Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
IDP Institut für Datenanalyse
und Prozessdesign

HS 2016

Statistisches Data Mining (StDM)

Woche 11

Aufgabe 1

Krabben Geschlecht

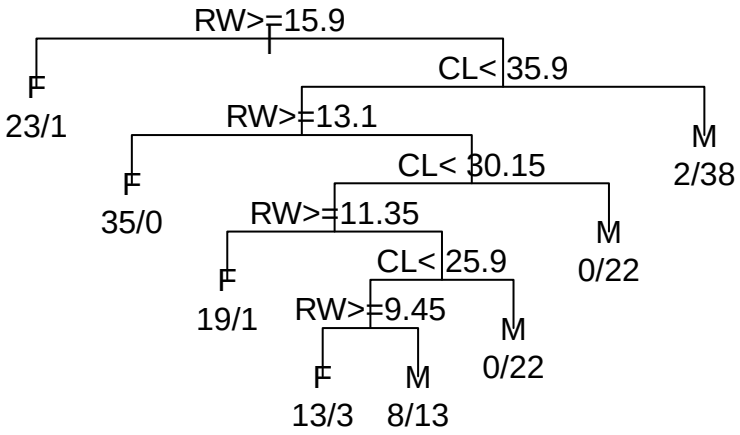
Wir betrachten den Crabs Datensatz, der im MASS package enthalten ist. Machen Sie sich mit dem Datensatz und der rpart-Funktion vertraut.

a) Erstellen Sie mit der Funktion rpart aus der rpart Library einen Klassifikationsbaum für die Variable Sex. Stellen Sie den Klassifikationsbaum grafisch dar. Quantifizieren Sie die Performance auf dem Trainingsdatensatz durch eine Kofusionsmatrix. R-Hinweise:

```
library(MASS)  # Datensatz crabs
library(rpart)  # rpart-Funktion
#?crabs
#Wir sehen, dass die Daten 5 Eigenschaften von einer bestimmten Krabbenart in Australien beschreib
#Dabei sind fast alle Eigenschaften bestimmte Groessen wie Koerperhoehe.
#Es wurde auch das Geschlecht des jeweiligen Tiers erfasst.

t1 <- rpart(sex ~., data=crabs)
class(t1)
```

```
## [1] "rpart"
```

```
#Der Klassifikationsbaum wird im Workspace als Liste angezeigt ist aber ein Objekt des Typs "rpart
#t1
#Hier erkennen wir die verschiedenen Splits welche vom Algorithmus gefunden wurden.

#Zur Darstellung des Dendogramms muss ein zweistufiges Verfahren angewendet werden.
plot(t1, margin=0.2, uniform=TRUE)
#Es wird der "Baum" dargestellt. Margin=0.2 macht die Raender um den "Baum" groesser so dass
#nacher noch genug Platz fuer den Text ist.
text(t1,use.n=TRUE)
```




```
  #Jetzt das Dendogramm komplett daregestellt.

  #Der naechste Schritt besteht aus der Kontrolle der Performance unseres Klassifikationsbaums.

  #Die Konfusionsmatrix kann von Hand erstellt werden:
  table(predict(t1,type="class"),crabs$sex)
```

```
##
##      F  M
##   F 90  5
##   M 10 95
```

```
  #Wir erkennen, dass 90 von 100 Weibchen korrekt als Weibchen klassier wurden und die restliichen
  #10 faelschlicherweise als Maennchen.

  #Bei den Maennchen wurden 95 von 100 korrekt klassiert und 5 faelschlicherweise als Weibchen.

  # wir wollen confusion-Fkt des Pakets benutzen
  require(mda)
  # library(help=mda)
  confusion(object=predict(t1,type="class"), true=as.factor(crabs$sex))
```

```
##           true
## predicted  F  M
##         F 90  5
##         M 10 95
```

```
  #
  table(crabs$sex)
```

