Exercise Set 2 - solutions and grading

Please read the following before doing the peer reviews:

- Problem sheet at https://moodle.helsinki.fi/pluginfile.php/3324196/mod_folder/content/0/DATA1100 2-2020-E2.pdf?forcedownload=1
- General grading instructions at https://moodle.helsinki.fi/mod/page/view.php?id=2072012
- Step-by-step instructions for the peer review week at https://moodle.helsinki.fi/mod/page/view.php?i d=2084536

Please participate to one peer review session during 24-26 November and submit your peer reviews on 27 November, at latest.

Problem 9

Task a

The 1-dimensional normal distribution has the probability density

$$f_k(x) = P(X = x \mid Y = k) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right).$$

The class probabilitues are given by $\pi_k = P(Y = k)$. By Bayes rule we can write the posterior probability $P(Y = k \mid X = x)$ as

$$P(Y = k \mid X = x) = \frac{P(X = x \mid Y = k)P(Y = k)}{P(X = x)} = \frac{P(Y = y)P(X = x \mid Y = k)}{\sum_{k'=1}^{k} P(X = x \mid Y = k')P(Y = k')}.$$

Or:

$$P(Y = k \mid X = x) = \frac{f_k(x) \times \pi_k}{\sum_{k'=1}^{K} f_{k'}(x) \times \pi_{k'}}$$

Grading: For full points it must be apparent from the answer that the students starts from the Bayes rule and ends up with the correct equation for the requested posterior probability density, i.e., just writing the final equation is not sufficient for full points.

Task b

We want to solve $\hat{k} = \arg\max_k P(Y = k \mid X = x)P(Y = k)/P(x)$. Because the denominator $P(X = x)$ does not depend on the value of Y it is enough to find the class with the largest numerator $P(X = x \mid Y = k)P(Y = k)$. Also, because logarithm is a monotone function the maximum of a function and its logarithm is at the same location and therefore can equivalently solve for

$$\hat{k} = \arg\max_k \left[\log P(Y = k \mid X = x)P(Y = k)\right] = \arg\max_k \left[\log P(Y = k \mid X = x) + \log P(Y = k)\right]$$

By using the formulas from task a and taking the logarithms we can write this as

$$\hat{k} = \arg\max_k \left[-\log\left(2\pi\sigma_k^2\right) - \frac{1}{2\sigma_k^2}(x-\mu_k)^2 + \log \pi_k\right]$$

Expanding the squared term:

$$\hat{k} = \arg\max_k \left[-x^2/(2\sigma_k^2) + x\mu_k/\sigma_k^2 - \log\left(2\pi\sigma_k^2\right) - \mu_k^2/(2\sigma_k^2) + \log \pi_k\right]$$

We notice that when σ_k^2 differ there is a term that is quadratic in x. Because here we assume that all variances are equal, or $\sigma^2 = \sigma_k^2$, we can however write:

$$\hat{k} = \arg\max_k \left[-x^2/(2\sigma^2) + x\mu_k/\sigma^2 - \log(2\pi\sigma^2) - \mu_k^2/(2\sigma^2) + \log \pi_k\right]$$

The terms $-x^2/(2\sigma^2)$ and $\log(2\pi\sigma^2)$ that do not depend on k have no effect on the maximal k and hence they can be dropped, resulting to the requested equation:

$$\hat{k} = \arg\max_k \left(-\mu_k^2/(2\sigma^2) + x\mu_k/\sigma^2 + \log \pi_k\right).$$

Grading

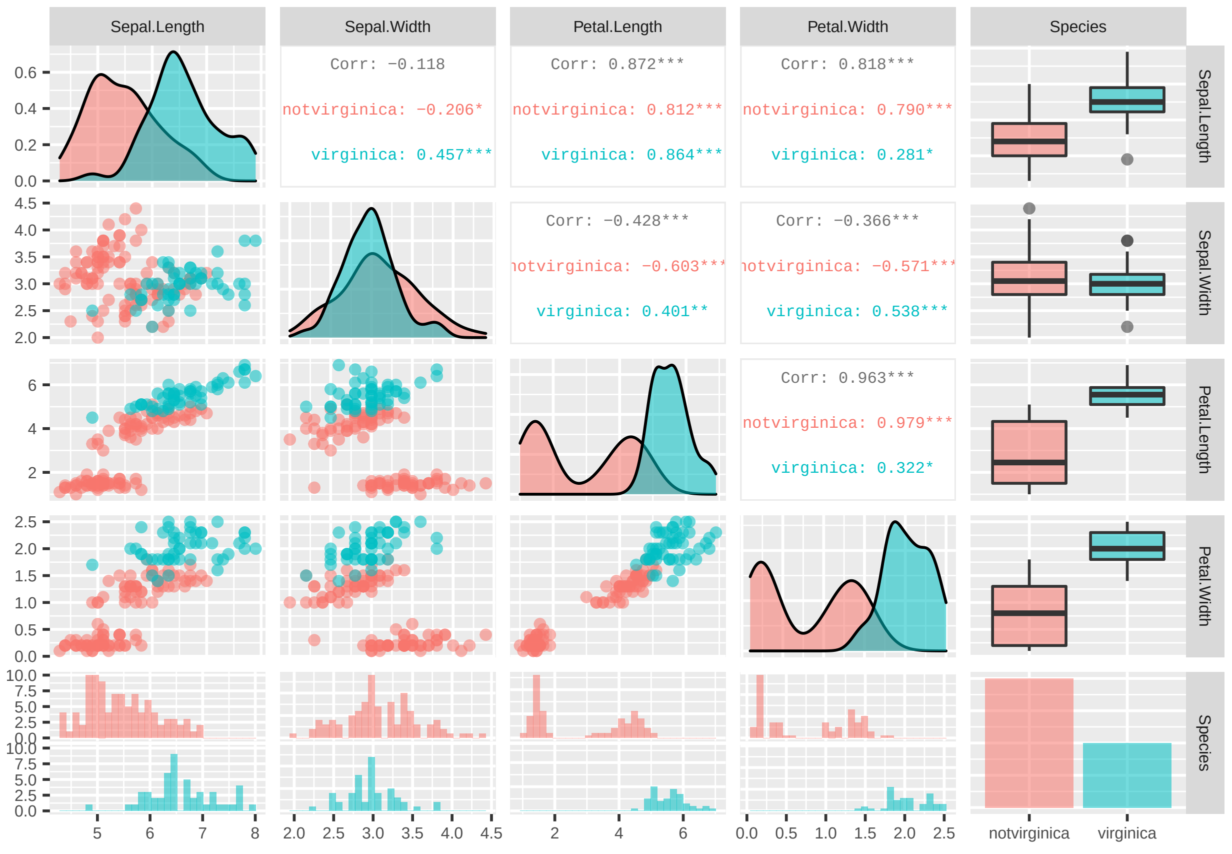
Points: max. 15 (a: 7, b: 8)

Problem 10

Task a

We will first load the dataset as instructed:

```r
set.seed(42)
library(GGally)
## Create new data with Species in "virginica" or "notvirginica"
data2 <- iris
data2$Species <- factor("virginica",levels=c("notvirginica","virginica"))
data2$Species[iris$Species!="virginica"] <- "notvirginica"
ggpairs(data2,aes(color=Species,alpha=0.4),
        upper=list(continuous=wrap("cor", size=2)))+
  theme(text=element_text(size=7))
```



We split the dataset in random into training and test sets of equal sizes using *stratified sampling*: first we sample into training set 25 items in random from class `virginica` and then 50 items from class `notvirginica`. The remaining 75 items (25 from `virginica` and 50 from `notvirginica`) form the test set. (Stratified sampling helps to avoid extra variance due to class imbalances in the training set, which may be issue especially for small data sets.)

```
stratified <- function(x,p=0.5) {
  sample(unlist(tapply(X=1:length(x),
                       INDEX=as.factor(x),
                       FUN=function(y) sample(y,size=floor(p*length(y))))))
}
i.tr <- stratified(data2$Species)
i.te <- setdiff(1:nrow(data2),i.tr)
data_tr <- data2[i.tr,] # training set
data_te <- data2[i.te,] # test set

summary(data_tr)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.400   Min.   :2.000   Min.   :1.200   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.700   1st Qu.:1.500   1st Qu.:0.200
##  Median :5.700   Median :3.000   Median :4.100   Median :1.200
##  Mean   :5.756   Mean   :3.047   Mean   :3.599   Mean   :1.131
##  3rd Qu.:6.300   3rd Qu.:3.400   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.200   Max.   :6.700   Max.   :2.500
##          Species
##  notvirginica:50
##  virginica   :25
##
##
##
##
```

The resulting dataset should have exactly 50 items of class `notvirginica` and 25 items of class `virginica`.

Task b

Means for class `notvirginica`:

```
mu0 <- apply(data_tr[data_tr$Species=="notvirginica",1:4],2,mean)
mu0
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##        5.372        3.102        2.656        0.696
```

Standard deviations for class `notvirginica`:

```
sd0 <- apply(data_tr[data_tr$Species=="notvirginica",1:4],2,sd)
sd0
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.5318585    0.5020326    1.3894882    0.5443438
```

Means for class `virginica`:

```
mu1 <- apply(data_tr[data_tr$Species=="virginica",1:4],2,mean)
mu1
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##        6.524        2.936        5.484        2.000
```

Standard deviations for class `virginica`:

```
sd1 <- apply(data_tr[data_tr$Species=="virginica",1:4],2,sd)
sd1
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width 
##    0.5932116    0.3828403    0.5112729    0.2677063
```

We will use μ_{ji} and σ_{ji} to denote the mean and standard deviation of variable $i \in \{1,2,3,4\}$ in class $j \in \{0,1\}$, respectively. We use "j=1" to denote class `virginica` and "j=0" class `notvirginica`.

