Lab report – Lab 2

Group A4: Ingrid Wendin, Per Bark, and David Åström

Assignment 1

1. Firstly, the data is loaded from the iris data and the MASS package is loaded. The plot is then made:

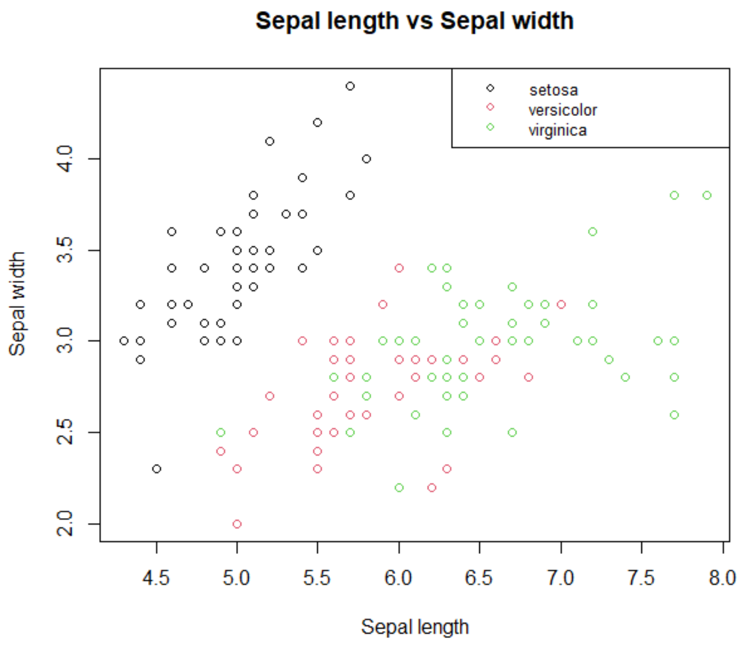


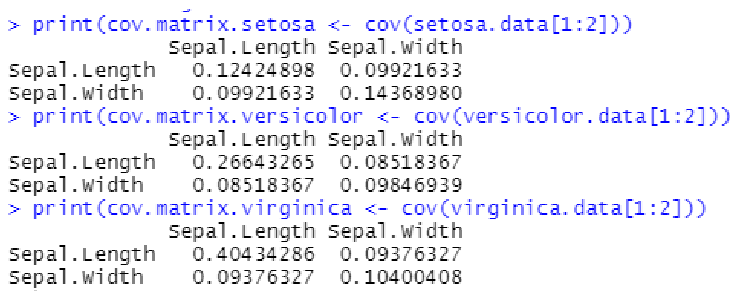
Figure 1: Scatterplot of sepal length and width, and species based on iris data

As can be seen from the plot that given plot, setosa is distinctly separately distributed from versicolor and virginica, whereas the versicolor and virginica are quite equally distributed based on sepal width and sepal length. So setosa should be easily classified with LDA, whereas misclassification is expected for versicolor and virginica as no linear expression can separate the species without errors.

2a. The data was separated and using the mean() and cov() function the means and covariance matrixes was calculated. The priors was calculated using division of nrow() of species by of total data. See table and figure below for computed values:

Table 1: Species mean values for sepal length and width and prior probability.

Species	Sepal length mean	Sepal width mean	Prio
Setosa	5.006	3.428	1/3
Versicolor	5.936	2.77	1/3
Virginica	6.588	2.974	1/3



```
> print(cov.matrix.setosa <- cov(setosa.data[1:2]))
             Sepal.Length Sepal.Width
Sepal.Length   0.12424898  0.09921633
Sepal.Width    0.09921633  0.14368980
> print(cov.matrix.versicolor <- cov(versicolor.data[1:2]))
             Sepal.Length Sepal.Width
Sepal.Length   0.26643265  0.08518367
Sepal.Width    0.08518367  0.09846939
> print(cov.matrix.virginica <- cov(virginica.data[1:2]))
             Sepal.Length Sepal.Width
Sepal.Length   0.40434286  0.09376327
Sepal.Width    0.09376327  0.10400408
```

Figure 2: Species covariance matrixes for sepal length and width

2b. The pooled covariance matrix is calculated from the following equation:

$$\hat{\Sigma} = \frac{1}{N}\sum_{c=1}^{k} N_c\hat{\Sigma}_c$$

And the resulting pooled matrix is the following:

Table 2: Pooled covariance matrix of sepal length and width for all species

	Sepal.Length	**Sepal.Width**
Sepal.Length	0.26500816	0.09272109
Sepal.Width	0.09272109	0.11538776

2c. Probabilistic model:

$$\boldsymbol{x}|y = C_i, \mu_i \sim N(\mu_i, \Sigma)$$

$$y|\pi \sim Multinomial(\pi_1, \dots, \pi_k)$$

2d. Using the equations below the following discriminant functions for the classes was calculated:

$$w_{0i} = -\frac{1}{2}\boldsymbol{\mu}_i^T\Sigma^{-1}\boldsymbol{\mu}_i + log\pi_i$$

$$w_i = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i$$

```
# delta.setosa = x%*%(11.81827, 20.21183) - 65.32281
# delta.versicolor = x%*%(19.475666, 8.356129) - 70.47563
# delta.virginica = x%*%(22.037717, 8.065317) - 85.68398
```

2e. Decision boundary is computed from the following equation by solving for x, and is presented below:

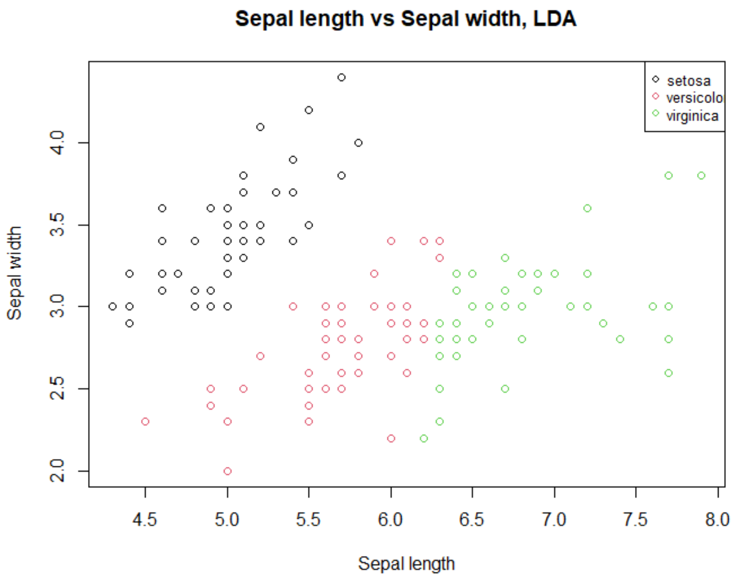
$$w_i\boldsymbol{x}^T + w_{0i} = w_j\boldsymbol{x}^T + w_{0j}$$

```
#DB.setver: 0 = x%*%(-7.657399, 11.855698) + 5.152822
#DB.setvir: 0 = x%*%(-10.21945, 12.14651) + 15.20835
#DB.vervir: 0 = x%*%(-2.5620509, 0.2908121) + 20.36117
```

Do estimated covariance matrices seem to fulfill LDA assumptions?

- The covariance matrixes are not equal, which is not according to LDA assumptions

3. The discriminant function is calculated as described above over all the data using a loop function, giving a discriminant value for every datapoint for each class. The discriminant function results in a 150x1 vector which can be combined with the other classes to create a combined 150x3 matrix with all the discriminant values. Whereas the class is determined by the value for each row which is the highest. The classification is done by looping through the discriminant matrix and assigning the class accordingly using the which.max() function for each row. By using the table() function the misclassification rate is then calculated by counting the miscalculations and then dividing by total number of data points. The misclassification rate is 20%, which is quite high. However, except for one setosa, there are only versicolor and virginica that are misclassified. See figure below for the scatterplot of the LDA classification. Both the LDA performed by the lda() function and the computed discriminant functions from 2) give the same results, which is expected since they use the same models. See the table below for the misclassification of respective function, and notice how they are the same.