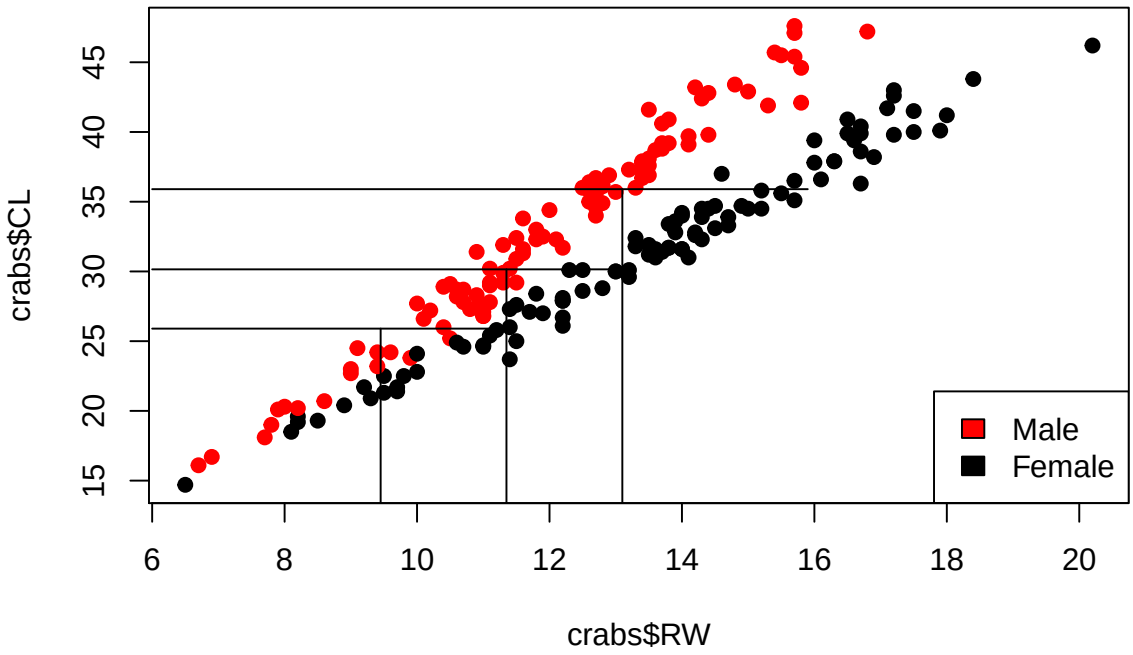
```
##
##   F   M
## 100 100
```

```
  #***** Es ist unbedingt zu beachten, dass es sich hier um Insample Betrachtungen handelt!!*****
```

b) Im Klassifikationsbaum wurden als Splitvariablen nur die Variablen RW und CL verwendet. Erstellen Sie ein Streudiagramm mit diesen beiden Variablen und färben Sie die Punkte gemäss dem Geschlecht der Krabben ein. Kennzeichnen Sie den ersten Split als Linie im Streudiagramm. Was ist der grosse Nachteil von Klassificationsbäumen?

```
plot(crabs$RW, crabs$CL,pch=19,col=crabs$sex)
legend("bottomright", legend = c("Male", "Female"), fill =  c(2,1))
```

```
#Die Weibchen sind durch schwarze Punkte dargestellt und die Maennchen durch rote!

#Jetzt zeichnen wir den ersten Split in das Streudiagramm ein.
#abline(v=15.9)

## Hier noch weitere splitts
lines(x = c(6, 15.9), y = c(35.9,35.9))#abline(h=35.9)
lines(x = c(13.1,13.1), y = c(13, 35.9)) #abline(v=13.1)
lines(x = c(6,13.1), y = c(30.15,30.15)) #abline(h=30.15)
lines(x = c(11.35, 11.35), y=c(13, 30.15)) #abline(v=11.35)
lines(x = c(6,11.35), y = c(25.9,25.9)) #abline(h=25.9)
lines(x = c(9.45,9.45), y = c(13,25.9)) #abline(v=9.45)
```




```
#  Klassifikationsbäume können nur horizontale und vertikale Grenzen ziehen. Dies ist hier ineffizie
#  da eigentlich eine Diagonale die Gruppen am besten trennt.
```

c) Führen Sie mit den numerischen Variablen des Crabs Datensatz eine PCA durch und visualisieren Sie die Position der Datenpunkte im Streudiagramm der ersten beiden Hauptkomponenten. Färben Sie die Punkte gemäss dem Geschlecht der Krabben ein.