We estimate the class probabilities, using Laplace smoothing *with pseudocount* of 1 on the training set $\pi_0 = \hat{P}(Y=0)$, first for class `notvirginica`…

```
pi0 <- (1+sum(data_tr$Species=="notvirginica"))/(2+nrow(data_tr))
pi0
```

```
## [1] 0.6623377
```

…and then for `virginica`, $\pi_1 = \hat{P}(Y=1)$:

```
pi1 <- (1+sum(data_tr$Species=="virginica"))/(2+nrow(data_tr))
pi1
```

```
## [1] 0.3376623
```

Notice that the Laplace smoothing "pulls" the means towards 0.5: π_0 (`pi0`) is little less than the fraction of 0s or 2/3 and π_1 (`pi1`) is little more than the fraction of 1s or 1/3.

Grading: To get full points from task b the values of the class probabilities need to be exactly 0.6623377 and 0.3376623, as above. The means and standard deviations may vary slightly from the ones reported above due to random sampling.

Task c

First, we write down the probability density function for the normal distribution with mean μ and variance of σ^2, $f(x \mid \mu, \sigma) = \exp\left(-(x-\mu)^2/(2\sigma^2)\right)/\sqrt{2\pi\sigma^2}$.

The requested probability $P(y=1 \mid x)$, where $x = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4$ is the 4-dimensional vector of morphological measurements, can be written as

$$\hat{P}(Y=1 \mid X=x) = \frac{\hat{P}(X=x \mid Y=1)\hat{P}(Y=1)}{\hat{P}(X=x \mid Y=0)\hat{P}(Y=0) + \hat{P}(X=x \mid Y=1)\hat{P}(Y=1)}.$$

The Naive Bayes (NB) assumption means that we can express the class-specific likelihood $\hat{P}(x \mid y=1)$ as a product of distributions over (in this case) the four (4) dimensions. Here we assume that the distributions are Gaussian and can therefore write $\hat{P}(X=x \mid Y=y) = \prod_{i=1}^4 f(x, \mu_{yi}, \sigma_{yi})$.

The requested probability can then be written as follows in the terms of numbers computed in task b above:

$$\hat{P}(Y=1 \mid X=x) = \frac{\pi_1 \prod_{i=1}^4 f(x_i \mid \mu_{1i}, \sigma_{1i})}{\sum_{y=0}^1 \pi_y \prod_{i=1}^4 f(x_i \mid \mu_{yi}, \sigma_{yi})}$$

Grading: To get full points the student must present the above equation (or equivalent).

Task d

Next, we write the equations from task c in R. First the normal distribution:

```
f <- function(x,mu,sigma) exp(-(x-mu)^2/(2*sigma^2))/sqrt(2*pi*sigma^2)
```

Then the estimated probability of one class, as a function of 4-dimensional covariate vector (this is the equation from the task c above!):

```
phat <- function(x) pi1*prod(f(x,mu1,sd1))/(pi0*prod(f(x,mu0,sd0))+pi1*prod(f(x,mu1,sd1)))
```

We compute probabilities $\hat{P}(Y = 1 \mid X = x)$ for all items in the test set and predict 1 (`virginica`) if this probability is at least 0.5, and 0 (`notvirginica`) otherwise.

```
acc <- mean(ifelse(apply(data_te[,1:4],1,phat)>=0.5,"virginica","notvirginica")==data_te$Species)
acc
```

```
## [1] 0.8266667
```

The classification accuracy turns out to be 0.8266667, which is rather good.

Grading: For full points the equation derived in task c should be used. The accuracy should be roughly the same as reported above. There can however be small variation due to random sampling.

Task e

We then repeat task d with logistic regression.

```
## First train a logistic regression model m.
m <- glm(Species ~ .,data_tr,family=binomial)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Summary of results:
summary(m)
```

```
##
## Call:
## glm(formula = Species ~ ., family = binomial, data = data_tr)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -6.535e-04  -2.000e-08  -2.000e-08   2.000e-08   4.876e-04
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -7184.9   396889.7  -0.018    0.986
## Sepal.Length    669.3    37647.8   0.018    0.986
## Sepal.Width    -424.5    23254.1  -0.018    0.985
## Petal.Length    404.1    22222.8   0.018    0.985
## Petal.Width    1399.5    76467.6   0.018    0.985
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9.5477e+01  on 74  degrees of freedom
## Residual deviance: 1.0060e-06  on 70  degrees of freedom
## AIC: 10
##
## Number of Fisher Scoring iterations: 25
```

```r
## "link" (the default output of predict, see R help page: ?predict.glm)
## is the linear response (the term \beta^Tx). The predicted
## probability would be sigmoid of this response or \sigma(\beta^Tx).
link_te <- predict(m,data_te)
link_tr <- predict(m,data_tr)

## Predictions (probability of 0.5 is at link==0):
yhat <- ifelse(link_te>=0,"virginica","notvirginica")
```

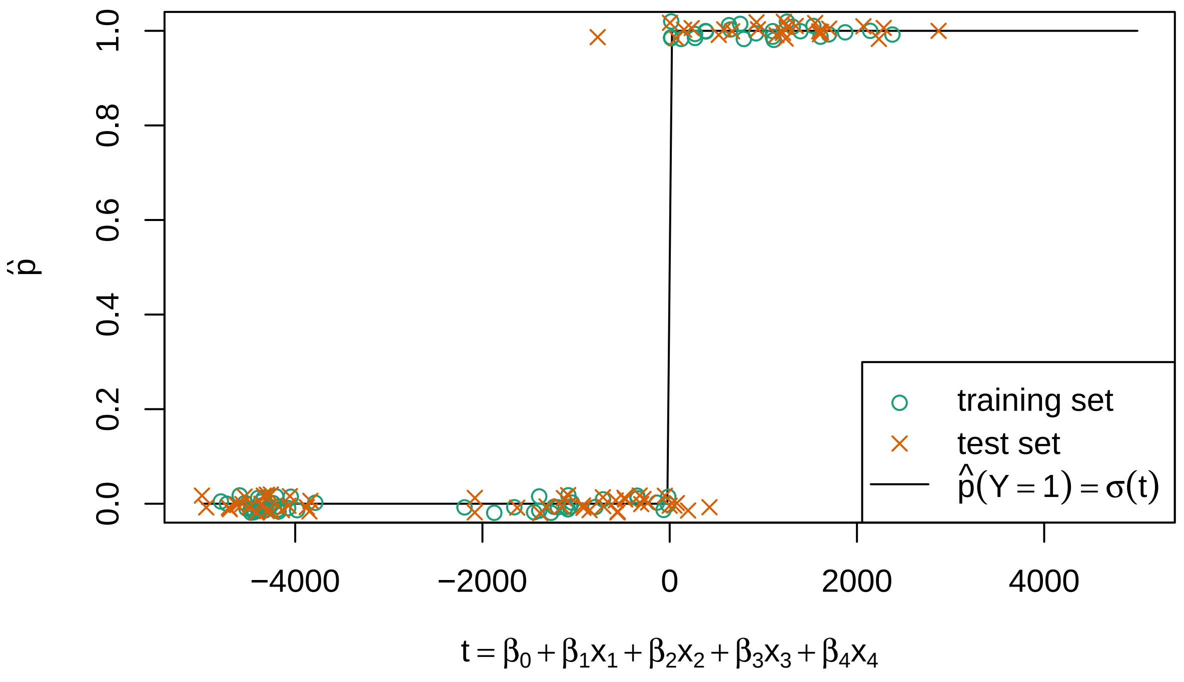
The classification accuracy can be computed as follows;

```r
acc <- mean(yhat==data_te$Species)
acc
```

```
## [1] 0.92
```

The classification accuracy of the logistic regression is 0.92.

```r
## Color-blind safe palette from https://colorbrewer2.org/#type=qualitative&scheme=Dark2&n=3
cols <- c("#1b9e77","#d95f02","#7570b3")
xlim <- max(abs(c(link_tr,link_te)))
plot(c(-xlim,xlim),c(0,1),type="n",
     xlab=expression(t==beta[0]+beta[1]*x[1]+beta[2]*x[2]+beta[3]*x[3]+beta[4]*x[4]),
     ylab=expression(hat(p)))
## Add little jitter to the plots to make them visually better looking
x <- seq(from=-xlim,to=xlim,length.out=200)
lines(x,1/(1+exp(-x)))
points(link_tr,jitter(1*(data_tr$Species=="virginica"),amount=0.02),pch=1,col=cols[1])
points(link_te,jitter(1*(data_te$Species=="virginica"),amount=0.02),pch=4,col=cols[2])
legend("bottomright",c("training set","test set",expression(hat(p)(Y==1)==sigma(t))),
       pch=c(1,4,NA),lty=c(NA,NA,"solid"),col=c(cols[1:2],"black"))
```



The logistic regression gave a warning because there is a hyper-plane that separates the classes in training data without an error. This causes the values of the coefficients β to go to infinity, which means that the sigmoid function $\sigma(t)$ in the above plot looks like a step function.

Notice that the hyperplane does not separate the test set perfectly, which is why the classification accuracy on the test set is smaller than 100% even though accuracy on the training set is 100%.

Grading: To get full points the student should report accuracy which should be around 0.9 and produce figure or figures similar to one shown above showing both the training and test set data. There can be slight variation due to random sampling and how the used logistic regression library behaves when the β diverges. It is possible that due to random sampling of the training data there is no separating hyperplane, in which case the accuracy on training data would be less than 100% and the logistic regression library would not give any warning. It does not matter if in the figure 0 and 1 are flipped as long as the student understands to flip them the right way (e.g., accuracy of around 0.1 means that the labels have probably been flipped!).

Grading

Points: max. 15 (a: 3, b: 3, c: 3, d: 3, e: 3)

Problem 11

Task a

The authors claim that although discriminative classifiers are traditionally considered superior compared to generative classifiers because of their lower asymptotic error (the error rate of the classifier as the sample size grows to infinity), generative classifiers converge faster to their asymptotic error rate, and thus may have a higher accuracy on small sample sizes.

Task b

Given class labels y and predictors $\mathbf{x} = (x_1, \ldots, x_p)$, the objective function that generative classifier h_{Gen} maximizes (with respect to parameter vector $\boldsymbol{\beta}$) is the joint likelihood $p(\mathbf{x}, y)$ (or equivalently its logarithm $\log p(\mathbf{x}, y)$), while discriminative classifiers h_{Dis} maximize directly the conditional likelihood $p(y|\mathbf{x})$ (or equivalently its logarithm $\log p(y|\mathbf{x})$ or 0-1 loss.