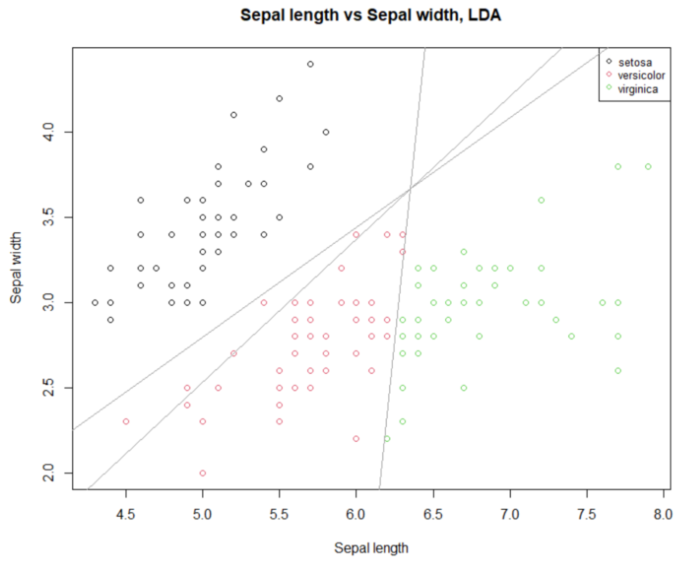


Figure 3: Scatterplot of LDA classification without(left) and with decision boundaries (right)

```
## delta.predicted setosa versicolor virginica
##      setosa         49          0         0
##      versicolor      1         36        15
##      virginica       0         14        35



##
##                setosa versicolor virginica
##   setosa          49          0         0
##   versicolor       1         36        15
##   virginica        0         14        35
```

Table 3: Misclassifications of the classification from 2 (above) and the LDA function (below)

4. The mvtnorm package is installed and then loaded into the script. Afterwards the samples are created through the sample() function using the three species and their priors. The sampling generates 62 setosa, 46 versicolor and 42 virginica. The result is presented in the scatterplot below. Comparing with the original data the distribution is quite similar, but the versicolor and virginica seems to be a bit more separated even though they are still overlapping quite much.

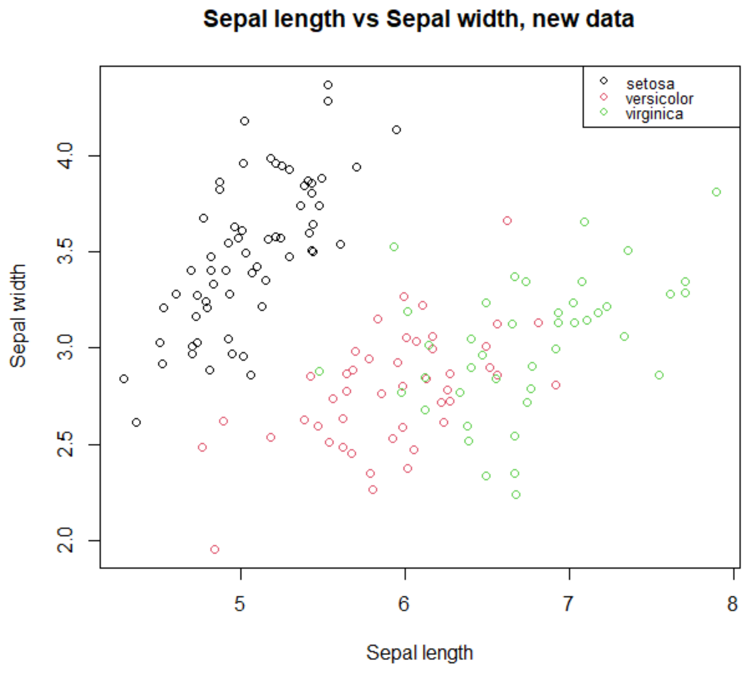


Figure 4: Scatterplot of new data

5. After loading the package the multinom() function is used on the original data. The plot of the result is presented below:

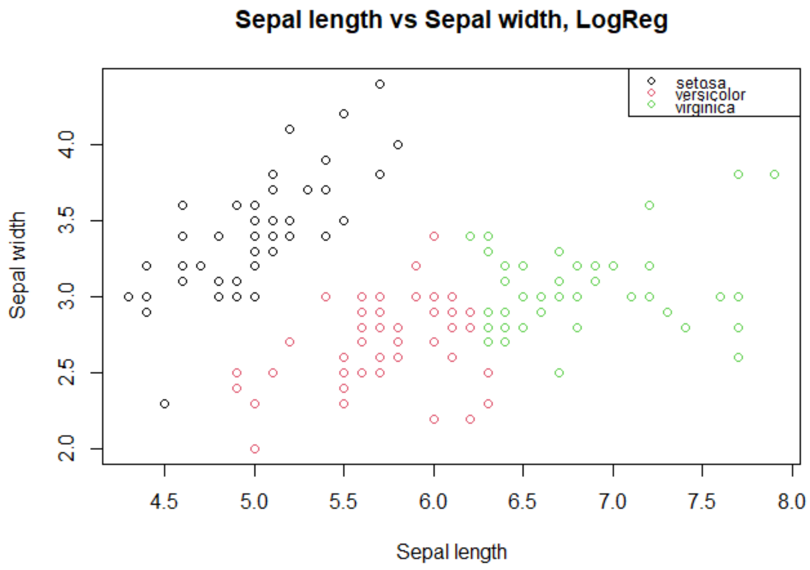


Figure 5: Scatterplot of logistic regression classification

By using the table() function for the predict() of logreg and species of the original data the misclassification rate is 16.67% percent, which is marginally better than LDA. It can be seen in the scatterplot that the logistical regression is not making linear decision boundaries as the LDA is doing, which is a marginally better method given this data.

Assignment 2

1. Import the data – remove duration

The banking data was imported as specified on the lecture slides, with a modification to make all strings as factors for easier handling.

2. Fit decision trees to the training data

The following trees were obtained in task a (default tree), b (larger minimum node size), c (smaller minimum deviance). The tree obtained in c is plotted without text so that the many splits can be observed.

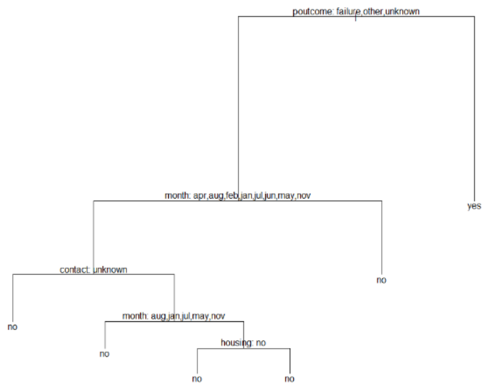

poutcome: failure,other,unknown
month: apr,aug,feb,jan,jul,jun,may,nov
yes
contact: unknown
no
month: aug,jan,jul,may,nov
no
no
housing: no
no
no

Figure 6 The default tree in a

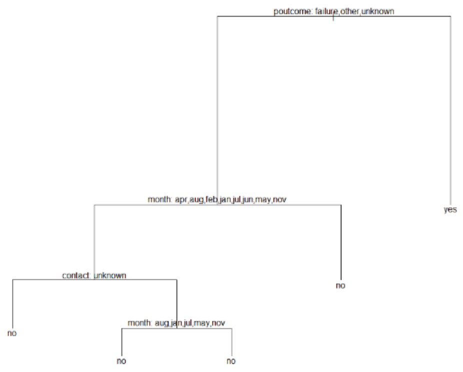

poutcome: failure,other,unknown
month: apr,aug,feb,jan,jul,jun,may,nov
yes
contact: unknown
no
no
month: aug,jan,jul,may,nov
no
no

