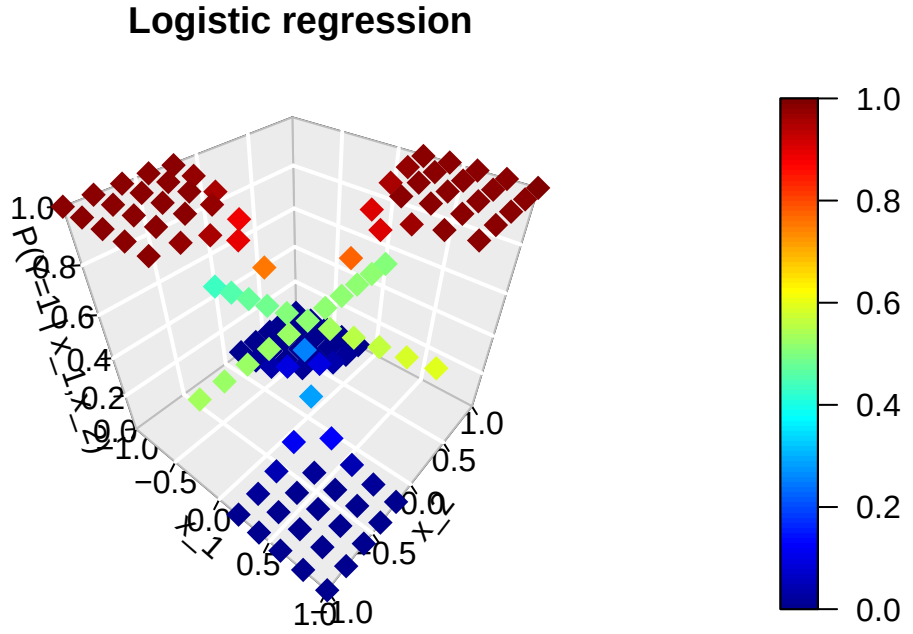
```
##
## Classification tree:
## rpart(formula = sample ~ x_1 + x_2, method = "class", parms = list(split = "information"),
##     control = list(maxdepth = 5))
##
## Variables actually used in tree construction:
## [1] x_1 x_2
##
## Root node error: 488/1000 = 0.49
##
## n= 1000
##
##     CP nsplit rel error xerror  xstd
## 1 0.13      0      1.00   1.00 0.032
## 2 0.11      5      0.33   0.32 0.023
## 3 0.01      6      0.22   0.28 0.022
## 4 0.01      7      0.21   0.25 0.021
```

```
plotcp(tree_entropy)
```

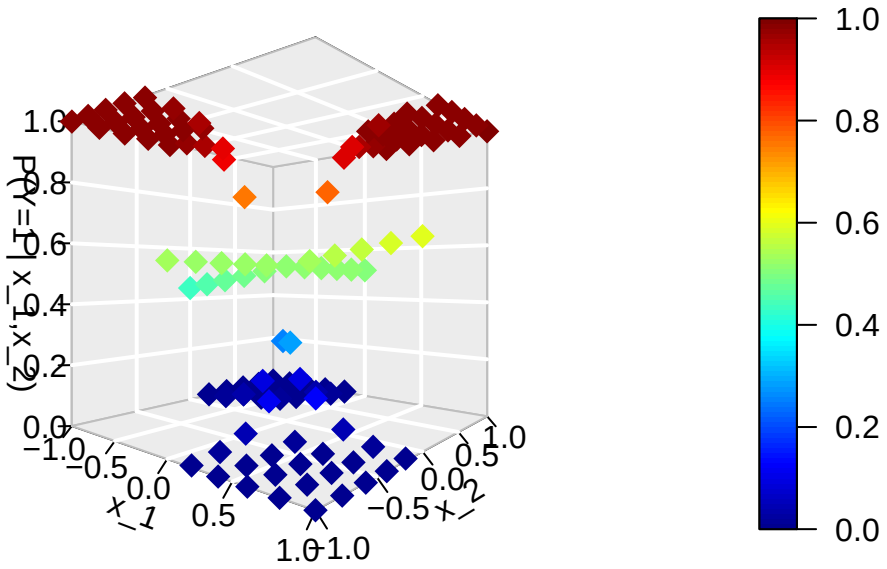


Finally, we visualize the fitted regression functions for the logistic regression and each of the two trees and compare them.

