Introduction to Machine Learning - Lab 1

Gustav Sternelöv

Tuesday, October 27, 2015

Assignment 1

1.1 - 1.2

To get started with the analysis of the e-mail data file the data set first is divided into a train and a test data set. Each data set contains 50 percent each of the original data set.

In the next step the distance matrix is computed and used to implement a k-nearest-neighbor function, called *knearest*. The distance matrix is given by 1 - c(X,Y) where c(X,Y) is defined as:

$$\frac{X^T Y}{\sqrt{\sum_{i=1} X_i^2}\sqrt{\sum_{i=1} Y_i^2}}$$

The code that creates the knearest function can be seen in the appendix "R-Code" at the end of the report.

1.3

The implemented k-nearest-neighbors function is then used to classify the e-mails in both the training and test data. The classification principle that decides how to classify each e-mail is stated as following:
$\hat{y} = 1$ *if* p (Y = 1|X) > 0.5, *otherwise* $\hat{y} = 0$
With K, the number of neighbors, set to 5 the confusion matrix generated from classifying the training data looks like this:

```
##
##         0    1
##    0 1324   87
##    1   80  809
```

The misclassification rate for the training data then is 0.0726087. It can be seen that it is approximately the same amount of non-spam and spam e-mails that has been misclassificated, but that a higher rate of the spam e-mails has been misclassificated.
For the classification of the e-mails in the test data set this is the obtained confusion matrix:

```
##
##         0    1
##    0 1263  114
##    1  108  816
```

For the test data the misclassification rate is 0.0964798, and the same pattern as for the training data regarding which e-mails that has been misclassificated can be seen for the test data.
Unsurprisingly the misclassification rate is higher for the test data than the training data. Since the training data is used to make the classification of the e-mails it should work better for the training data, although the low rate for the test data implies that the model is quite good for the whole data set.

1.4

With K set to 1 instead of 5 the classification of the training data improves, and the misclassification rate is now 0.0143478. A summary over the classifications can be seen in the confusion matrix below.

```
##
##         0    1
##    0 1407    4
##    1   29  860
```

For the test data set the opposite result is obtained. The misclassification rate when K is set to 1 is 0.1038679 which is a little bit worse than before, but still the model seem to work pretty good.

```
##
##         0    1
##    0 1262  115
##    1  124  800
```

1.5

When the function *kknn* from the package **kknn** is used to classify the e-mails with the k-nearest-neighbor algorithm, the follwing results are obtained. K is set to 5 and for the test data the confusion matrix looks like this:

```
##
##         0    1
##    0 1262  115
##    1  125  799
```

This gives a misclassification rate of 0.1043025, which is the highest misclassification rate received so far for the test data set. It therefore seems like the *knearest* function performs slightly better than the *kknn* function.