Figure 7 The tree with minimum node size = 7000 in b

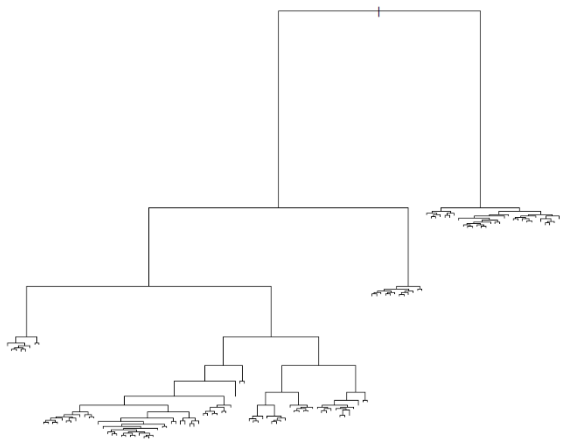


Figure 8 Tree with minimum deviance = 0.0005 in c

The trees had the following misclassification:

Data type	Default (a)	Node size (b)	Within-node deviance (c)
Training data	0.1048	0.1048	0.09362
Validation data	0.1092679	0.1092679	0.1118484

Best

The third tree (c) has the lowest misclassification for training data, but had a much larger value for the validation data, indicating severe overfitting. The first and second tree have the same misclassification since the trees are the same until the last split. Because that last split (in a) does not contribute to make a better classification (result are no no), the misclassification does not become better. Therefore, the second tree (b) is the best, since it has the smallest misclassification for validation data and is the less complex.

Node size

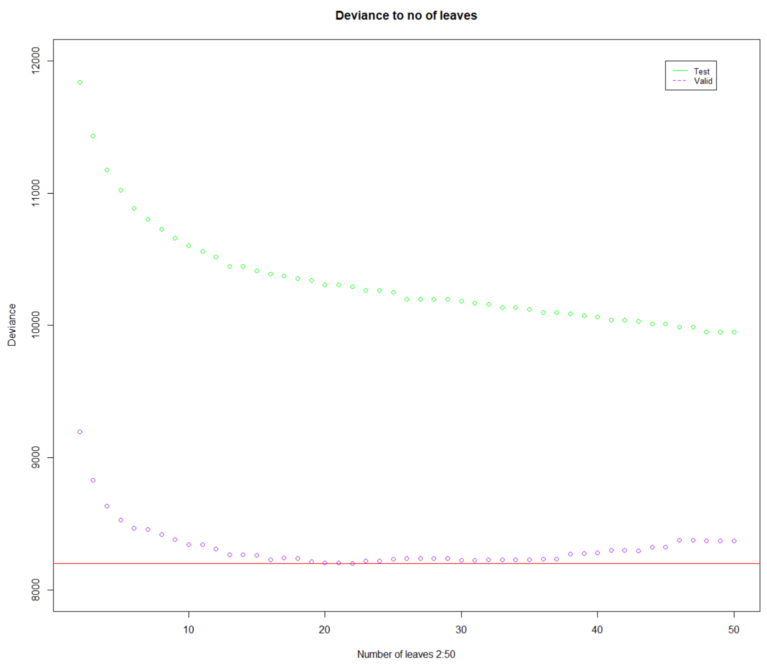
The increased value of required minimum node size for a split to happen (b) makes the splitting stop earlier than in the default tree (the default is minsize = 10). This is because the number of observations in the resulting nodes become lower than 7000 after the second split of month, thus not allowing the split to continue in b.

Within-node deviance

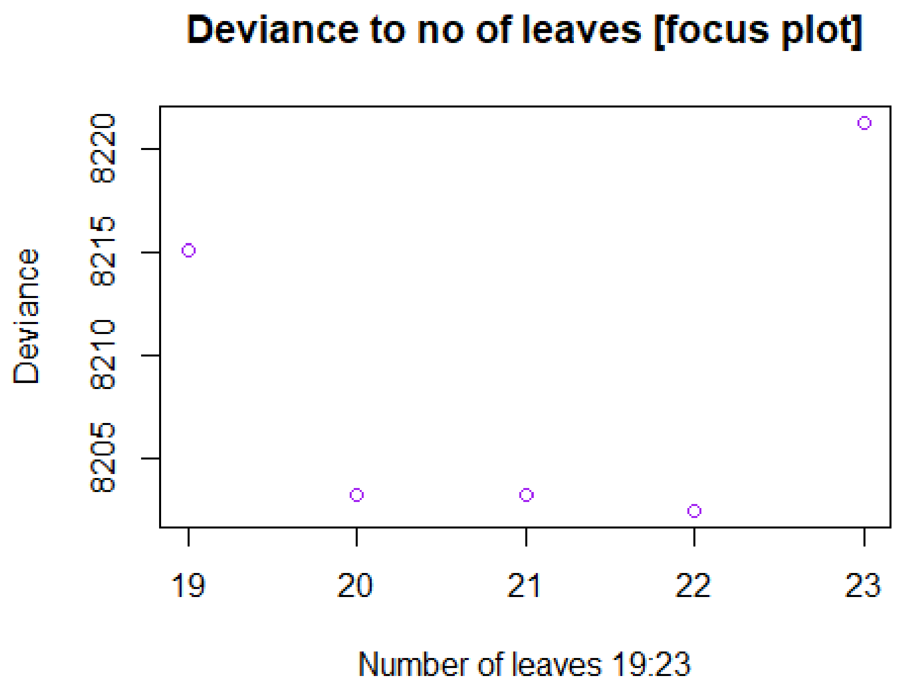
Which lower deviance in c compared to the default in a, the split in all nodes continue longer. Deviance is the variability among the observations that reach the specific node, compared to the root node. With lower mindev in c we thus allow smaller variability and therefore split can continue for longer.

3. *Choose the optimal tree depth by pruning the tree*

The tree was pruned with 2 to 50 leaves to see how the model performed on training and validation data. The result is shown in the plot below, with a linear red line to help determine which validation point is the lowest:



The help line indicates that the best explanatory power of the tree is around 20 leaves. We can see that with an increasing amount of leaves the (residual mean) deviance is decreased up until a certain point for the validation data, where the model becomes too fitted to the data. A zoomed in plot was made which showed that 22 leaves provides the smallest deviance in the validation set, which means that 22 leaves is the best.



Using summary on the optimal model with 22 leaves, we could see that the following variables were the most important to categorize the input:

```
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"  "pdays"    "age"      "day"      "balance"
## [8] "housing"  "job"
```

The misclassification rate for training data is 0.1039 and residual mean deviance is 0.5698, which means that the classification for training data is better than for models a and b. When plotting the tree we can see that the yes from the first split is further branching off, which creates a possibility to categorize some of those yes:es as no:s, thus creating greater detailed classification.

The confusion matrix for the test data was calculated to be the following:

```
table("Target"=test$y, "Predicted"=test.opt)[c(2,1),c(2,1)]

##       Predicted
## Target   yes    no
##    yes   214  1371
##    no    107 11872
```

It is clear that there are quite many false positives (1371), which is problematic for the model. False positives are often seen as very severe and should be reduced for the predictive power of the model to be good.

The misclassification was determined as 0.1089649 for test data, which is around the same value as models a and b for the validation data, and better than the validation data for c.

4. *Perform classification of the test data with the following loss matrix:*

$$L = \begin{array}{cc} & \textit{Predicted} \\ \textit{Observed} & \begin{array}{c} yes \\ no \end{array} \begin{pmatrix} 0 & 5 \\ 1 & 0 \end{pmatrix} \end{array}$$

The optimal model with applied loss function to punish false positives returned the following confusion matrix:

```
table(test$y, test.loss)[c(2,1),c(2,1)]

##      test.loss
##         yes    no
##   yes   778   807
##   no   1099 10880
```

Compared to the confusion matrix of the optimal function from 3 (but on test data) below, we can see that using the loss function has decreased the false positives as we wanted, but at the same time increased the true negatives. This is because with the loss matrix we signal that we accept an increase in true negatives if we can decrease the true positives.