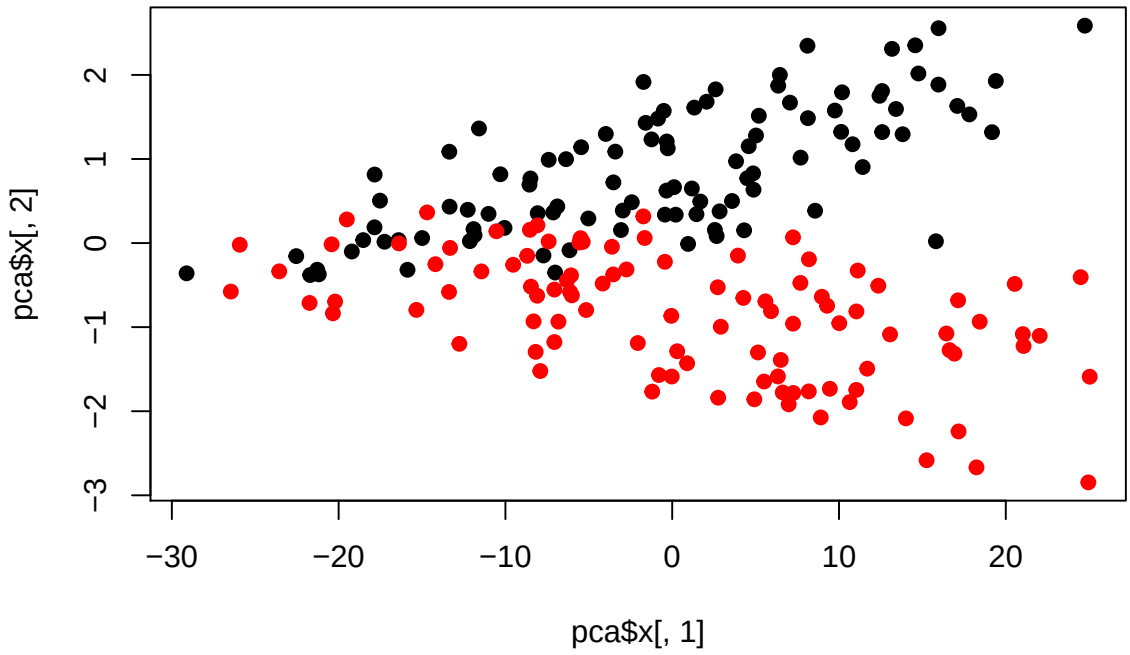
```
#Welche Variablen sind nummerisch?
str(crabs)
```

```
## 'data.frame':    200 obs. of  8 variables:
##  $ sp   : Factor w/ 2 levels "B","O": 1 1 1 1 1 1 1 1 1 1 ...
##  $ sex  : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
##  $ index: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ FL   : num  8.1 8.8 9.2 9.6 9.8 10.8 11.1 11.6 11.8 11.8 ...
##  $ RW   : num  6.7 7.7 7.8 7.9 8 9 9.9 9.1 9.6 10.5 ...
##  $ CL   : num  16.1 18.1 19 20.1 20.3 23 23.8 24.5 24.2 25.2 ...
##  $ CW   : num  19 20.8 22.4 23.1 23 26.5 27.1 28.4 27.8 29.3 ...
##  $ BD   : num  7 7.4 7.7 8.2 8.2 9.8 9.8 10.4 9.7 10.3 ...
```

```r
  #Variable 4 bis 8!

  # Berechnung der Hauptkomponenten
  pca <- prcomp(crabs[,4:8])
  summary(pca)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5
## Standard deviation     11.8619 1.13879 1.00013 0.36783 0.27913
## Proportion of Variance  0.9825 0.00906 0.00698 0.00094 0.00054
## Cumulative Proportion   0.9825 0.99153 0.99851 0.99946 1.00000
```

```r
  #Wir stellen die Daten in der 2. und 3. Hauptkomponente dar.
  plot(pca$x[,1],pca$x[,2],pch=19,col=crabs$sex)
```




```r
  #Female ist schwarz!!
```