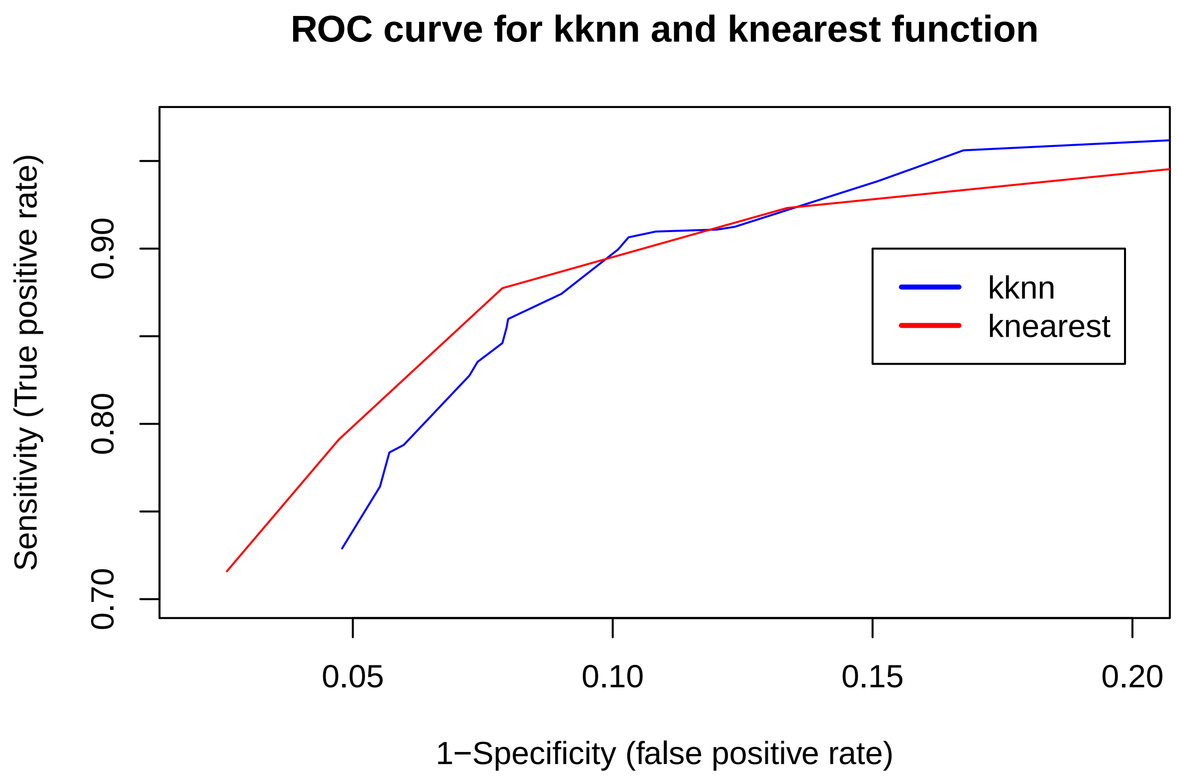
1.6

The sensitivity and specificity is computed for both the *knearest* and the *kknn* function. For each classification for the respective methods the sensitivity and specificity is computed according to the following formulas.

$$Sensitivity = \frac{TruePositive}{AllPositive}$$

$$Specificity = \frac{TrueNegative}{AllNegative}$$

The ROC curve for both models is then fitted and plotted in the graph below.



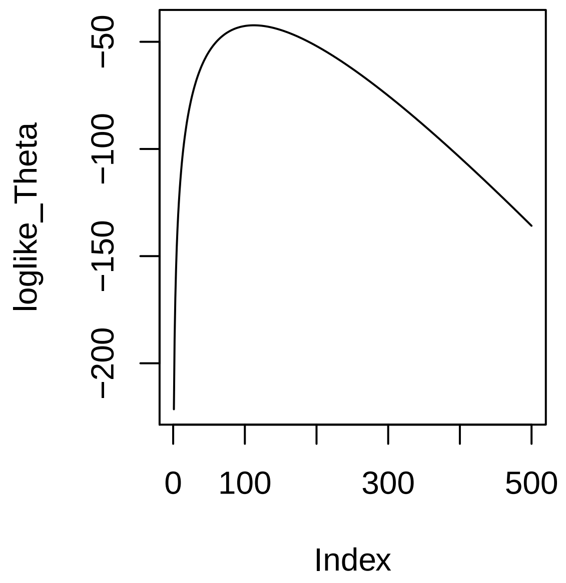
Some conclusions that can be drawn out of the ROC curves is that both models seem to be pretty good classifiers of spam and non-spam e-mails. There are no major differences between the models, the respective lines follow each other rather well. When the decision value for the classification principle is low the *knearest* function performs better and for high values the *kknn* function performs a little bit better.
Looking more closely to the respective lines it can be seen that the red one, *knearest*, only change direction when the decision value reaches 0.2, 0.4, 0.6 and so on. That is related to how the function works when it classifies the e-mails, and apparently there is some difference in how the functions use the information about the five nearest neighbors.

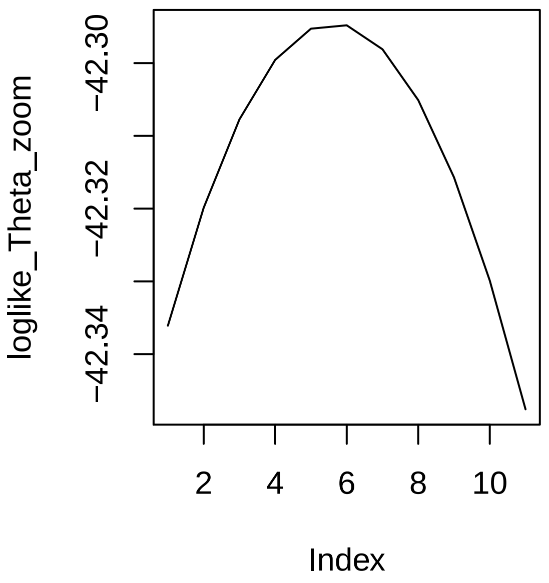
Assignment 2

In the second assignment a specific type of machine and information about its lifetime is analysed. The main purpose of the analysis is to get more information about the underlying process for the lifetimes of the machines.