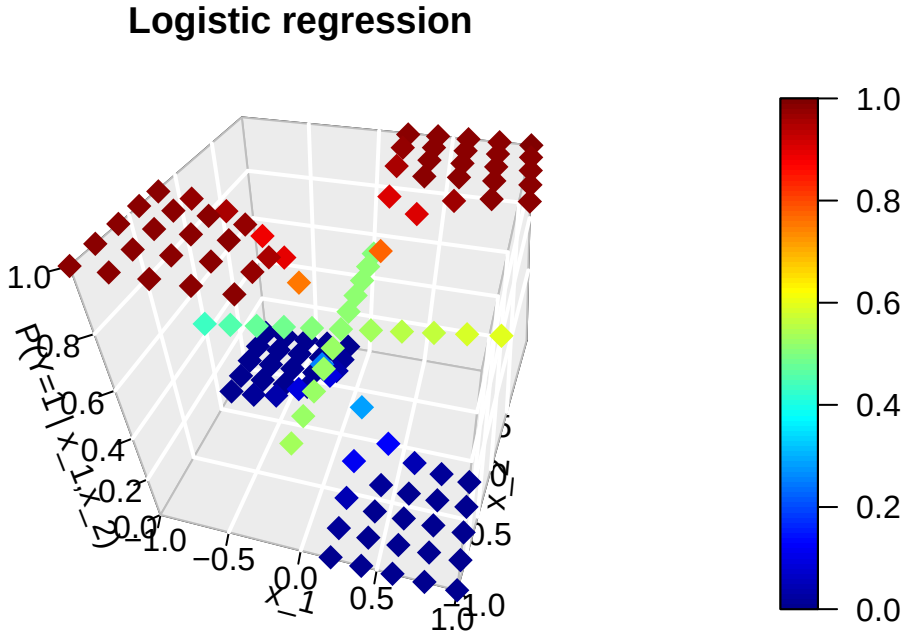
```
#we use type="response" for the glm in order to obtain predicted probabilities rather than log-odds
expand <- expand.grid(x_1_plot,x_2_plot)

plot_matrix <- numeric(nrow(expand))

for(i in 1:nrow(expand)){
  plot_matrix[i] <- predict(log_model_interact,newdata = data.frame(x_1=expand[i,1],x_2=expand[i,2]),ty
}

scatter3D(expand[,1],expand[,2],plot_matrix,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)",main =
```



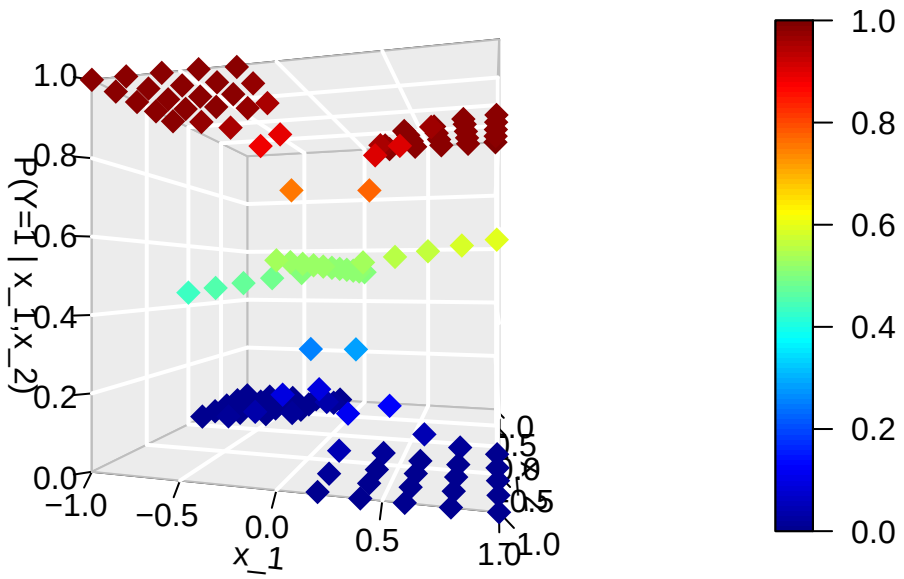

```
scatter3D(expand[,1],expand[,2],plot_matrix,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)",main =
```