d) Fügen Sie die Hauptkomponenten als weitere Variablen an den Datensatz Crabs an. Trainieren Sie mit dem erweiterten Datensatz wieder einen Klassifikationsbaum und stellen Sie ihn dar. Welche Variablen wurden als Splitvariaben verwendet? Stellen Sie die Daten im Streudiagramm der beiden häufigsten benutzen Split-Variablen dar. Quantifizieren Sie die Performance auf dem Trainingsdatensatz durch eine Kofusionsmatrix. Was fällt im Vergleich zu den Ergebnissen aus den letzten Teilaufgaben auf? Können Sie sich die Unterschiede erklären?

```r
  #Jetzt fuegen wir die Hauptkomponenten an den Datensatz an
dim(crabs)
```

```
## [1] 200   8
```

```r
crabs2 <- cbind(crabs,pca$x)
dim(crabs2)
```

```
## [1] 200  13
```

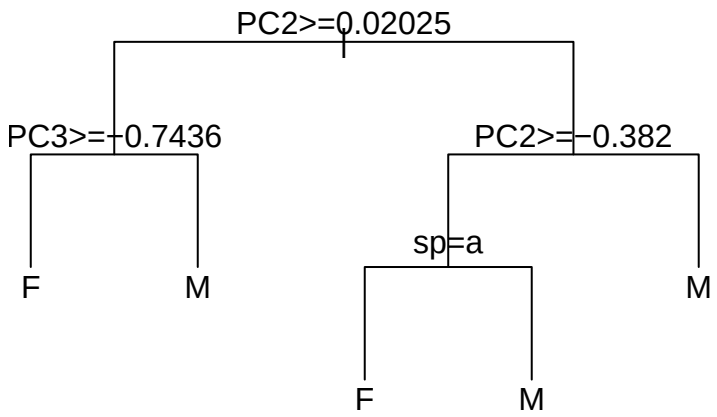
```r
#Wir haben 5 Spalten hinzugefuegt
names(crabs2)
```

```
##  [1] "sp"    "sex"   "index" "FL"    "RW"    "CL"    "CW"    "BD"
##  [9] "PC1"   "PC2"   "PC3"   "PC4"   "PC5"
```

```
#Klassifikationsbaum mit dem gesamten Datensatz
t2 = rpart(sex ~ .,data=crabs2)
t2
```

```
## n= 200
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 200 100 F (0.50000000 0.50000000)
##    2) PC2>=0.0202501 97   9 F (0.90721649 0.09278351)
##      4) PC3>=-0.7435508 85   1 F (0.98823529 0.01176471) *
##      5) PC3< -0.7435508 12   4 M (0.33333333 0.66666667) *
##    3) PC2< 0.0202501 103  12 M (0.11650485 0.88349515)
##      6) PC2>=-0.3820408 31  12 M (0.38709677 0.61290323)
##       12) sp=B 16   4 F (0.75000000 0.25000000) *
##       13) sp=O 15   0 M (0.00000000 1.00000000) *
##      7) PC2< -0.3820408 72   0 M (0.00000000 1.00000000) *
```

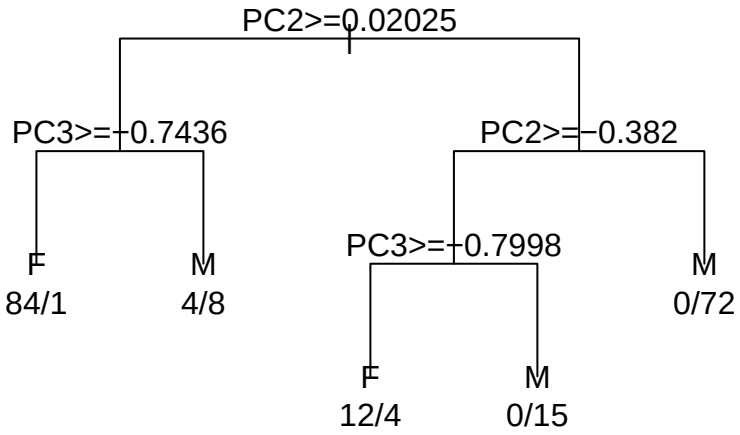
```
plot(t2,margin=0.2, uniform=TRUE)
text(t2)
```