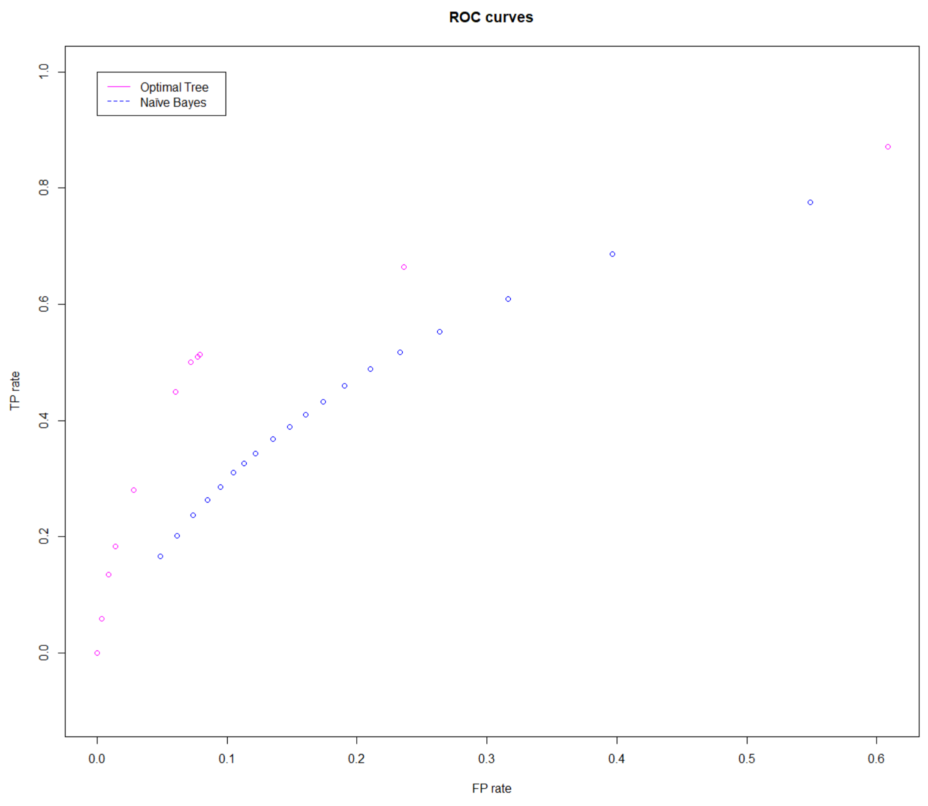
```
table(test$y, test.opt)[c(2,1),c(2,1)]

##      test.opt
##         yes    no
##   yes   214  1371
##   no    107 11872
```

5. *Classify the data with tree and naïve bayes following the given principle:*

$$\hat{Y} = 1\ if\ p(Y = 'good'|X) > \pi, otherwise\ \hat{Y} = 0$$

where π=0.05, 0.1, 0.15, ... 0.9, 0.95



We can see that the curve for the optimal tree is higher than the naïve bayes, which means that it has higher ratio of TP for the same FP rate. This in turn means that the optimal tree is better (it has a larger integral)

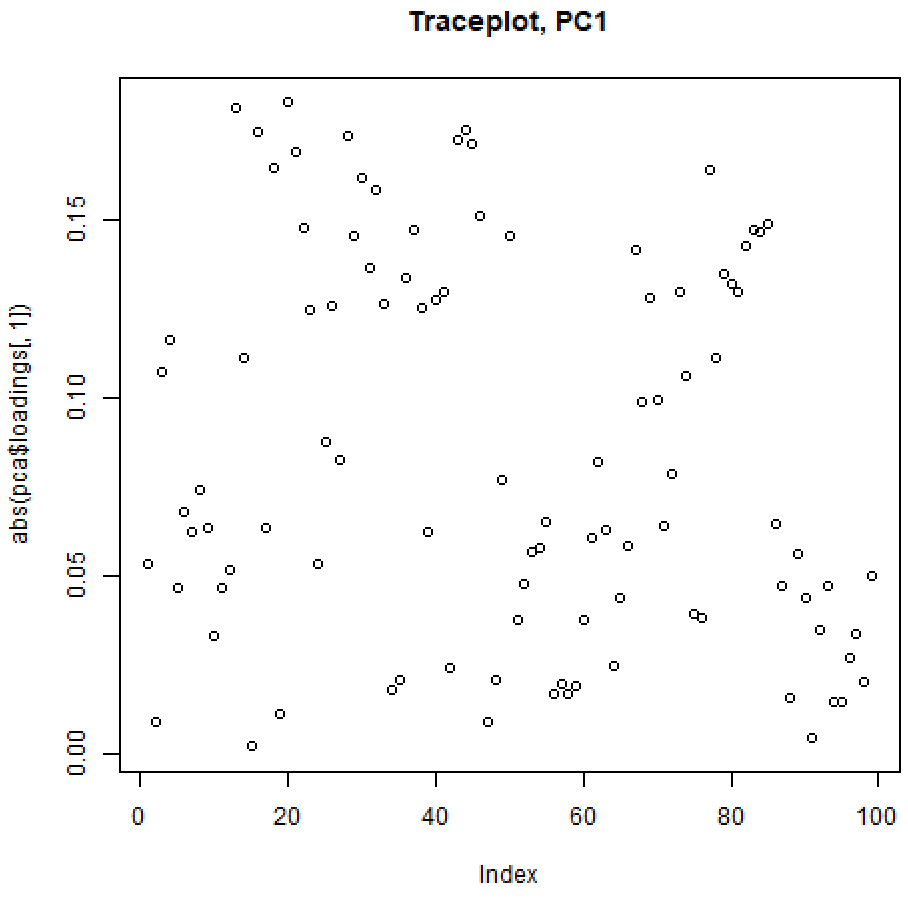
Assignment 3

1. The PCA was implemented using the covariance matrix and eigen on the scaled features. Looking at the sum of the percentile variance covered by each value of M:

```
##  [1]  25.26876  42.24434  51.63082  59.25119  64.94147  69.18331  72.44046
##  [8]  75.39113  77.48000  79.07403  80.60583  82.05765  83.43232  84.47099
## [15]  85.41223  86.30769  87.06384  87.77618  88.43155  89.06784  89.67219
## [22]  90.21954  90.74496  91.25705  91.74495  92.21661  92.67796  93.11349
## [29]  93.50811  93.87721  94.23797  94.57802  94.89676  95.19076  95.46562
## [36]  95.72475  95.97596  96.22064  96.44476  96.65848  96.86604  97.06875
## [43]  97.26079  97.44710  97.61372  97.77585  97.91941  98.06104  98.18870
## [50]  98.30247  98.41164  98.51815  98.61933  98.71018  98.79247  98.87028
## [57]  98.94462  99.01447  99.08130  99.14603  99.20858  99.26750  99.31893
## [64]  99.36851  99.41485  99.45909  99.50138  99.54213  99.58066  99.61646
## [71]  99.65067  99.68289  99.71375  99.74211  99.76797  99.79205  99.81474
## [78]  99.83605  99.85630  99.87490  99.89241  99.90810  99.92214  99.93495
## [85]  99.94652  99.95591  99.96438  99.97044  99.97630  99.98149  99.98566
## [92]  99.98950  99.99226  99.99432  99.99611  99.99743  99.99855  99.99936
## [99] 100.00000
```

   We see that 95 percent of variance is reached when 34 features are used, i.e. M=34
   The first two components explains 25.269 and 16.806 percent of the total variance.