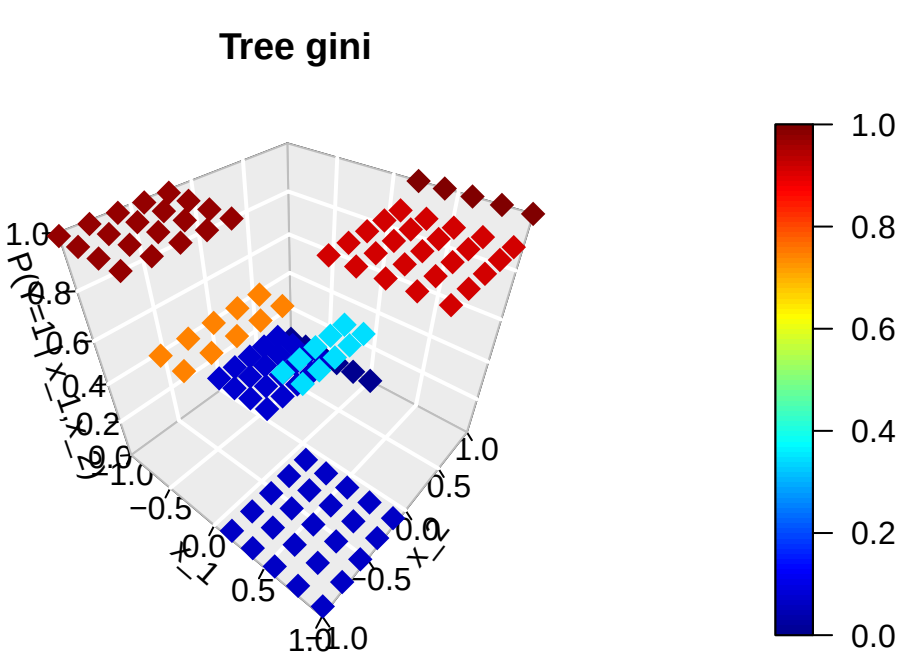

```
scatter3D(expand[,1],expand[,2],plot_matrix,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)",main =
```



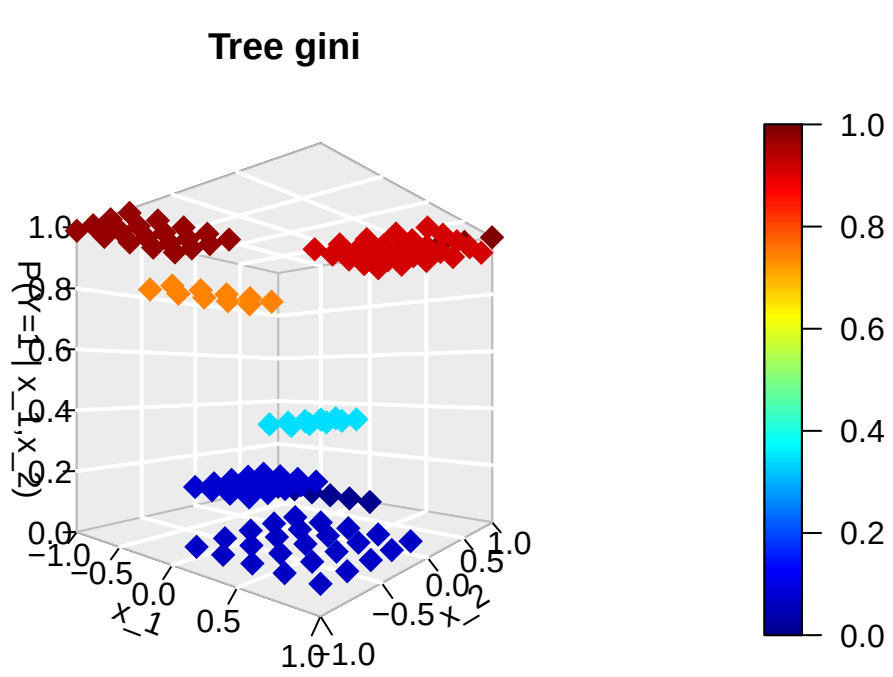

```
scatter3D(expand[,1],expand[,2],plot_matrix,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)",main =
```