Two models which the authors discuss are the case of continuous predictors, in which each $p(x_i|y)$ is normal distribution, and a discrete predictor case, in which each $p(x_i|y)$ is a Bernoulli distribution. In both of the models the predictors are assumed independent given the class variable. In the first case the generative-discriminative pair is normal discriminant analysis (authors seem to refer to LDA with a diagonal covariance matrix) and logistic regression, and in the second case the pair is Naive Bayes and logistic regression.

Task c

It seems that in most of the data sets the error rate of the generative classifiers (Naive Bayes and normal discriminant analysis) does indeed initially decrease faster than the error rate of the logistic regression as the sample size grows, but logistic regression has a smaller error rate with higher sample sizes. However, with the smaller data sets the logistic regression does not catch up generative classifiers, because the sample size cannot be grown high enough to reach its asymptotic error rate. As suggested in the introduction, although discriminative classifiers have better asymptotic performance, generative classifiers may outperform them on smaller sample sizes.

Grading

Points: max. 15 (a: 5, b: 5, c: 5)

To get full point the answers should address the questions asked in the problem sheet.

Problem 12

Here we consider a toy data with a binary class variable $y \in \{0,1\}$ and with two discrete features $x_1 \in \{0,1\}$ and $x_2 \in \{0,1,2\}$. The true distribution from which the data is sampled is such that $P(y=1) = 1 - P(y=0) = 0.6$ and the class-conditioned distributions for (x_1, x_2) are specified as follows:

$$P(x_1, x_2 \mid y=0) = \begin{cases} 0.2 & , \quad x_1 = 0 \wedge x_2 = 0 \\ 0.4 & , \quad x_1 = 0 \wedge x_2 = 1 \\ 0 & , \quad x_1 = 0 \wedge x_2 = 2 \\ 0.1 & , \quad x_1 = 1 \wedge x_2 = 0 \\ 0.2 & , \quad x_1 = 1 \wedge x_2 = 1 \\ 0.1 & , \quad x_1 = 1 \wedge x_2 = 2 \end{cases}$$

and

$$P(x_1, x_2 \mid y=1) = \begin{cases} 0.6 & , \quad x_1 = 0 \wedge x_2 = 0 \\ 0.1 & , \quad x_1 = 0 \wedge x_2 = 1 \\ 0.1 & , \quad x_1 = 0 \wedge x_2 = 2 \\ 0.1 & , \quad x_1 = 1 \wedge x_2 = 0 \\ 0.1 & , \quad x_1 = 1 \wedge x_2 = 1 \\ 0 & , \quad x_1 = 1 \wedge x_2 = 2 \end{cases}$$

Task a

The naive Bayes (NB) assumption is not valid for this data. For the NB assumption to be valid the attributes should be independent given the class. In this case we should have $p(x_1, x_2 \mid y=0) = p(x_1 \mid y=0)p(x_2 \mid y=0)$ and $p(x_1, x_2 \mid y=1) = p(x_1 \mid y=1)p(x_2 \mid y=1)$ for all $x_1 \in \{0,1\}$ and $x_2 \in \{0,1,2\}$, which is not satisfied.

Grading: The above is sufficient answer. Below, I show probabilities that would satisfy NB assumption, they are not however required for full points.

Here the marginal probabilities are: $p(x_1 = 0 \mid y = 0) = 0.2 + 0.4 + 0 = 0.6$, $p(x_1 = 0 \mid y = 0) = 0.1 + 0.2 + 0.1 = 0.4$, $p(x_2 = 0 \mid y = 0) = 0.2 + 0.1 = 0.3$, $p(x_2 = 1 \mid y = 0) = 0.4 + 0.2 = 0.6$, $p(x_2 = 2 \mid y = 0) = 0{+}0.1 = 0.1$, $p(x_1 = 0 \mid y = 1) = 0.6{+}0.1{+}0.1 = 0.8$, $p(x_1 = 1 \mid y = 1) = 0.1{+}0.1{+}0 = 0.2$, $p(x_2 = 0 \mid y = 1) = 0.6 + 0.1 = 0.7$, $p(x_2 = 1 \mid y = 1) = 0.1 + 0.1 = 0.2$, and $p(x_2 = 2 \mid y = 1) = 0.1 + 0 = 0.1$. The following probabilities (with these marginal probabilities!) would obey the NB assumption:

$$P_{NB}(x_1, x_2 \mid y=0) = \begin{cases} 0.6 \times 0.3 = 0.18 & , \quad x_1 = 0 \wedge x_2 = 0 \\ 0.6 \times 0.6 = 0.36 & , \quad x_1 = 0 \wedge x_2 = 1 \\ 0.6 \times 0.1 = 0.06 & , \quad x_1 = 0 \wedge x_2 = 2 \\ 0.4 \times 0.3 = 0.12 & , \quad x_1 = 1 \wedge x_2 = 0 \\ 0.4 \times 0.6 = 0.24 & , \quad x_1 = 1 \wedge x_2 = 1 \\ 0.4 \times 0.1 = 0.04 & , \quad x_1 = 1 \wedge x_2 = 2 \end{cases}$$

and

$$P_{NB}(x_1, x_2 \mid y=1) = \begin{cases} 0.8 \times 0.7 = 0.56 & , \quad x_1 = 0 \wedge x_2 = 0 \\ 0.8 \times 0.2 = 0.16 & , \quad x_1 = 0 \wedge x_2 = 1 \\ 0.8 \times 0.1 = 0.08 & , \quad x_1 = 0 \wedge x_2 = 2 \\ 0.2 \times 0.7 = 0.14 & , \quad x_1 = 1 \wedge x_2 = 0 \\ 0.2 \times 0.2 = 0.04 & , \quad x_1 = 1 \wedge x_2 = 1 \\ 0.2 \times 0.1 = 0.02 & , \quad x_1 = 1 \wedge x_2 = 2 \end{cases}$$

Task b

We will first generate a test data set of 10000 points and 10 training data sets of sizes $n \in \{2^4, 2^5, \ldots, 2^{13}\}$, respectively.

```r
set.seed(42)
makedata <- function(n) {
  a <- data.frame(y =factor(c(0,0,0,0,0,0,1,1,1,1,1,1)),
                  x1=factor(c(0,0,0,1,1,1,0,0,0,1,1,1)),
                  x2=factor(c(0,1,2,0,1,2,0,1,2,0,1,2)))
  p <- c(0.4*c(0.2,0.4,0.0,0.1,0.2,0.1),
         0.6*c(0.6,0.1,0.1,0.1,0.1,0.0))
  a[sample.int(12,n,replace=TRUE,prob=p),,drop=FALSE]
}
data_test <- makedata(10000) # test data set
data <- lapply(2^(4:13),makedata) # training data sets
```

We make a function `phat` that takes the training and testing data as input and outputs the the probabilities $p(y = 1 \mid x_1, x_2)$ and then computes accuracy and perplexity by using theses probabilities.

```
library(e1071)
## estimate phat. The idea is to make a function that outputs the predicted
## probabilities and the parameters can be modified "easily" for different
## models.
phat <- function(data.tr, # training data
                 data.te=data_test, # test data - here data_test
                 form=y ~ x1+x2, # formula
                 model=function(f,d) naiveBayes(f,d,laplace=1), # model family
                 m=model(form,data.tr), # model trained on data.tr
                 pred=function(m,x) predict(m,x,type="raw")[,2]) { # function to make the predictions b
  pred(m,data.te)
}

## accuracy
acc <- function(p,y=data_test$y) mean(ifelse(p>=0.5,1,0)==y)

## perplexity
perp <- function(p,y=data_test$y) exp(-mean(log(ifelse(y==1,p,1-p))))
```

Then we collect the requested numbers for NB into dataframe `res`:

```
# collect results to data frames
res <-  data.frame(n=sapply(data,nrow))

phat_nb <- lapply(data,phat)
res$nb_acc  <- sapply(phat_nb,acc)
res$nb_perp <- sapply(phat_nb,perp)
```

Probabilities for logistic regression:

```
phat_glm <- lapply(data,function(d) {
  phat(d,
       model=function(f,d) glm(f,d,family=binomial),
       pred=function(m,x) predict(m,x,type="response"))
})
res$glm_acc <- sapply(phat_glm,acc)
res$glm_perp <- sapply(phat_glm,perp)
```

Logistic regression with interaction:

```
phat_glmx <- lapply(data,function(d) {
  phat(d,
       form=y ~ x1*x2,
       model=function(f,d) glm(f,d,family=binomial),
       pred=function(m,x) predict(m,x,type="response"))
})
res$glmx_acc  <- sapply(phat_glmx,acc)
res$glmx_perp <- sapply(phat_glmx,perp)
```

SVM (rbf with default parameters):

```
phat_svm <- lapply(data,function(d) {
  phat(d,
       model=function(...) svm(...,probability=TRUE),
       pred=function(m,x) attr(predict(m,x,probability=TRUE),"probabilities")[,"1"])
})
res$svm_acc  <- sapply(phat_svm,acc)
res$svm_perp <- sapply(phat_svm,perp)
```

The Bayes classifier uses the true probability $p(y = 1 \mid x) = p(x, y = 1)/(p(x, y = 0) + p(x, y = 1))$ which we can compute, because we know the generating distribution. We first compute $p(x, y = 0)$ and $p(x, y = 1))$ for all x and then use $p(y = 1 \mid x) = p(x, y = 1)/(p(x, y = 0) + p(x, y = 1))$ to compute the needed probabilities.

```
bayesc <- function(data=data_test) {
  p0 <- 0.4*c(0.2,0.4,0.0,0.1,0.2,0.1)  # p(x,y=0)
  p1 <- 0.6*c(0.6,0.1,0.1,0.1,0.1,0.0)  # p(x,y=1)
  a <- data.frame(x1=factor(c(0,0,0,1,1,1)),
                  x2=factor(c(0,1,2,0,1,2)),
                  p=p1/(p0+p1))
  apply(data[,c("x1","x2")],1,function(x) a[a$x1==x[1] & a$x2==x[2],"p"])
}
phat_bayes <- bayesc()
res$bayes_acc <- acc(phat_bayes)
res$bayes_perp <- perp(phat_bayes)
```