2.2

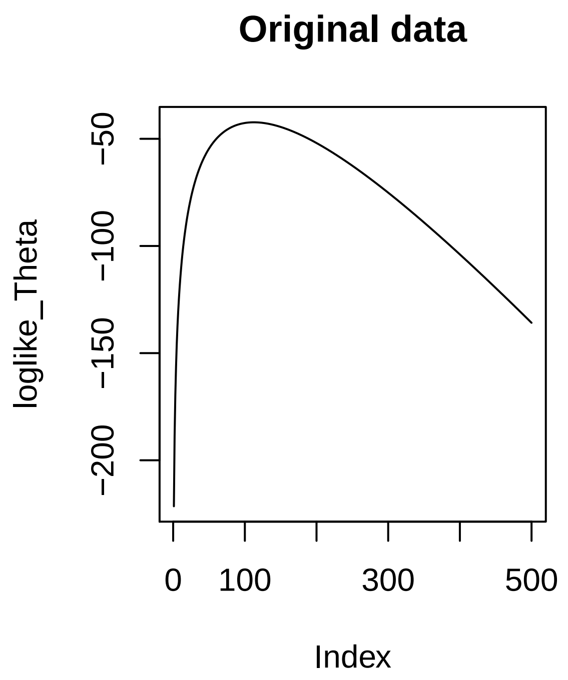
Since the probability model for x is equal to $\Theta e^{\theta x}$, the x-values in the data set follows an exponential distribution. To investigate the dependence of the log-likelihood on Θ, the log-likelihood is computed for a range of values for Θ. A plot that shows this dependence can be seen below to the left where the values for Θ goes from 0.01 to 5 by steps of 0.01.
To find the maximum likelihhod value for Θ the plot on the right side is used. It is zoomed in and only covers the range of Θ going from 1.08 to 1.18 by 0.01. By the look of this plot the maximum likelohood value of Θ seem to be 1.13.

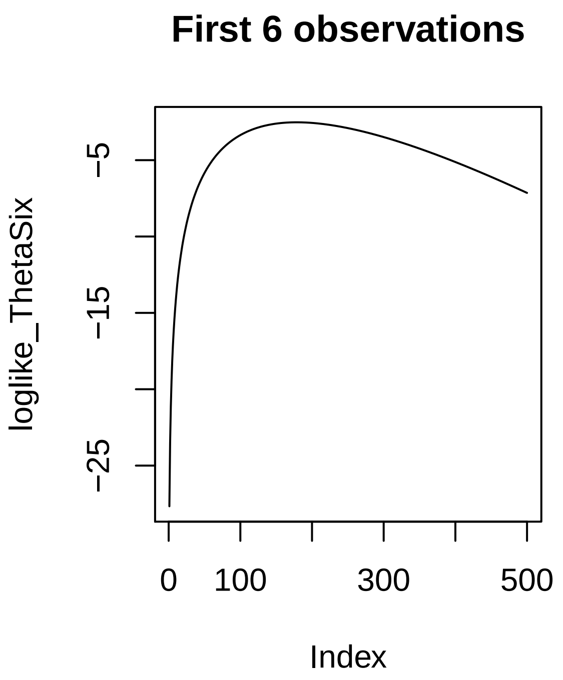




2.3

In the third step of assignment 2 the former step is repeated, but this time only for the first 6 observations in the data set. The dependence of log likelihood on θ for the case when only the first 6 observations are used and when the whole data set is used is plotted by the graphs below.