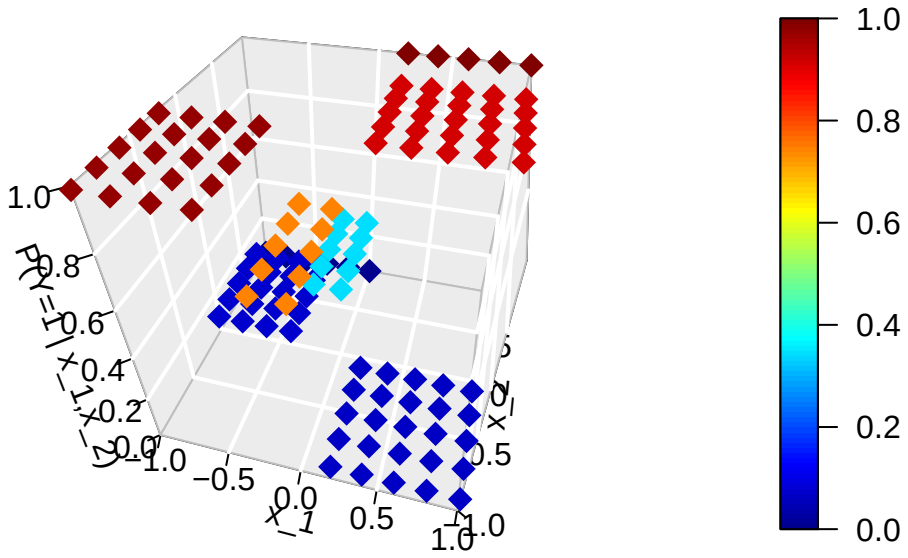

```
plot_matrix_gini <- numeric(nrow(expand))

for(i in 1:nrow(expand)){
  plot_matrix_gini[i] <- predict(tree_gini,newdata = data.frame(x_1=expand[i,1],x_2=expand[i,2]),type="
}

scatter3D(expand[,1],expand[,2],plot_matrix_gini,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)",ma
```



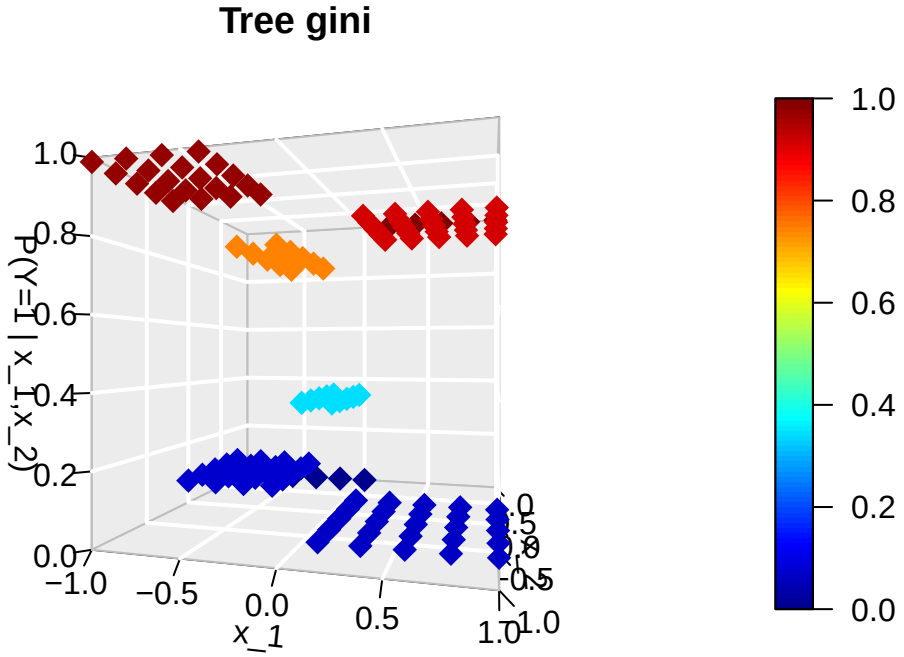

```
scatter3D(expand[,1],expand[,2],plot_matrix_gini,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)",ma
```