The dummy model always predics $\hat{p} = 0.6$:

```
phat_dummy <- rep(0.6,nrow(data_test))
res$dummy_acc <- acc(phat_dummy)
res$dummy_perp <- perp(phat_dummy)
```

We will then collect the results into tables:

```
knitr::kable(res[c("n","nb_acc","glm_acc","glmx_acc","svm_acc","bayes_acc","dummy_acc")],
             "simple")
```

n	nb_acc	glm_acc	glmx_acc	svm_acc	bayes_acc	dummy_acc
16	0.4152	0.4152	0.4152	0.3040	0.7591	0.6
32	0.6184	0.7144	0.7407	0.6000	0.7591	0.6
64	0.6631	0.6631	0.7591	0.6631	0.7591	0.6
128	0.7144	0.7144	0.7591	0.7591	0.7591	0.6
256	0.7144	0.7144	0.7591	0.7591	0.7591	0.6
512	0.7144	0.7144	0.7367	0.7367	0.7591	0.6
1024	0.7591	0.7591	0.7591	0.7591	0.7591	0.6
2048	0.7591	0.7144	0.7591	0.7591	0.7591	0.6
4096	0.7591	0.7144	0.7591	0.7591	0.7591	0.6
8192	0.7591	0.7591	0.7591	0.7591	0.7591	0.6

```r
knitr::kable(res[c("n","nb_perp","glm_perp","glmx_perp","svm_perp","bayes_perp","dummy_perp")],
             "simple")
```

n	nb_perp	glm_perp	glmx_perp	svm_perp	bayes_perp	dummy_perp
16	2.046262	4.348715	4.348715	2.056649	1.651256	1.960132
32	1.862901	1.905383	3.204073	2.344053	1.651256	1.960132
64	1.796884	1.800804	1.677375	1.839301	1.651256	1.960132
128	1.779885	1.787347	1.662549	1.732863	1.651256	1.960132
256	1.795831	1.798557	1.667568	1.718206	1.651256	1.960132
512	1.797024	1.793216	1.658611	1.719848	1.651256	1.960132
1024	1.780974	1.776542	1.651964	1.725619	1.651256	1.960132
2048	1.777716	1.777272	1.651731	1.725524	1.651256	1.960132
4096	1.779998	1.776702	1.651795	1.725387	1.651256	1.960132
8192	1.778044	1.775646	1.651614	1.725243	1.651256	1.960132

Task c

Question: "Discuss your observations and what you can conclude. Which of the models above are probabilistic, discriminative, and generative? How do accuracy and perplexity (log-likelihood) compare? Is there a relation to the insights from Problem 11 above? Why does logistic regression with the interaction term perform so well for larger datasets? Does your dummy classifier ever outperform other classifiers, or do other classifiers ever outperform the Bayes classifier?"

The probabilistic discriminative models here are the two logistic regression models. The probabilistic generative models are represented by NB. SVM is not a probabilistic model so it is not really a discriminative model, even though it might be called that way by someone.

Accuracies. We notice that the accuracies of all models, except dummy, converge to the same value when the dataset is large. For small datasets there is however some variation. NB is the best for very small data and logistic regression without interaction is not very far behind. The best performance is by the logistic regression with the interaction term which reaches optimal accuracy already for a training data of size $n = 64$. For large datasets all models report the same accuracy with the exception of the dummy model. The dummy model predicts always majority class and gives therefore accuracy of 0.6, which is the fraction of the majority class in the test set.

Perplexities. The perplexity (or log-likelihood) behaves a bit differently. The logistic regression with the interaction term is the only one to reach the perplexity of the optimal Bayes classifier for large datasets. This is because the the logistic regression is flexible enough with the interaction term to model the generating distribution exactly asymptotically, at the limit of infinitely large data. For smaller datasets the dummy model beats many of the other models in perlexity. For small datasets of size $n = 32$ or smaller NB beats other models (except of course the optimal Bayes classifier). The results are in line with Jordan et al. The generative model, here NB, outperforms the corresponding discriminative model, here the logistic regression without the interaction term, for datasets up to the size of $n = 256$.

We notice that dummy classifier beats many classifiers both in accuraxy and perplexity especially for very small datasets. No other classifier is able to beat the Bayes classifier either in accuracy nor in perplexity for any dataset size.

Grading:

To get full points the answer should address all questions asked.

Grading

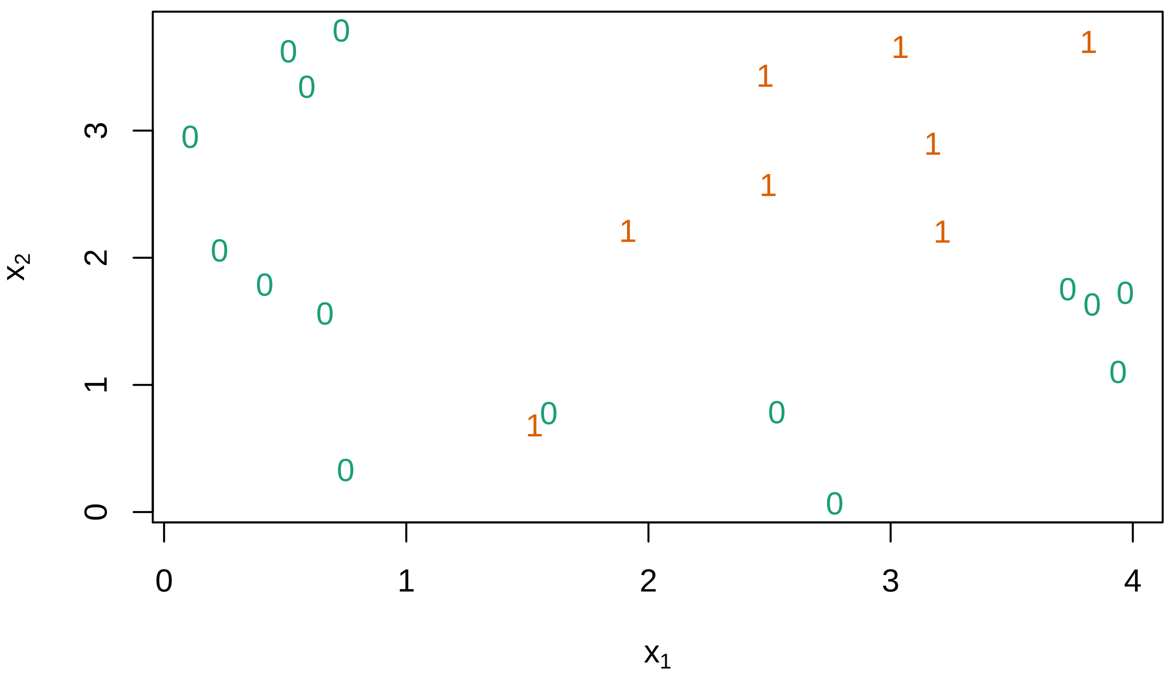
Points: max. 20 (a: 6, b: 7, c: 7)

Problem 13

[15% points]

Objectives: basic principles of decision trees

```r
set.seed(42)
rtoy <- data.frame(
  x1=c(runif(8,min=0,max=0.8),runif(15,min=1.2,max=4)),
  x2=c(runif(8,min=0,max=4),runif(7,min=2.2,max=4),runif(8,min=0,max=1.8)),
  class=factor(c(rep(0,8),rep(1,7),rep(0,8))))
rtoy[14:23,"class"][which.min(rtoy[14:23,"x1"])] <- 1
rtoyplot <- function() {
  plot(rtoy[,1:2],type="n",xlab=expression(x[1]),ylab=expression(x[2]))
  text(rtoy[,1:2],labels=rtoy[,3],col=cols[rtoy[,"class"]])
}
rtoyplot()
```



Task a

We'll choose here the *Gini index*, defined by Eq. (8.6) of James et al. As discussed in the lectures, for binary data the value of the gini index is given by $G(p) = 2p(1-p)$, where p is the fraction of ones in the split, derived as $G(p) = \sum_{c=0}^{1} \hat{p}_{mc}(1-\hat{p}_{mc}) = 2\hat{p}_{m1}(1-\hat{p}_{m1})$, where $\hat{p}_{mc}$ is the fraction of class c in node m and where we have used $\hat{p}_{m_0} = 1 - \hat{p}_{m1}$ for binary case.

```r
G <- function(p) 2*p*(1-p)
```

We will also define here auxiliary function to compute all splits (**not required for full points!**):