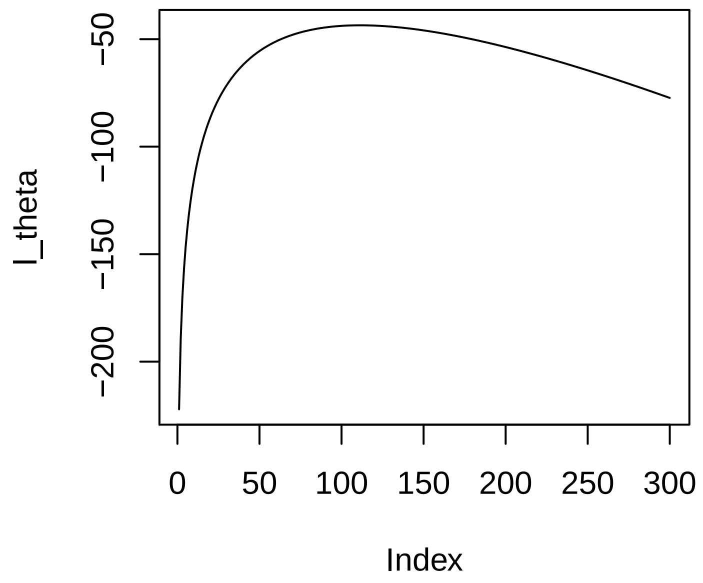




When comparing the log-likelihood curves it is evident that the estimation of the Θ value is more reliable when more observations are available. The curve recieved when only 6 observations are used is more smooth which makes it harder to see what the optimal maximum likelihood value of Θ is. A rough interpretation of the the plot gives that the maximum likelihood value of Θ lies somewhere around 1.6-1.9. By looking at the exakt value for the log likelihood it is given that the optimal value of Θ is 1.79. This value is far from the optimal value given when the whole data set is used, 1.13, and therefore is a good illustration to why the maximum likelihood solution is more unreliable for small data sets.

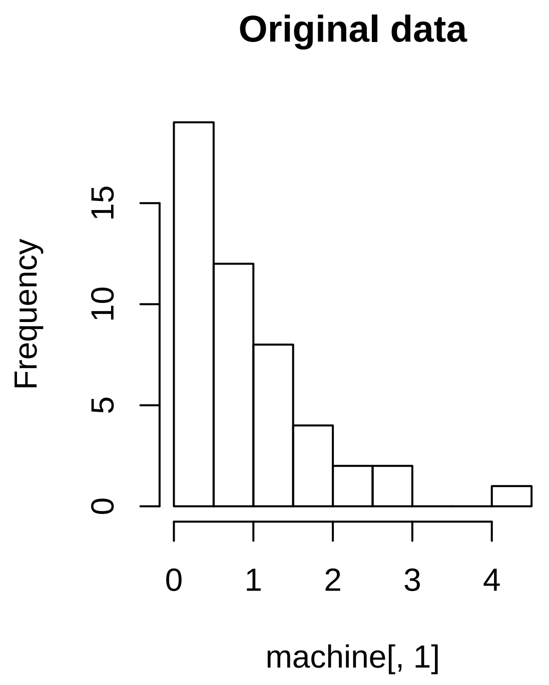
2.4

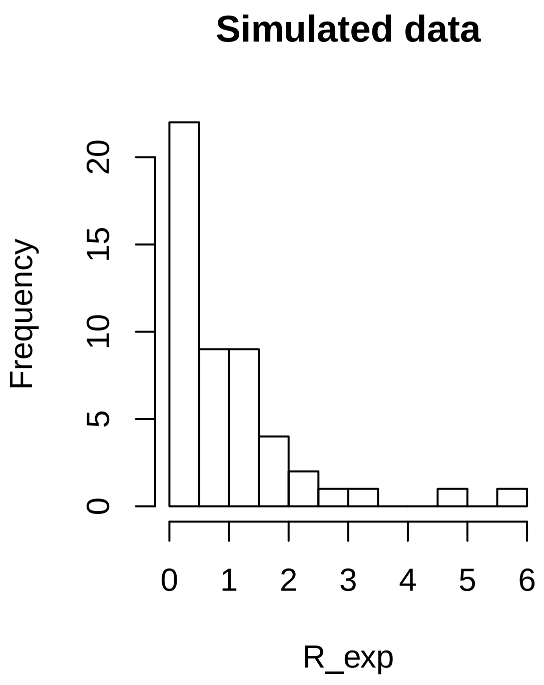
The function computed in this step, $l(\Theta) = \log(\Theta^n * e^{\Theta \sum x_i} * 0.5\ e^{-0.5\theta})$, measures the log likelihood for the posterior distribution of Θ. In the plot below the dependence of l(Θ) on Θ is illustrated, for the range of Θ values from 0.01 to 3 by steps of 0.01. The optimal value of Θ is found to be 1.11, which is close to the result obtained for Θ in 2.2, 1.13.
The difference between the optimal values in 2.2 and 2.4 for Θ is a result of the prior distribution used in 2.4. The plot below and the plot in 2.2 are almost identical, with the difference that the plot in this step is slightly shifted to the left. This is the effect of the prior distribution for Θ.