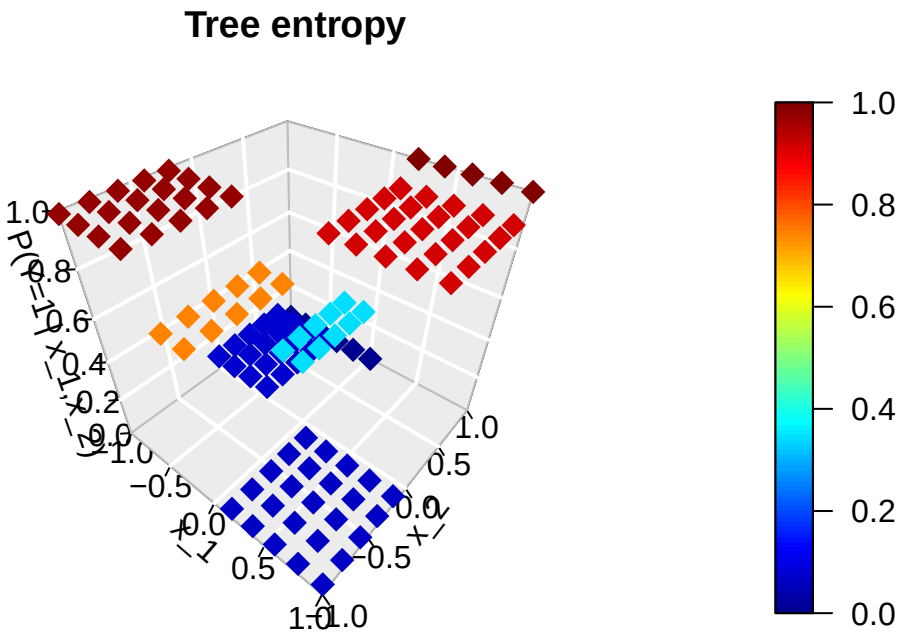

```
scatter3D(expand[,1],expand[,2],plot_matrix_gini,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)",ma
```



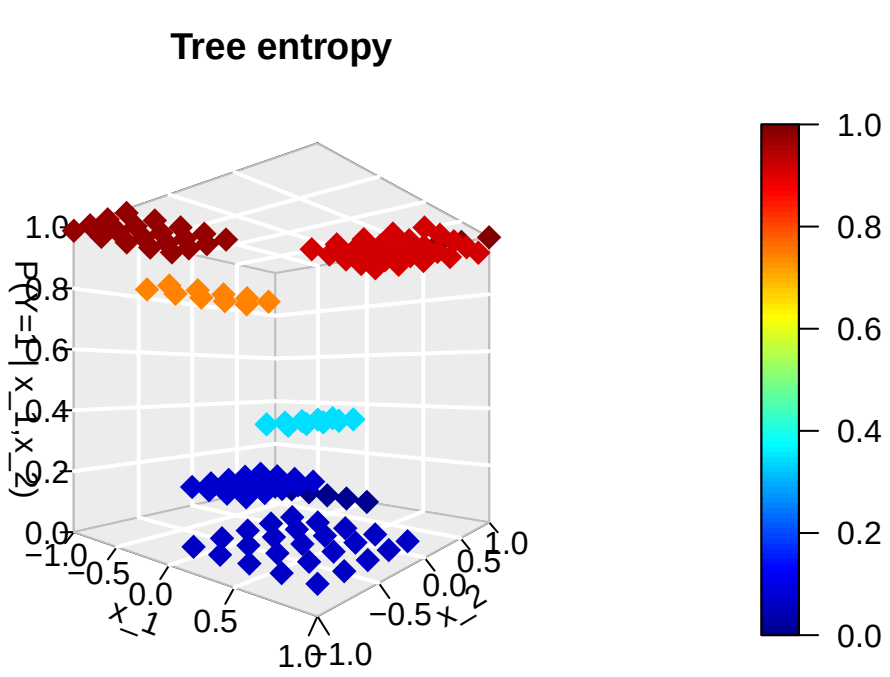

```
scatter3D(expand[,1],expand[,2],plot_matrix_gini,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)",ma
```