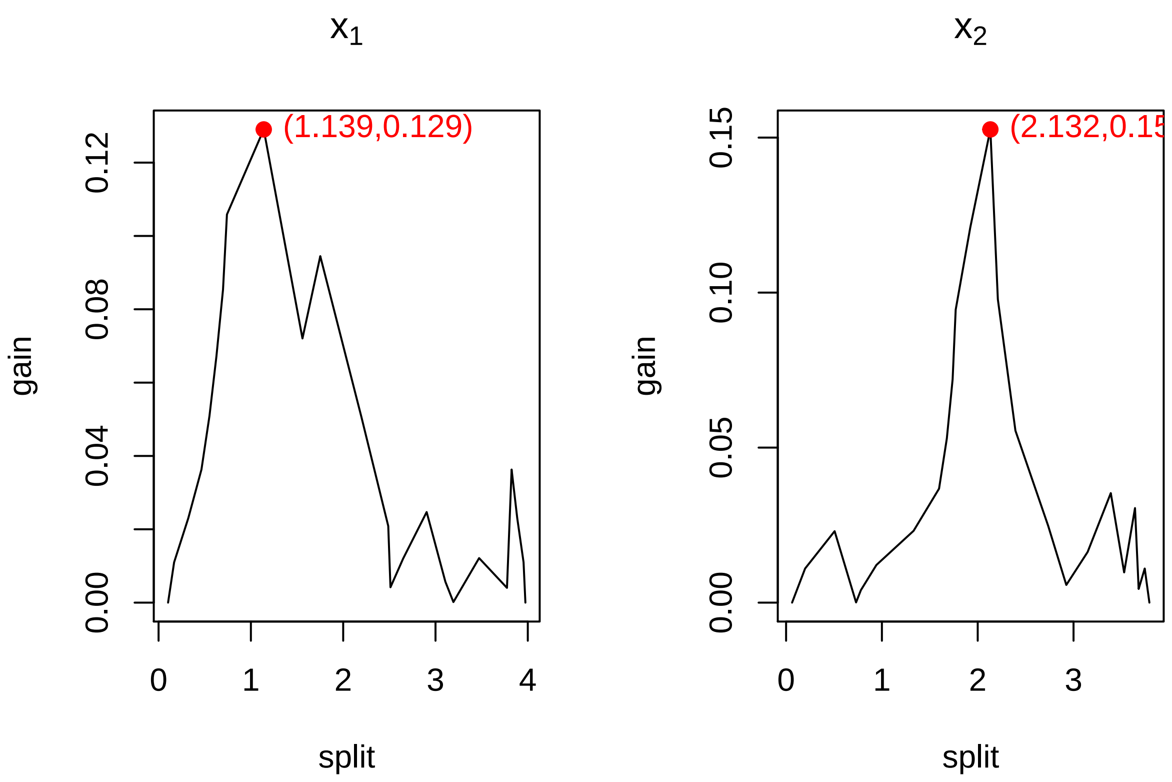
```r
ginigain <- function(class,x,g=function(r) 2*r*(1-r),delta=0.01) {
  ## First order the items and classes
  i <- order(x)
  x <- x[i]
  class <- class[i]
  ## The possible splits are between items
  splits <- (c(x,x[length(x)]+delta)+c(x[1]-delta,x))/2 # splits
  Q0 <- g(mean(class=="1")) # Original impurity
  data.frame(split=splits,
             gain=sapply(splits,function(s) {
               c1 <- class[x<s]
               c2 <- class[s<=x]
               Q0-
                 (if(length(c1)==0) 0 else (length(c1)/length(class))*g(mean(c1=="1")))-
                 (if(length(c2)==0) 0 else (length(c2)/length(class))*g(mean(c2=="1")))
             }))
}
```

A typical decision tree algorithm works recursively. For a rectangular subset of data, the algorithm goes through all possible splits for all axis and then picks a split with highest gain. The gain is computed as gain $= G(D_1 \cup D_2) - G(D_1) \times |D_1|/|D_1 \cup D_2| - G(D_2) \times |D_2|/|D_1 \cup D_2|$, where $D_1 \cup D_2$ are the datapoints in the initial subset and D_1 and D_2 are the dataset created by the split. The gain is denoted by $G()$, here it is the Gini index.

We look at the figure and notice that splits at $x_1 = 1.139$ or $x_2 = 2.132$ would result in quite pure splits. Lets compare the gains. If we split at $x_1 = 1.139$ we would have 8 zeros on the left side and 15 numbers out of which 8 are ones on the right hand side. Initially, there are 8 ones out of 23 data items. The gain would therefore be $G(8/23) - 8/23 \times G(0/8) - 15/23 \times G(8/15) = 0.1290485$. On the other hand, if we would split at $x_2 = 2.132$ then we would have 11 items above of which 7 are ones and below we would have 12 items of which 1 is 1. The gain would be $G(8/23) - 11/23 \times G(7/11) - 12/23 \times G(1/12) = 0.1526322$. The gain is larger for the split at $x_2 = 2.132$, where we will place our first split

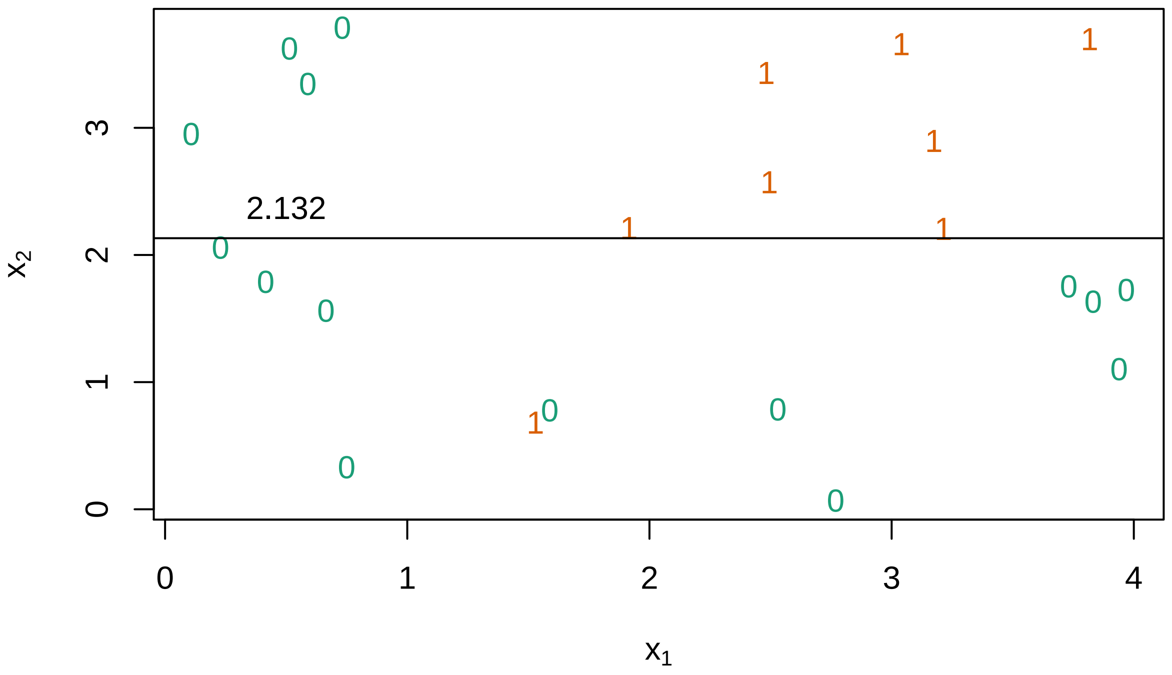
We plot possible splits along x_1 and x_2 axis (**going through all possible splits is not required for full points!**):

```
gain1 <- ginigain(rtoy$class,rtoy$x1)
gain2 <- ginigain(rtoy$class,rtoy$x2)
plotgain <- function(g,...) {
  plot(g$split,g$gain,type="l",xlab="split",ylab="gain",...)
  i <- which.max(g$gain)
  points(g$split[i],g$gain[i],col="red",pch=19)
  text(g$split[i],g$gain[i],label=sprintf("(%.3f,%.3f)",g$split[i],g$gain[i]),pos=4,col="red")
}
par(mfrow=c(1,2))
plotgain(gain1,main=expression(x[1]))
plotgain(gain2,main=expression(x[2]))
```



We get the highest gain if we split x_2 at $x_2 = 2.132$ with the gain of 0.153.

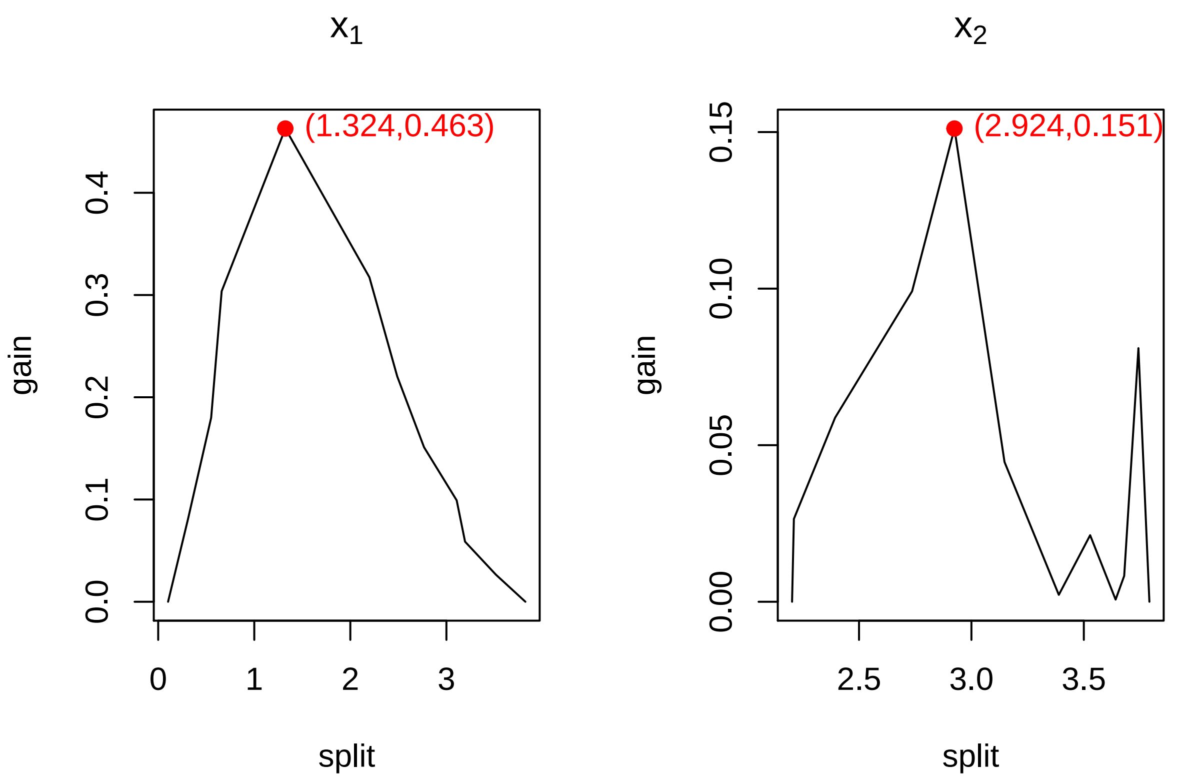
```
rtoyplot()
split1 <- gain2$split[which.max(gain2$gain)]
abline(h=split1)
text(0.5,split1,label=sprintf("%.3f",split1),pos=3)
```



Now the area is split into two. Then we will recurse into the subsets. Lets start from the top one at $x_2 \geq 2.132$. It is rather obvious that we get the best split by splitting with respect to $x_1 = 1.324$ so that we perfectly separate the classes. The upper part contains 11 points out of which 7 are ones. The gain is therefore $G(7/11) - 4/11 * G(0/4) - 7/11 * G(7/7) = 0.4628099$.