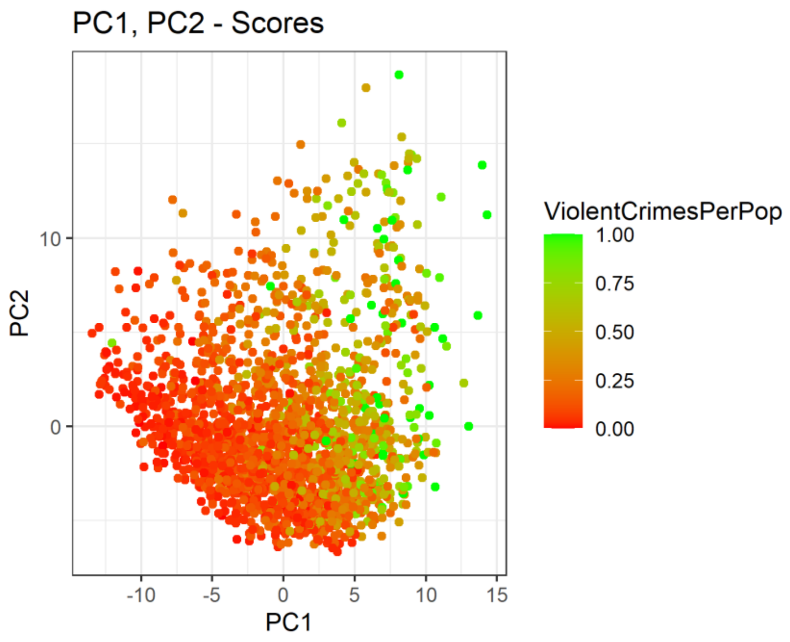
2. PCA was calculated using princomp, and looking at the first principal component, the features contribution to the component is plotted as follows using the absolute values:



   We see that many features are in the top part of the graph, indicating that several features have a notable contribution to the component. The ones that contribute the most can be seen below:

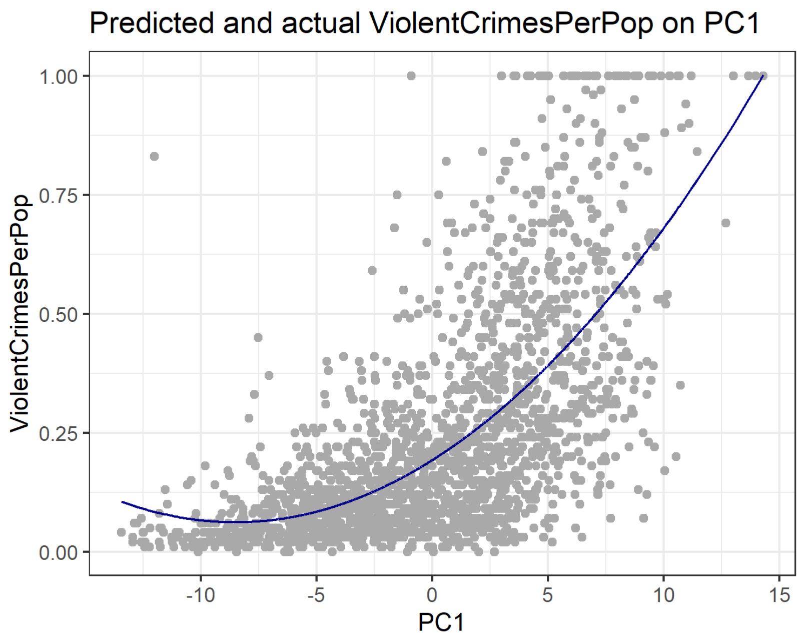
```
##       medFamInc      medIncome     PctKids2Par      pctWInvInc PctPopUnderPov
##       0.1832453      0.1819115       0.1755956       0.1749060      0.1738039
```

These all seem to be income or economy related, as three are directly connected to income, amount of kids indicate costs among other things, and poverty also connect directly to economy. All this seem to indicate that the most important factor to crime rate is the economic situation.
Looking at PC1 and PC2 together:



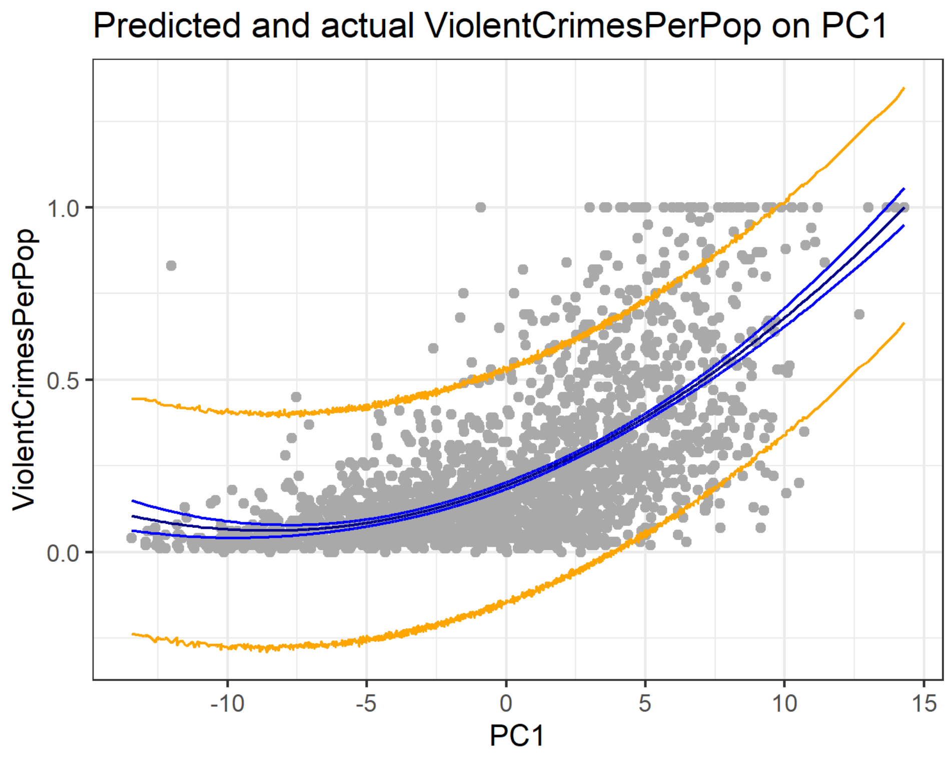
We see that although PC2 has relatively high variation, the score seems to be mainly based on the x-axis, indicating that PC1 is more important for the final crime rate score.

3. We compute the model for PC1 and crime rate as seen in the code below, which results in the following plot:



We see that there does not seem to be a very clear relationship between the feature and target, which might also be indicated by the fact that PC1 only explains about 25 percent of the total feature variance. It is also made clear from the model, which does not seem to capture the relationship very well, especially for larger values of PC1. It follows the general shape of the relationship but does not increase steeply enough.

4. We add the confidence and prediction intervals for the model which result in the following plot:



   a. The confidence interval can be seen as blue lines in the plot. This shows a rather small interval, which would indicate that the parameters of the model are close to the true parameters.
   b. The prediction interval is orange in the plot and shows a larger interval. Prediction interval describes the variation in individual values of the model, and therefor does not necessarily decrease with more data points. The width of the band seems reasonable as the datapoints do show a large variation.

Together, the confidence interval would indicate that the model captures the component as well as possible, but the prediction interval shows that the model, or component itself, does not manage to predict the crime rate very well on its own.

Statement of Contribution

- Assignment 1: Was solved by Per Bark (perba583)
- Assignment 2: Was solved by Ingrid Wendin (ingwe018)
- Assignment 3: Was solved by David Åström (davas593)

The report was written by all three group members, who wrote their respective part, and then control read the whole report.