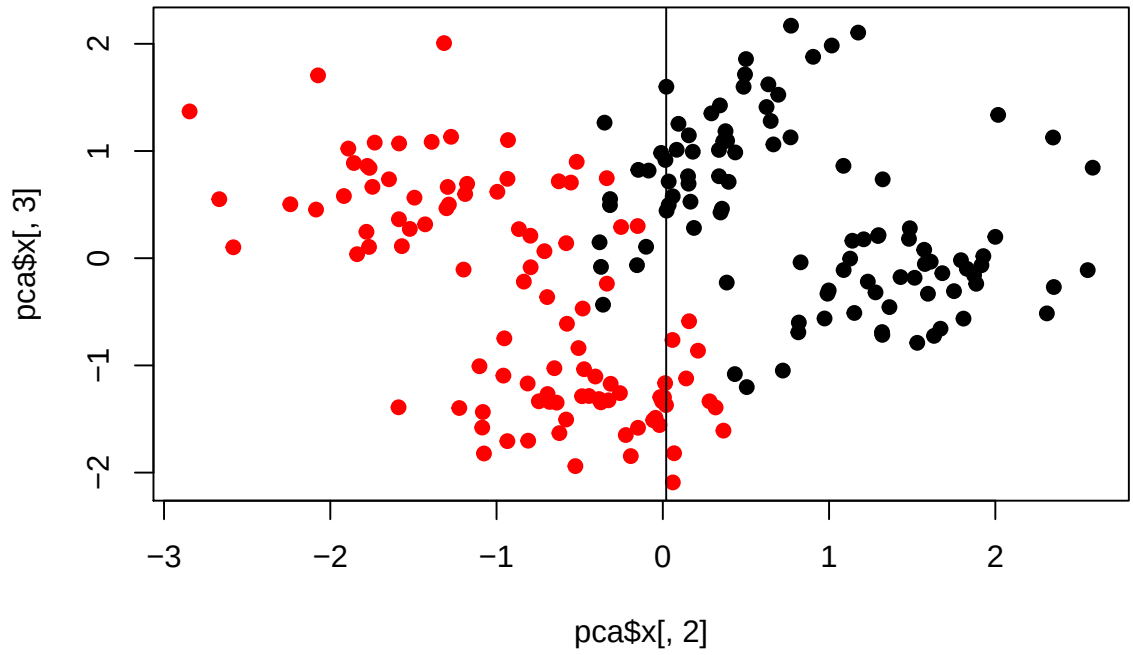

```
#Die Variablen PC2, PC3 und sp werden verwendet.

#Wie sieht die Performance auf Trainingsdaten aus?
confusion(object=predict(t2,type="class"), true=as.factor(crabs2$sex))
```

```
##           true
## predicted  F  M
##         F 96  5
##         M  4 95
```

```
#Das Ergebnis ist noch besser geworden!

# Klassifikationsbaum nur auf den Hauptkomponenten
t3 <- rpart(sex ~ PC1+PC2+PC3+PC4+PC5,data=crabs2)

plot(t3, margin=0.2, uniform=TRUE)
text(t3,use.n=TRUE)
```




```
# Es werden nur die 2. und 3. Hauptkomponente als Split-
# variablen verwendet, da diese die 4 Gruppen am besten trennen.

#Wir stellen die Daten in der 2. und 3. Hauptkomponente dar.
plot(pca$x[,2],pca$x[,3],pch=19,col=crabs$sex)
#Female ist schwarz!!
#Wir zeichnen die Grenzen welche der Algorithmus gefunden hat ein.
abline(v=0.02025) #Erster Split PC2>=0.02025
```




```
# #Zweiter Split auf der rechten Seite PC3>=0.7436
# lines(x = c(0.02025, 3.5), y= c(-0.7436,-0.7436))
# #Erster Branch ist eingetragen
#
# #Dritter Split PC2>=0.382
# lines(x = c(-0.382,-0.382), y = c(-3,3))
# #Vierter Split PC3>=0.7998
# lines(x = c(-0.382, 0.02025), y= c(-0.7998,-0.7998))

# Konfusionsmatrix f?r Klassifikation der Trainingsdaten
```

```r
table(predict(t3,type="class"),crabs$sex)
```

```
##
##      F  M
##   F 96  5
##   M  4 95
```

```r
# Klassifikation ist bedeutend besser.
confusion(predict(t2,type="class"),crabs$sex)
```

```
##          true
## predicted  F  M
##         F 96  5
##         M  4 95
```

```r
#Die Klassifikation ist besser als ohne die Hauptkomponenten!!
```