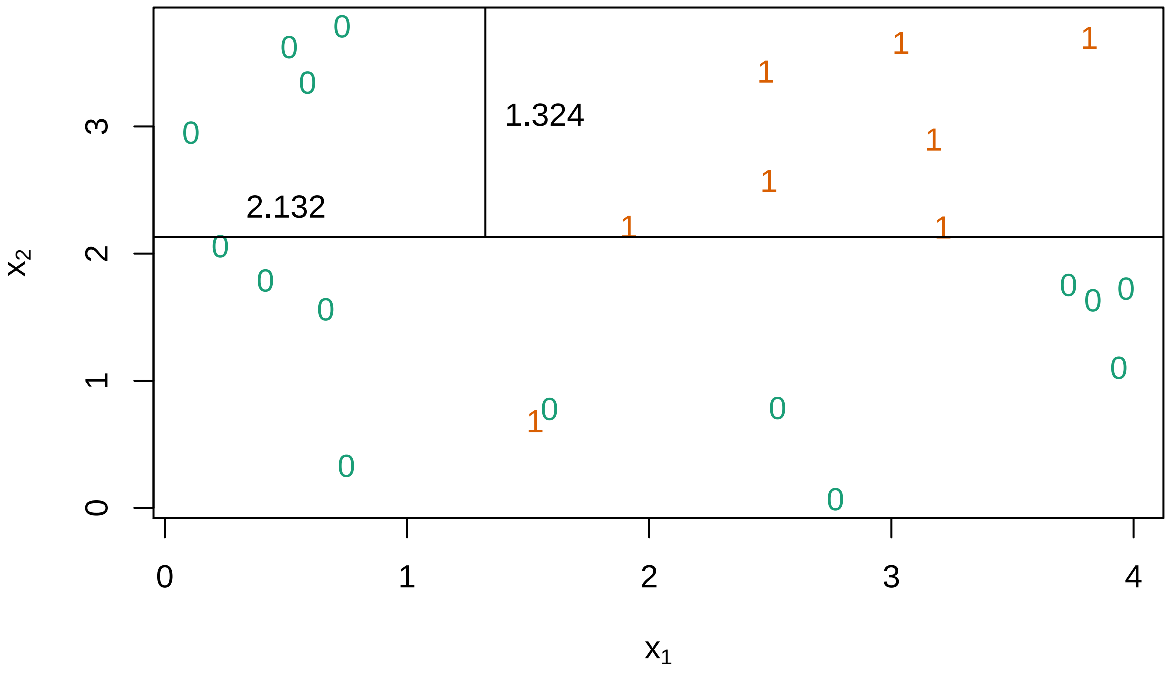
We can also check all possible splits (**going through all possible splits is not required for full points!**):

```
gain3 <- ginigain(rtoy$class[rtoy$x2>=split1],rtoy$x1[rtoy$x2>=split1])
gain3b <- ginigain(rtoy$class[rtoy$x2>=split1],rtoy$x2[rtoy$x2>=split1])
par(mfrow=c(1,2))
plotgain(gain3,main=expression(x[1]))
plotgain(gain3b,main=expression(x[2]))
```



Lets put the split to $x_1 = 1.324$, where the gain is 0.463.

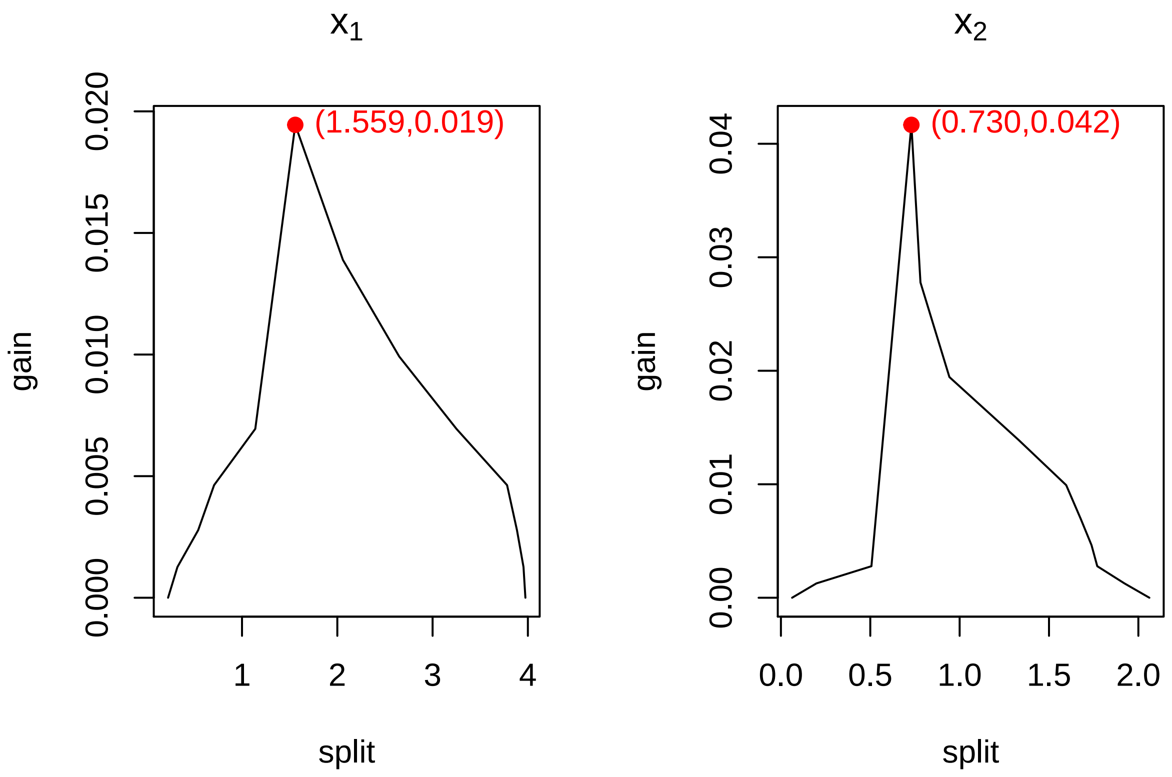
```
rtoyplot()
split2 <- gain3$split[which.max(gain3$gain)]
abline(h=split1)
text(0.5,split1,label=sprintf("%.3f",split1),pos=3)
lines(c(split2,split2),c(split1,4))
text(split2,(split1+4)/2,label=sprintf("%.3f",split2),pos=4)
```



We are done up there at $x_2 \geq 2.132$. Lets now look at the area $x_2 < 2.132$. There is only one 1 left and many ways to do the split. The area has 12 numbers out of which 1 is one. Lets first choose to split at $x_2 = 0.730$, just above the one. The gain is then $G(1/12) - 3/12 * G(1/3) - 9/12 * G(0/9) = 0.0416667$.

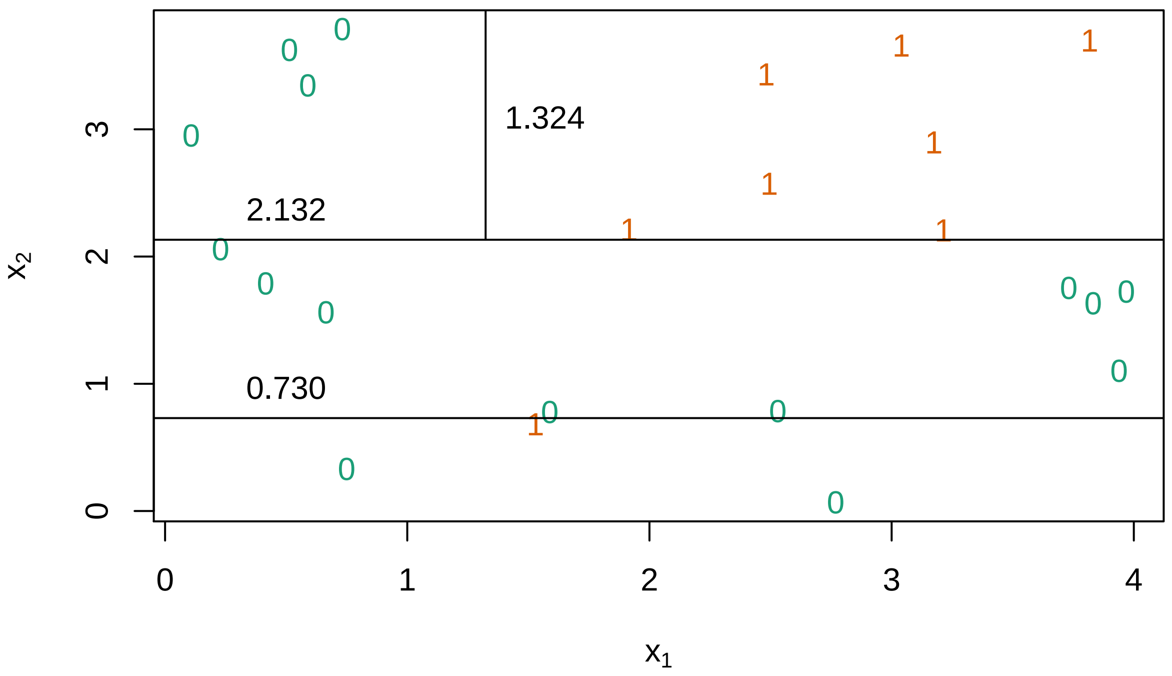
We can of course check all splits both to the direction of x_1 and x_2 (**going through all possible splits is not required for full points!**):

```
gain4 <- ginigain(rtoy$class[rtoy$x2<split1],rtoy$x1[rtoy$x2<split1])
gain5 <- ginigain(rtoy$class[rtoy$x2<split1],rtoy$x2[rtoy$x2<split1])
par(mfrow=c(1,2))
plotgain(gain4,main=expression(x[1]))
plotgain(gain5,main=expression(x[2]))
```



We get the highest gain of 0.042 by splitting at $x_2 = 0.730$.