Appendix – R-code

Assignment 1

```r
# Lab 2 TDDE01
# Assignment 1. LDA and logistic regression
set.seed(12345)

# Getting library for LDA and QDA
library(MASS)

# Loading the data from the flowers
mydata <- iris

plot(x=mydata$Sepal.Length, y=mydata$Sepal.Width, main="Sepal length vs Sepal width", xlab
= "Sepal length", ylab = "Sepal width", col = mydata$Species)
legend('topright', legend = levels(mydata$Species), col = 1:3, cex = 0.8, pch = 1)
```

```r
# Given the pattern of the data, setosa and virginica are distinctly distributed offset fr
om versicolor to for LDA % logistical regression to be used

# 2a. Computing mean

# Splitting data based on species
setosa.data <- mydata[0:50, ]
versicolor.data <- mydata[51:100, ]
virginica.data <- mydata[101:150, ]

# Calculating mean length
mean(setosa.data$Sepal.Length)

## [1] 5.006

mean(versicolor.data$Sepal.Length)

## [1] 5.936

mean(virginica.data$Sepal.Length)

## [1] 6.588

# Calculating mean width
mean(setosa.data$Sepal.Width)

## [1] 3.428

mean(versicolor.data$Sepal.Width)

## [1] 2.77

mean(virginica.data$Sepal.Width)

## [1] 2.974

# Constructing mean vectors
setosa.mean <- c(mean(setosa.data$Sepal.Length), mean(setosa.data$Sepal.Width))
versicolor.mean <- c(mean(versicolor.data$Sepal.Length), mean(versicolor.data$Sepal.Width)
)
virginica.mean <- c(mean(virginica.data$Sepal.Length), mean(virginica.data$Sepal.Width))

# Calculating covariance matrices
cov.matrix.setosa <- cov(setosa.data[1:2])
cov.matrix.versicolor <- cov(versicolor.data[1:2])
cov.matrix.virginica <- cov(virginica.data[1:2])

prior.setosa = nrow(setosa.data)/nrow(mydata)
prior.versicolor = nrow(versicolor.data)/nrow(mydata)
prior.virginica = nrow(virginica.data)/nrow(mydata)


# 2b Compute overall (pooled) covariance matrix
pooled.cov <- (1/150)*(50)*(cov.matrix.setosa+cov.matrix.versicolor+cov.matrix.virginica)


# 2c. LDA probabilistic model
# x|𝑦 = Ci,𝜇i, Σ ~N(𝜇i,Σ)
# 𝑦|𝜋~ Multinomial(𝜋1, …,𝜋k)

# Sigma is pooled variance of all classes
# mu is mean vector
# 𝜋k is probability distribution of class, in this case relative frequencies of class in t
he data (prior)


# 2d. Discriminant functions for each class
# delta.setosa = x^t*pooled.cov*setosa.mean + (-1/2)setosa.mean^t*pooled.cov*setosa.mean +
log(prior)

#Setosa
w0.setosa = (-1/2)*t(setosa.mean)%*%solve(pooled.cov)%*%setosa.mean + log(prior.setosa) #i
ntercept
w.setosa = solve(pooled.cov)%*%setosa.mean #feature coefficients

# delta.setosa = x%*%(11.81827, 20.21183) - 65.32281


#Versicolor
w0.versicolor = (-1/2)*t(versicolor.mean)%*%solve(pooled.cov)%*%versicolor.mean + log(prio
r.versicolor)
w.versicolor = solve(pooled.cov)%*%versicolor.mean

# delta.versicolor = x%*%(19.475666, 8.356129) - 70.47563


#Virginica
w0.virginica = (-1/2)*t(virginica.mean)%*%solve(pooled.cov)%*%virginica.mean + log(prior.v
irginica)
w.virginica = solve(pooled.cov)%*%virginica.mean

# delta.virginica = x%*%(22.037717, 8.065317) - 85.68398


# 2e. Compute equations of decision boundaries
# Decision boundary is computed from: wi*x+w0i = wj*x+w0j

# Boundary setosa-versicolor
w0.setver = w0.setosa - w0.versicolor
w.setver = w.setosa-w.versicolor

#DB.setver = x%*%(-7.657399, 11.855698) + 5.152822

# Boundary setosa-virginica
```

```r
w0.setvir = w0.setosa - w0.virginica
w.setvir = w.setosa-w.virginica

#DB.setvir = x%*%(-10.21945, 12.14651) + 15.20835

# Boundary versicolor-virginica
w0.vervir = w0.versicolor - w0.virginica
w.vervir = w.versicolor-w.virginica

#DB.vervir = x%*%(-2.5620509, 0.2908121) + 20.36117

#Predicting with constructed discriminant functions

#Setosa discriminant function calculated
delta.setosa = as.matrix(mydata[1:2])%*%w.setosa
for (i in 1:150) {
  delta.setosa[i] = delta.setosa[i] + w0.setosa
}


#Versicolor discriminant function calculated
delta.versicolor = as.matrix(mydata[1:2])%*%w.versicolor
for (i in 1:150) {
  delta.versicolor[i] = delta.versicolor[i] + w0.versicolor
}

#Virginica discriminant function calculated
delta.virginica = as.matrix(mydata[1:2])%*%w.virginica
for (i in 1:150) {
  delta.virginica[i] = delta.virginica[i] + w0.virginica
}

#Compounding results
delta.result <- cbind(delta.setosa, delta.versicolor, delta.virginica)
colnames(delta.result) <- c("setosa", "versicolor", "virginica")

# Class result compounding
delta.predicted <- 1:150
for (i in 1:150) {
if (which.max(delta.result[i,])==1) {
  delta.predicted[i]="setosa"
  }
if (which.max(delta.result[i,])==2) {
  delta.predicted[i]="versicolor"
  }
if (which.max(delta.result[i,])==3) {
  delta.predicted[i]="virginica"
  }
}

# Plot of predicted classes
deltaplot <- data.frame(cbind(mydata$Sepal.Length, mydata$Sepal.Width, delta.predicted))
colnames(deltaplot) <- c("Sepal Length", "Sepal Width", "Species")
plot(x=deltaplot$`Sepal Length`, y=deltaplot$`Sepal Width`, main="Sepal length vs Sepal wi
dth, LDA", xlab = "Sepal length", ylab = "Sepal width", col=as.factor(deltaplot$Species))
legend('topright', legend = levels(iris$Species), col = 1:3, cex = 0.8, pch = 1)

abline(-( 5.152822/11.855698), -(-7.657399/11.855698), col =8) #Setver
abline(-(20.36117/12.14651), -(-10.21945/ 12.14651), col =8) #Setvir
abline(-( 15.20835/0.2908121), -(-2.5620509/0.2908121), col =8) #Vervir
```

```r
# Estimate missclassification
table(delta.predicted, mydata$Species)

##
## delta.predicted setosa versicolor virginica
##      setosa         49          0         0
##      versicolor      1         36        15
##      virginica       0         14        35

missclassicitaion.rate = (1+15+14)/150
# missclassicitaion.rate = 0.2=20%, quite high, though clear majority (all except one) are
versicolor and virgninica missclassified

# Performing LDA with lda() function
linnear = lda(Species ~ Sepal.Length + Sepal.Width, mydata)
table(predict(linnear)$class, mydata$Species)

##
##                setosa versicolor virginica
##   setosa         49          0         0
##   versicolor      1         36        15
##   virginica       0         14        35

# Observe that the tables are the same
# Since it is the same method, the results should be the same

# install.packages(), installed mvtnorm to load it
library(mvtnorm)

# First sample new data points, start with species
samples <- sample(unique(mydata$Species), 150, replace=T, prob=c(prior.setosa, prior.versi
color, prior.virginica))
table(samples) # 62 setosa, 46 versicolor & 42 virginica

## samples
##     setosa versicolor  virginica
##         60         45         45

# Generate parameters
sample.setosa <- cbind(rmvnorm(62, mean = setosa.mean, sigma = cov.matrix.setosa), "setosa
")
sample.versicolor  <- cbind(rmvnorm(46, mean = versicolor.mean, sigma = cov.matrix.versico
lor), "versicolor")
sample.virginica  <- cbind(rmvnorm(42, mean = virginica.mean, sigma = cov.matrix.virginica
), "virginica")

# Combining samples to one matrix
sampledata <- rbind(sample.setosa, sample.versicolor, sample.virginica)

# Making plot of new data
plot(x=sampledata[,1], y=sampledata[,2], main="Sepal length vs Sepal width, new data", xla
b = "Sepal length", ylab = "Sepal width", col=as.factor(sampledata[,3]))
legend('topright', legend = levels(iris$Species), col = 1:3, cex = 0.8, pch = 1)
```

```r
# Load nnet package
library(nnet)

# Prepare sampledata
```

```r
sampledata <- data.frame(sampledata[,1], sampledata[,2], sampledata[,3])
colnames(sampledata) <- c("Sepal.Length", "Sepal.Width", "Species")

# Perform regression
logreg <- multinom(Species ~ Sepal.Length + Sepal.Width, mydata)

## # weights:  12 (6 variable)
## initial  value 164.791843
## iter  10 value 62.715967
## iter  20 value 59.808291
## iter  30 value 55.445984
## iter  40 value 55.375704
## iter  50 value 55.346472
## iter  60 value 55.301707
## iter  70 value 55.253532
## iter  80 value 55.243230
## iter  90 value 55.230241
## iter 100 value 55.212479
## final  value 55.212479
## stopped after 100 iterations

# Plot of logistic regression
plot(x=mydata$Sepal.Length, y=mydata$Sepal.Width, main="Sepal length vs Sepal width, LogRe
g", xlab = "Sepal length", ylab = "Sepal width", col = as.factor(predict(logreg)))
legend('topright', legend = levels(iris$Species), col = 1:3, cex = 0.8, pch = 1)
```

```r
# Checking classification errors
table(predict(logreg), mydata$Species)

##
##                setosa versicolor virginica
##   setosa           50          0         0
##   versicolor        0         38        13
##   virginica         0         12        37

MCrate=(13+12)/150 #16.67%
```