Aufgabe 2

Manuelle Bagging Aufgabe

In this exercise we build classification trees from Boston dataset from MASS package and bag (bootstrap aggregate) them manually.

First, load the packages:

```r
library(rpart)
library(MASS) # For the data set
```

a) Make a new binary variable from crim variable, which is `TRUE` if an observation of crim is greater than the median of crim, and `FALSE` otherwise.

```r
# Making binary variable
Boston$crim = Boston$crim > median(Boston$crim)
```

b) Now split the data into training set (75%) and test set (25%) randomly. Hinweis: don't forget to set the seed for reproducible results.

```r
# Splitting in the train and testset
set.seed(1)
index.train = sample(nrow(Boston), size = floor(nrow(Boston)*3/4))
data.train = Boston[index.train, ]
data.test = Boston[- index.train, ]
```

c) Fit a classification tree on the training data and predict its performance on the test data. Finally, produce a cross-table of correcly and incorreclty classified samples. Hinweis:

```r
# Fitting a regular tree
fit = rpart(as.factor(crim)  ~ ., data = data.train)
pred = predict(fit, data.test, type = "class")
table( pred, data.test$crim )
```

```
##
## pred    FALSE TRUE
##   FALSE    62    4
##   TRUE      3   58
```

d) Now let's procede to bootstrap aggregation (bagging). As the name suggests, you would need to aggregate the predictions on various bootstrap samples from the data. You need to grow 100 trees and loop over them by taking a different bootstrap sample of observations every time, then fitting a classification tree to it, and finally predicting its performance on the test set.

Hinweise:

```
#=========
# Bagging
n.trees = 100
preds = rep(0, nrow(data.test))
for (j in 1:n.trees ){
  # Doing the bootstrap
  index.train = sample( nrow(data.train), size = nrow(data.train), replace = TRUE )

  # Fitting to the training data
  fit = rpart( as.factor(crim) ~ ., data = data.train[index.train,])
  # Predict the test set
  tree.fit.test = predict(fit, data.test, type='class')
  preds = preds + ifelse(tree.fit.test == TRUE, 1, -1)
}
```

e) Now make a cross table of the prediction results.

```
table(preds > 0, data.test$crim > 0)
```

```
##
##         FALSE TRUE
##   FALSE    63    6
##   TRUE      2   56
```

So the naive error rate is $(2+6)/(63+6+2+56) = 0.063$

f) Optional: Verify your results with random forest. How close are they? Hinweise:

```
library(randomForest)
fit = randomForest(as.factor(crim) ~ ., data=data.train, ntree=100, mtry = ncol(data.train) -1 )
fit
```

```
##
## Call:
##  randomForest(formula = as.factor(crim) ~ ., data = data.train,      ntree = 100, mtry = ncol(dat
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 4.22%
## Confusion matrix:
##       FALSE TRUE class.error
## FALSE   181    7  0.03723404
## TRUE      9  182  0.04712042
```

```
table(predict(fit, data.test), data.test$crim)
```

```
##
##         FALSE TRUE
##   FALSE    63    5
```

```
##    TRUE       2    57
```