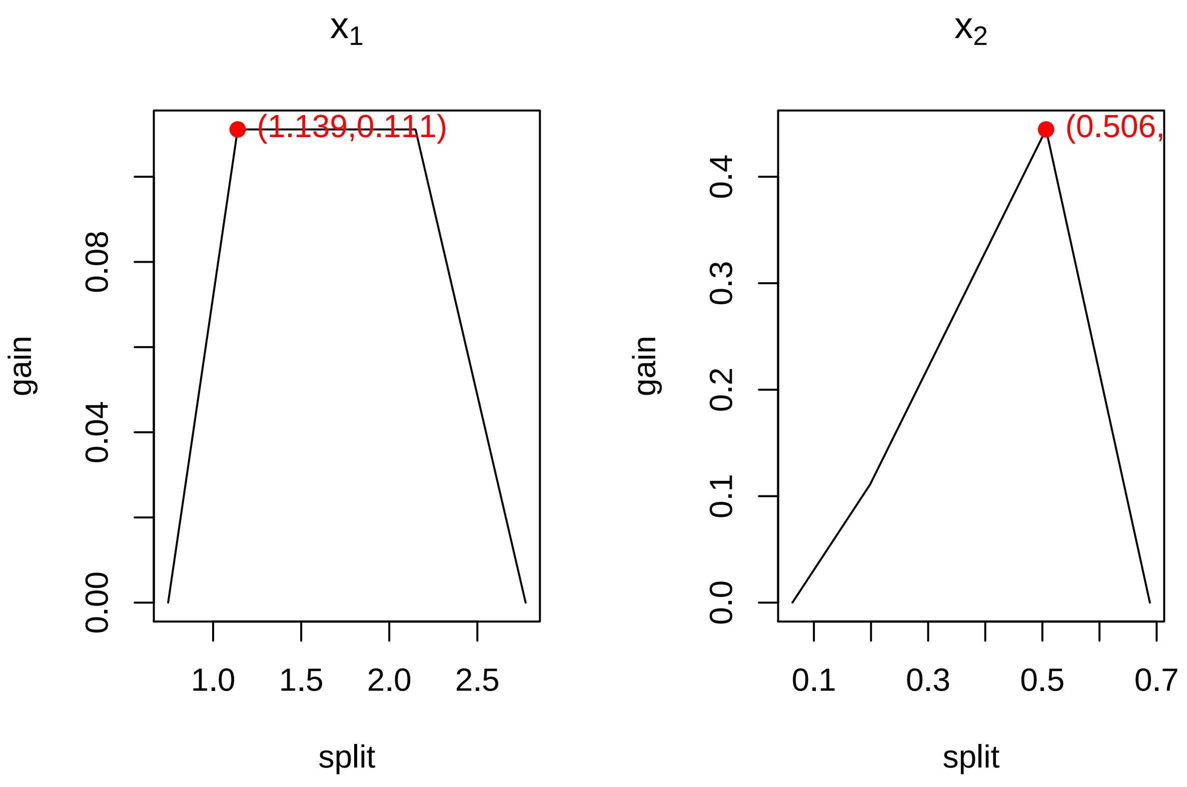
```
rtoyplot()
split3 <- gain5$split[which.max(gain5$gain)]
abline(h=split1)
text(0.5,split1,label=sprintf("%.3f",split1),pos=3)
lines(c(split2,split2),c(split1,4))
text(split2,(split1+4)/2,label=sprintf("%.3f",split2),pos=4)
abline(h=split3)
text(0.5,split3,label=sprintf("%.3f",split3),pos=3)
```



There is only one impure block left at $x_2 < 0.73$ which has 3 items of which 1 is one. Lets try a split at $x_2 = 0.560$ which is the only split resulting to pure blocks and which has the gain of $G(1/3) - 1/3 * G(1/1) - 2/3 * G(2/2) = 0.4444444$.

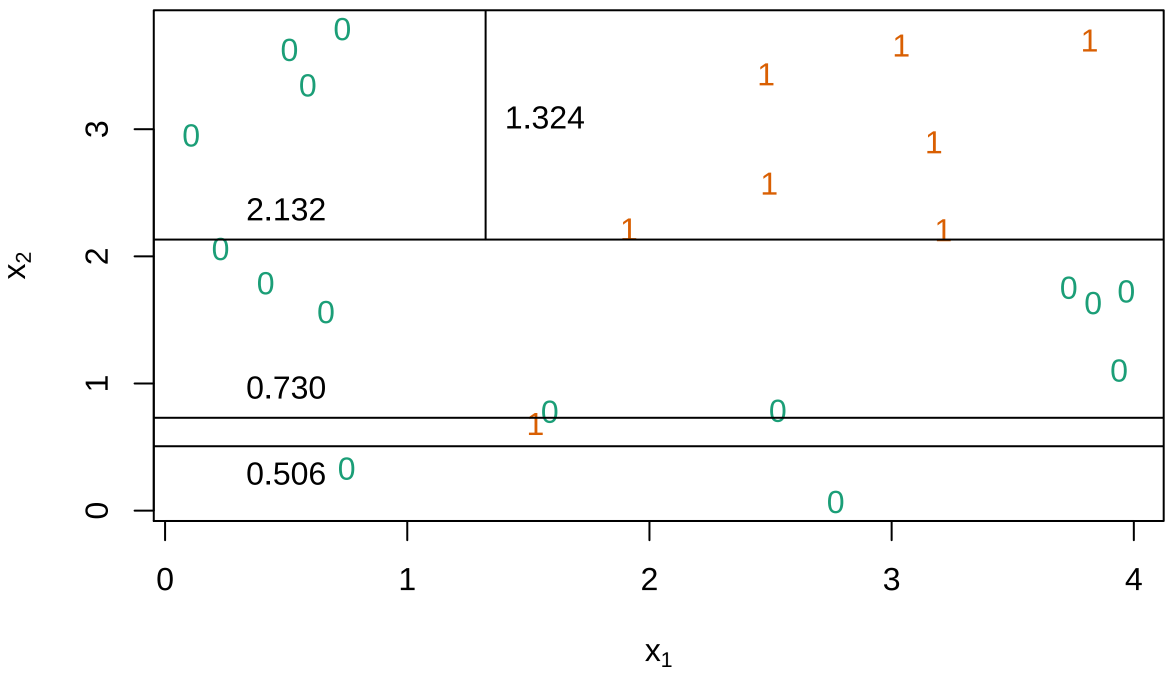
We can also do the search of all splits to both axis (**going through all possible splits is not required for full points!**), although in this case it is obvious that the above mentioned split is the best - it is the only one resulting to pure tree in one step:

```
gain6 <- ginigain(rtoy$class[rtoy$x2<split3],rtoy$x1[rtoy$x2<split3])
gain7 <- ginigain(rtoy$class[rtoy$x2<split3],rtoy$x2[rtoy$x2<split3])
par(mfrow=c(1,2))
plotgain(gain6,main=expression(x[1]))
plotgain(gain7,main=expression(x[2]))
```

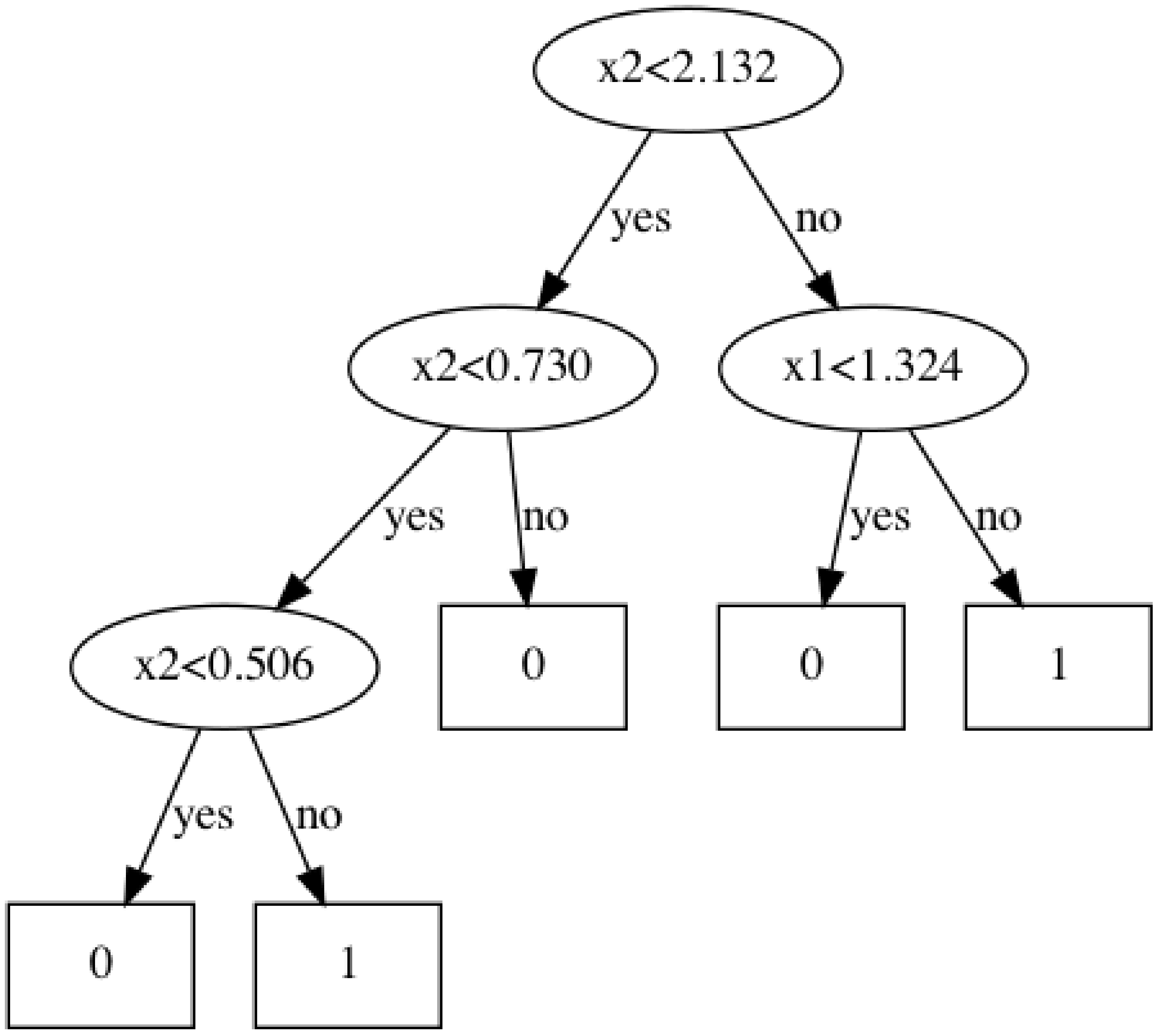


The best gain of 0.444 is obtained by splitting at $x_2 = 0.560$.

```r
rtoyplot()
split4 <- gain7$split[which.max(gain7$gain)]
abline(h=split1)
text(0.5,split1,label=sprintf("%.3f",split1),pos=3)
lines(c(split2,split2),c(split1,4))
text(split2,(split1+4)/2,label=sprintf("%.3f",split2),pos=4)
abline(h=split3)
text(0.5,split3,label=sprintf("%.3f",split3),pos=3)
abline(h=split4)
text(0.5,split4,label=sprintf("%.3f",split4),pos=1)
```