Assignment 2

```r
data = read.csv2("bank-full.csv", stringsAsFactors = TRUE)

# 1 remove duration - column
data = data[-12]
n = dim(data)[1]

set.seed(12345)
id = sample(1:n, floor(n*0.4))
train = data[id, ]

id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.3))
valid = data[id2, ]

id3 = setdiff(id1, id2)
test = data[id3, ]

# 2 Fit decision tree to training data (y = yes/no)

library(tree)

# 2a
tree.a <- tree(as.factor(y)~., data=train)
summary(tree.a)
## Misclassification error rate: 0.1048 = 1896 / 18084

plot(tree.a)
text(tree.a, pretty=0)

# 2b smallest node size = 7000 (up from 10)
tree.b <- tree(as.factor(y)~., data=train, control = tree.control(nobs=18084, mins
ize=7000))
summary(tree.b)
## Misclassification error rate: 0.1048 = 1896 / 18084

plot(tree.b)
text(tree.b, pretty=0)

# 2c Decision trees minimum deviance to 0.0005. (down from 0.01)
tree.c <- tree(as.factor(y)~., data=train, control = tree.control(nobs = 18084, mi
ndev=0.0005))
summary(tree.c)

## Misclassification error rate: 0.09362 = 1693 / 18084

plot(tree.c)
text(tree.c, pretty=0)


# validation
y.a <- predict(tree.a, newdata = valid, type="class") #class -> get the output as
yes/no
mis.a = 1-sum(diag(table(valid$y, y.a)))/length(y.a)
mis.a
```

```
## [1] 0.1092679

y.b <- predict(tree.b, newdata = valid, type="class")
mis.b = 1-sum(diag(table(valid$y, y.b)))/length(y.b)
mis.b

## [1] 0.1092679

y.c <- predict(tree.c, newdata = valid, type="class")
mis.c = 1-sum(diag(table(valid$y, y.c)))/length(y.c)
mis.c

## [1] 0.1118484


# 3 - optimal depth of 2c

trainScore = rep(0,50)
testScore = rep(0,50)

# 2:50 because object of class "singlenode" when only 1 leaf = only root
for(i in 2:50) {
  prunedTree = prune.tree(tree.c, best=i)
  pred.valid = predict(prunedTree, newdata = valid, type="tree") #tree -> get outp
ut as a tree
  trainScore[i] = deviance(prunedTree)
  testScore[i] = deviance(pred.valid)
}

plot(2:50, trainScore[2:50], col="green", ylim = c(8000,12000), main = "Deviance t
o no of leaves", xlab = "Number of leaves 2:50", ylab = "Deviance")
points(2:50, testScore[2:50], col="purple")
legend(x=45, y=12000, inset = 0.02, legend = c("Test", "Valid"), col = c("green",
"purple"), lty = 1:2, cex = 0.8 )
abline(8200, 0, col = "red")

# Control - zoom in
plot(19:23, testScore[19:23], col = "purple",  main = "Deviance to no of leaves [f
ocus plot]", xlab = "Number of leaves 19:23", ylab = "Deviance")


# Most important variables
tree.opt = prune.tree(tree.c, best = 22)
summary(tree.opt)
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"  "pdays"    "age"      "day"      "balance"
## [8] "housing"  "job"

print(tree.opt)

plot(tree.opt)
```

```
# Misclass rate and confusion matrix for the test data
test.opt <- predict(tree.opt, newdata = test, type = "class")
```

```
table("Target"=test$y, "Predicted"=test.opt)[c(2,1),c(2,1)]

##       Predicted
## Target   yes    no
##    yes   214  1371
##    no    107 11872

mis.opt = 1-sum(diag(table(test$y, test.opt)))/length(test.opt)
mis.opt

## [1] 0.1089649

# # 4 Classification of test data with loss matrix -> need rpart
library(rpart)
# Create loss matrix - in this case "no" is positive, "yes" is negative
loss <- matrix(c(0,1,5,0), ncol = 2, byrow = TRUE)
tree.loss <- rpart(as.factor(y)~., data = train, parms = list(loss = loss))
test.loss <- predict(tree.loss, newdata = test, type = "class")

table(test$y, test.loss)[c(2,1),c(2,1)] # change displaying order

##      test.loss
##         yes    no
##   yes   778   807
##   no   1099 10880

# optimal tree from 3
table(test$y, test.opt)[c(2,1),c(2,1)]

##      test.opt
##         yes    no
##   yes   214  1371
##   no    107 11872

mis.loss <- 1-sum(diag(table(test$y, test.loss)))/length(test.loss)
mis.loss

## [1] 0.140519

# 5 optimal tree + Naive bayes
# Since the result y="yes" is interpreted as good for the company,
# Y=1=yes if the probability of y="yes" > pi
# -> find probabilities of opt to classify as yes

test.opt.prob <- predict(tree.opt, newdata = test, type = "vector")
# translate yes and no in test data to 1 and 0 for comparison
test.num <- matrix(ifelse(test$y == "yes", 1, 0))


tp.rate <- data.frame()
fp.rate <- data.frame()

N.plus = sum(test.num)
N.minus = length(test.num) - N.plus

for (i in seq(from = 0.05, to = 0.95, by = 0.05)) {
  # compare probability of yes with pi, if greater than pi then y.hat = 1
  y.hat <- matrix(ifelse(test.opt.prob[,2]>i, 1, 0))

  # CONTROL confusion matrix with order
  #tn, fn, fp, tp
  #t = table(test.num, y.hat)

  tp = sum(y.hat*test.num)
  tp.rate = rbind(tp.rate, tp/N.plus)


  fp = sum(y.hat[which(y.hat == 1)])-tp
  fp.rate = rbind(fp.rate, fp/N.minus)

}

names(tp.rate)[1] <- c("tp.rate")
names(fp.rate)[1] <- c("fp.rate")

# Naive Bayes
library(e1071)
bayes.fit = naiveBayes(as.factor(y)~., data = train)
# yes and no class probabilities
bayes.pred = predict(bayes.fit, newdata = test, type="raw") # raw -> yes/no answer
s

bayes.tp.rate <- data.frame()
bayes.fp.rate <- data.frame()

for (i in seq(from = 0.05, to = 0.95, by = 0.05)) {
  # compare probability of yes with pi, if greater than pi then y.hat = 1
  bayes.y.hat <- matrix(ifelse(bayes.pred[,2]>i, 1, 0))

  # CONTROL confusion matrix with order
  #tn, fn, fp, tp
  #t = table(test.num, bays.y.hat)

  bayes.tp = sum(bayes.y.hat*test.num)
  bayes.tp.rate = rbind(bayes.tp.rate, bayes.tp/N.plus)


  bayes.fp = sum(bayes.y.hat[which(bayes.y.hat == 1)])-tp
  bayes.fp.rate = rbind(bayes.fp.rate, bayes.fp/N.minus)

}

names(bayes.tp.rate)[1] <- c("bayes.tp.rate")
names(bayes.fp.rate)[1] <- c("bayes.fp.rate")


# ROC curve
#fp rate  vs tp rate ROC?
plot(fp.rate$fp.rate, tp.rate$tp.rate, ylab = "TP rate", xlab = "FP rate", ylim =
c(-0.1,1), col = "magenta", main ="ROC curves")
points(bayes.fp.rate$bayes.fp.rate, bayes.tp.rate$bayes.tp.rate, ylab = "TP rate",
```

```r
xlab = "FP rate", ylim = c(-0.1,1), col = "blue")
legend(0,1, legend = c("Optimal Tree", "Naïve Bayes"), col = c("magenta", "blue"),
lty=1:2)
```