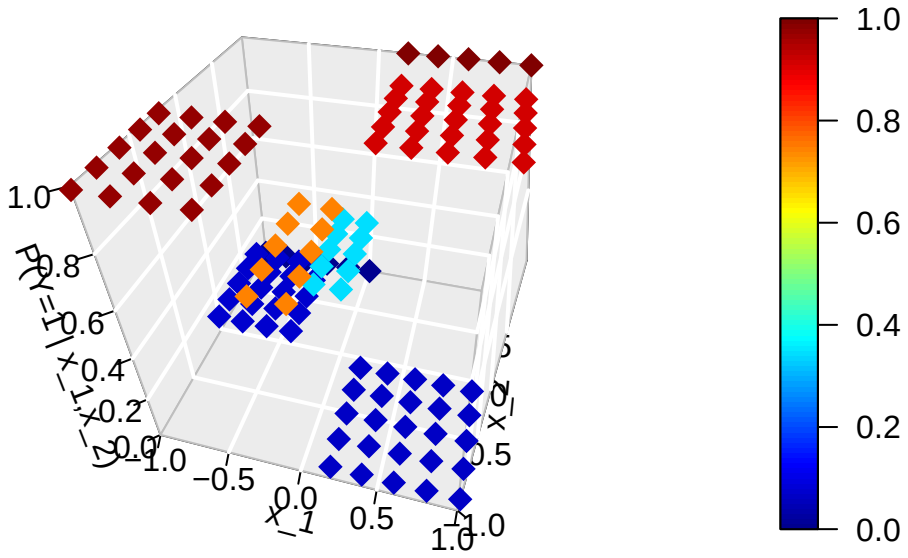

```
plot_matrix_entropy <- numeric(nrow(expand))

for(i in 1:nrow(expand)){
  plot_matrix_entropy[i] <- predict(tree_gini,newdata = data.frame(x_1=expand[i,1],x_2=expand[i,2]),typ
}

scatter3D(expand[,1],expand[,2],plot_matrix_entropy,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)
```



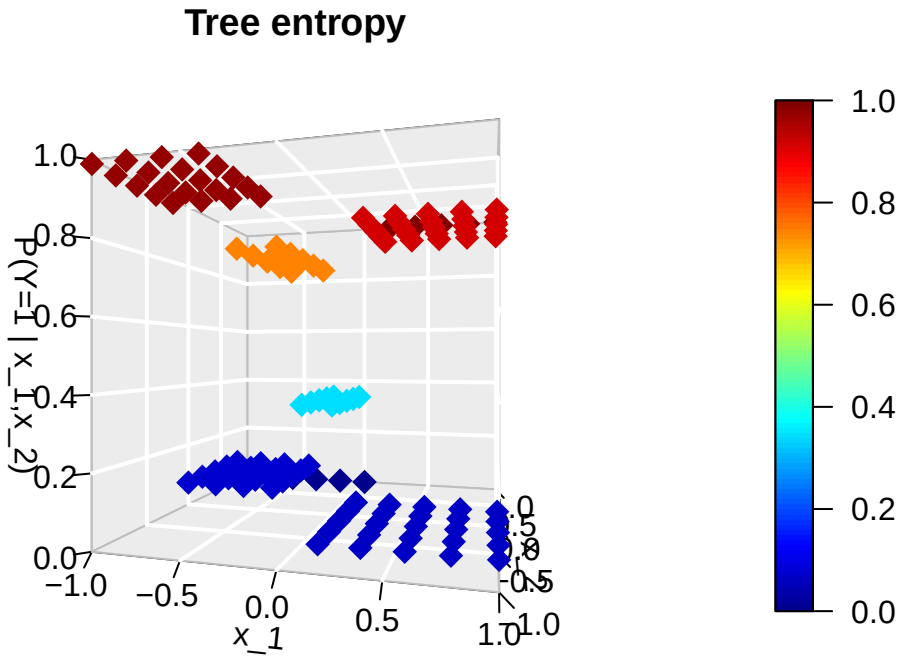

```
scatter3D(expand[,1],expand[,2],plot_matrix_entropy,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)
```




```
scatter3D(expand[,1],expand[,2],plot_matrix_entropy,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)
```




```
scatter3D(expand[,1],expand[,2],plot_matrix_entropy,pch=18,xlab="x_1",ylab="x_2",zlab="P(Y=1 | x_1,x_2)
```



We observe that (as we know from the lecture) the trees fitted a discontinuous function.