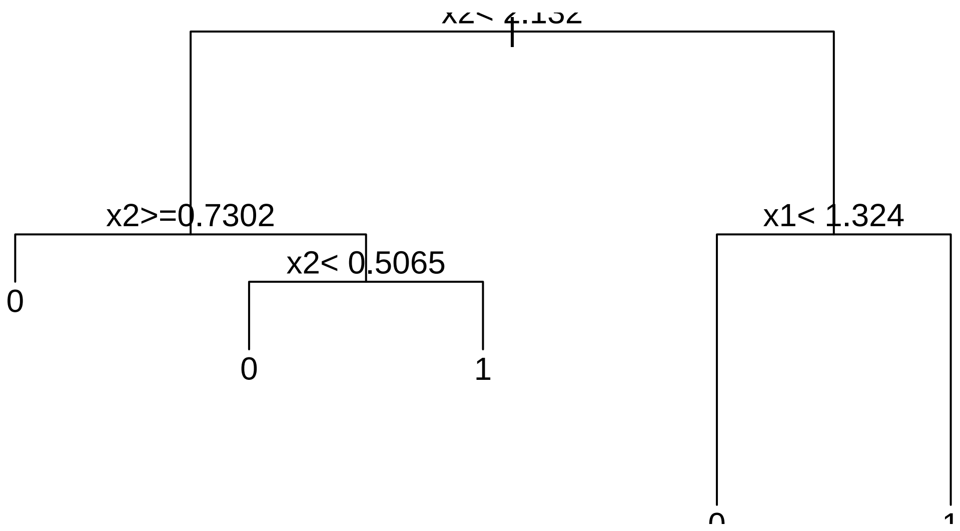


There are no impure blocks left: we are done. Let's draw the resulting decision tree.



Let's do the same with the `rpart` library and let the tree run to the bitter end (zero error on training data, **`rpart` is not required for full points!**).

```
library(rpart)
m <- rpart(class ~ .,rtoy,control=rpart.control(cp=0,minsplit=1,minbucket=1))
plot(m) ; text(m,pretty=0)
```



We get exactly the same tree! :)

Grading

Points: max. 15

The get full points it should be clear from the answer that the student understands how the decision tree algorithm works and the resulting solution has no errors on the training data. If some splits are at different locations it does not matter, as long as the locations "make sense". For example, it would have ok to have the first split at $x_1 \approx 1.139$ instead at $x_2 = 2.132$ - which would of course result in a different-looking tree. On the other hand, it would not be ok to have $x_1 \approx 0.5$ as the first split, because there is clearly a better split at $x_1 \approx 1.139$. Other requirements for full points: impurity measures of chosen splits reported and a figure of the resulting decision tree. It is not enough just to run library implementation of the decision tree algorithm (e.g., `rpart`) and report the result.

Problem 14

In this problem you consider applying *k-nearest neighbour* (k-NN) classifier to the training dataset $D = \{(x_i, c_i)\}_{i=1}^{14}$, where the covariates $x_i \in \mathbb{R}$ and the classes $c_i \in \{-1, +1\}$ are given in below.

i	x_i	c_i
1	0	$+1$
2	2	$+1$
3	3	$+1$
4	5	-1
5	6	$+1$
6	8	$+1$
7	9	$+1$
8	12	-1
9	13	-1
10	15	-1
11	16	$+1$
12	18	-1
13	19	-1
14	21	-1

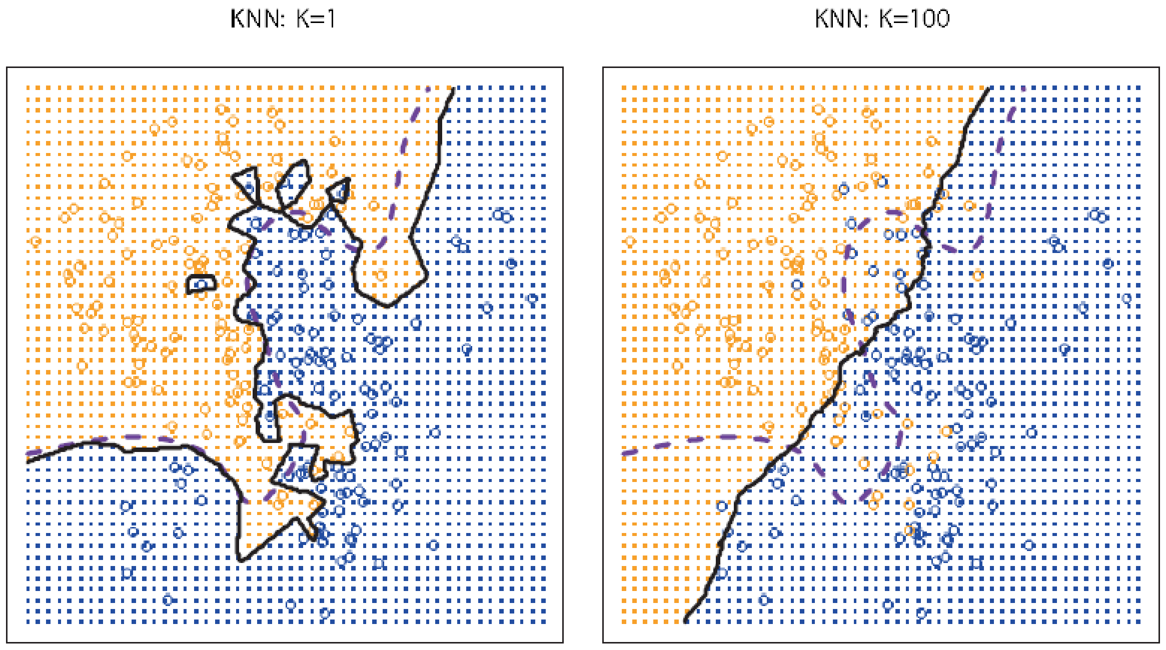
Task a

For $k = 1$ a new point is classified as $c = +1$ if $x \leq 4$, $5.5 \leq x \leq 10.5$, or $15.5 \leq x \leq 17$. Every training data point is classified corrrectly, i.e., the error on training data is zero.

For $k = 3$ a new point is classified as $c = +1$ if $x \leq 10.5$. Training data points $c_4 = -1$ and $c_1 1 = +1$ will be classified incorrectly, i.e., the classification error on training data is 2/14.

Task b

If k is small the model is flexible (complex-looking classification boundries, more prone to overfitting, less prone to underfitting), if k is large it is inflexible (smoother classification boundaries, less prone to over-fitting, more prone to under-fitting). At the extreme case of $k = n$ the classifier outputs constant prediction for any new data point.



Grading

Points: max. 15 (a: 7, b: 8)

Problem 15

Grading

Points: max. 5

Give full points if the answer contains some sentences that are about the topic of the question.

Machine Learning Guest Lectures

We will have two machine learning guest lectures on Friday **4 December 2020 at 10:15–12**. The tentative programme includes two 15–20 minute presentations, followed by a discussion where you can ask questions from the speakers.

The lectures will take place via the usual Zoom link at https://helsinki.zoom.us/j/68680705433?pwd=RTVTdWFSdS96S3h0NkU5S1lFRkdnZz09 (Meeting ID: 686 8070 5433, Password: 014569).

The speakers and the topics are:

Andreas Henelius: Data Science in Finance - Machine learning for overdue invoice prediction

Abstract: Invoice financing is a concept in banking enabling short-term borrowing by the bank's customers against the amounts due from the invoices held by the customer. Such borrowing can help the customer, e.g., with cash flow management. For this process to be feasible, it must be automated, and the bank needs to estimate the net present value of invoices, which depends on the expected payback time. In this presentation I talk about how to leverage machine learning to predict if invoices will be overdue.

About the speaker: D.Sc. (Tech.) Andreas Henelius is a data scientist at the OP Financial Group. Previously he worked at the University of Helsinki, Aalto University and the Finnish Institute of Occupational Health.

Antti Ukkonen: From voice to meaning: Machine learning for spoken language understanding

Abstract: I will discuss some of the many technical challenges we face at Speechly (www.speechly.com) when building a SaaS -solution using which developers with no prior ML or speech recognition experience can integrate voice functionality to their mobile or web applications. I will talk about deep learning, transfer learning, neural language models, perhaps a bit about reinforcement learning, and how to put these together in an autonomous cloud-based model training infrastructure.

About the speaker: Antti works currently as the Head of Natural Language Understanding at Speechly. In this role he is mainly in charge of developing and maintaining machine learning models that extract meaning from spoken language. Prior to joining Speechly in summer 2019, Antti spent almost 15 years in industry and academia as a data mining researcher. He has held positions at Yahoo! Research, Helsinki Institute for Information Technology HIIT, Finnish Institute for Occupational Health. Most recently, until December 2019, he was an Academy Research Fellow at University of Helsinki. Antti got his doctoral degree from Aalto University in 2008.

Welcome!

The lectures are part of the course DATA11002 Introduction to Machine Learning (access to the web site requires registration). Please see the course web site for possible updates.

Kai Puolamäki