Assignment 3

```
library(ggplot2)
data = read.csv("communities.csv")
set.seed(12345)

# scale features
features.scaled = scale(data[,2:100])

#get covariance matrix
covMatrix = cov(features.scaled)

#calculate and split eigenvalues/vectors
eigen = eigen(covMatrix)
eigenValues = eigen$values
eigenVectors = eigen$vectors

#calculate percentage of variance covered by each possible M
percentile = cumsum(eigenValues)/sum(eigenValues) * 100
percentile

##  [1]  25.26876  42.24434  51.63082  59.25119  64.94147  69.18331  72.44046
##  [8]  75.39113  77.48000  79.07403  80.60583  82.05765  83.43232  84.47099
## [15]  85.41223  86.30769  87.06384  87.77618  88.43155  89.06784  89.67219
## [22]  90.21954  90.74496  91.25705  91.74495  92.21661  92.67796  93.11349
## [29]  93.50811  93.87721  94.23797  94.57802  94.89676  95.19076  95.46562
## [36]  95.72475  95.97596  96.22064  96.44476  96.65848  96.86604  97.06875
## [43]  97.26079  97.44710  97.61372  97.77585  97.91941  98.06104  98.18870
## [50]  98.30247  98.41164  98.51815  98.61933  98.71018  98.79247  98.87028
## [57]  98.94462  99.01447  99.08130  99.14603  99.20858  99.26750  99.31893
## [64]  99.36851  99.41485  99.45909  99.50138  99.54213  99.58066  99.61646
## [71]  99.65067  99.68289  99.71375  99.74211  99.76797  99.79205  99.81474
## [78]  99.83605  99.85630  99.87490  99.89241  99.90810  99.92214  99.93495
## [85]  99.94652  99.95591  99.96438  99.97044  99.97630  99.98149  99.98566
## [92]  99.98950  99.99226  99.99432  99.99611  99.99743  99.99855  99.99936
## [99] 100.00000

#calculate Lowest M that covers 95% of variance
M95 = Position(function(x) x > 95, percentile)
M95

## [1] 34

#percentage of variance covered by PC1 and PC2
eigenValues[1]

## [1] 25.01607

eigenValues[2]

## [1] 16.80582

# calculate PCA using princomp
pca = princomp(features.scaled)

# show contribution of PC1
png("traceplot_pc1.png")
plot(abs(pca$loadings[,1]), main = "Traceplot, PC1")
dev.off()

# get top 5 features of PC1
loadings.PC1.sorted = sort(abs(pca$loadings[,1]), decreasing = TRUE)
```

```r
loadings.PC1.top_five = loadings.PC1.sorted[1:5]
loadings.PC1.top_five

##      medFamInc      medIncome    PctKids2Par     pctWInvInc PctPopUnderPov
##      0.1832453      0.1819115      0.1755956      0.1749060      0.1738039

# plot PC1, PC2 scores
data.plot = as.data.frame(cbind(data$ViolentCrimesPerPop, pca$scores))

# create plot
plot = ggplot(data.plot)
plot = plot + geom_point(aes(x = Comp.1, y = Comp.2, colour = V1)) + scale_colour_gradient
(low = "red", high = "green") + labs(x = "PC1", y = "PC2", colour = "ViolentCrimesPerPop",
title = "PC1, PC2 - Scores") + theme_bw()
# save plot
ggsave("PC1_PC2_scores.png")

# Polynomial regression model
data.regression = data.plot[,1:2]
data.regression = data.regression[order(data.regression$Comp.1),]
model = lm(V1 ~ poly(Comp.1, 2), data = data.regression)
fitted = predict(model)
data.regression$predictedV1 = fitted

# create plot
plot.regression = ggplot(data.regression)
plot.regression = plot.regression + geom_point(aes(x = Comp.1, y = V1), colour = "darkGrey
") + geom_line(aes(x = Comp.1, y = predictedV1), colour = "darkBlue") + labs(x = "PC1", y
= "ViolentCrimesPerPop", title = "Predicted and actual ViolentCrimesPerPop on PC1") + them
e_bw()
# save plot
ggsave("vcpp_pc1_3.png")

# Bootstrap
library(boot)

# boot rng-function
rng = function(data, model) {
  data1 = data.frame(V1 = data$V1, Comp.1 = data$Comp.1)
  n = length(data$V1)
  data1$V1 = rnorm(n, predict(model, newdata = data1), sd(model$residuals))
  return(data1)
}

# statistic function for confidence band bootstrap
statistic.confidence = function(data1) {
  res = lm(V1 ~ poly(Comp.1, 2), data = data1)
  V1.Predicted = predict(res, newdata = data.regression)
  return(V1.Predicted)
}

# calculate and envelope the confidence band
confidence = boot(data.regression, statistic = statistic.confidence, R = 1000, mle = model
, ran.gen = rng, sim = "parametric")
e.confidence = envelope(confidence)

# add confidence band lines to plot
plot.bootstrap = plot.regression + geom_line(aes(x = Comp.1, y = e.confidence$point[1,]),
colour = "blue") + geom_line(aes(x = Comp.1, y = e.confidence$point[2,]), color = "blue")

# statistic function for prediction band bootstrap
```

```
statistic.prediction = function(data1) {
  res = lm(V1 ~ poly(Comp.1, 2), data = data1)
  V1.Predicted = predict(res, newdata = data.regression)
  n = length(data.regression$V1)
  Predicted = rnorm(n, V1.Predicted, sd(model$residuals))
  return(Predicted)
}

# calculate and envelope the prediction bands
prediction = boot(data.regression, statistic = statistic.prediction, R = 10000, mle = mode
l, ran.gen = rng, sim = "parametric")
e.prediction = envelope(prediction)

# add prediction band lines to plot
plot.bootstrap = plot.bootstrap + geom_line(aes(x = Comp.1, y = e.prediction$point[1,]), c
olour = "orange") + geom_line(aes(x = Comp.1, y = e.prediction$point[2,]), color = "orange
")
# save the bootstrap plot
ggsave("vcpp_pc1_4.png")
```