2.5

In step 5, 50 new observations are generated from the exponential distribution with the Θ set to the optimal value found in step 2, 1.13. A comparsion between the histograms for the original data and the new data can be seen below.





In both histograms a somewhat decaying pattern can be seen, which is typical for values following a exponential distribution. In the original data the frequency decays a bit slower, but in general the data sets seem to be rather similar distributed. Since both the original data and the simulated data has the same "true" value for the Θ parameter this also was the expected result.

Appendix - R-code

```r
# 1
dat <- read.csv("spambase.csv", sep=";", header = TRUE)
dat <- as.matrix(dat)

n=dim(dat)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=dat[id,]
test=dat[-id,]

# 1.2
knearest <- function(data, K,p=0.5 ,newdata){
  Predict <- 0
  ProbOne <- 0
  ProbZero <- 0

  x <- as.matrix(data[,1:48])
  y <- as.matrix(newdata[,1:48])
  x_transformed <- t(apply(x, 1, function(x) x / sqrt(sum(x^2))))
  x_transformed[is.na(x_transformed)] <- 0
  y_transformed <- t(apply(y, 1, function(y) y / sqrt(sum(y^2))))
  y_transformed[is.na(y_transformed)] <- 0

  numerator <- (x_transformed) %*% t(y_transformed)
  d_xy <- data.frame(1 - numerator)

  for(i in 1:length(d_xy)){
    LabelsTrain <- data[order(d_xy[,i]), 49][1:K]
    ProbOne[i] <- mean(LabelsTrain)
    ProbZero[i] <- 1-ProbOne[i]
    if(ProbOne[i] > p){
      Predict[i] = 1
    }else{
      Predict[i] = 0
    }
  }
newdata <- data.frame(newdata)
newdata$predict <- Predict
newdata$prob_one <- ProbOne
newdata$prob_zero <- ProbZero
return(newdata)
}

# 1.3
misc_test <- knearest(train, 5, 0.5, test)
table(misc_test$predict, misc_test$Spam)

misc_train <- knearest(train, 5, 0.5, train)
table(misc_train$predict, misc_train$Spam)

# 1.4
misc_test <- knearest(train, 1, 0.5, test)
table(misc_test$predict, misc_test$Spam)

misc_train <- knearest(train, 1, 0.5, train)
table(misc_train$predict, misc_train$Spam)

# 1.5
library(kknn)
train <- data.frame(train)
test <- data.frame(test)
test_kknn <- kknn(Spam~., train, test, k=5)
fitted_v <- test_kknn$fitted.values
for (i in 1:length(fitted_v)){
if(fitted_v[i] > 0.5 ){
  fitted_v[i] = 1
}else{
  fitted_v[i] = 0
}
}
conf_mat <- data.frame(cbind(fitted_v, test[,49]))
table(conf_mat[,2], conf_mat[,1])
```

```r
# 1.6
train <- data.frame(train)
test <- data.frame(test)
test_kknn <- kknn(Spam~., train, test, k=5)
fitted_v <- test_kknn$fitted.values
phis <- seq(0.05, 0.95, by=0.05)
h <- 0
classify <- matrix(ncol=19, nrow=2301)
predic <- 0

for (j in phis){
  for (i in 1:length(fitted_v)){
    if(fitted_v[i] > j ){
      predic[i] = 1
    }else{
      predic[i] = 0
    }
  }
  h <- h+1
  classify[,h] <- predic
}

# For knearest()
Knear_class <- matrix(ncol=19, nrow=2301)
h <- 0
for (j in phis){
  testish <- knearest(train, 5, p=j, test)
  h <- h+1
  Knear_class[,h] <- testish$predict
}

# Computing sensitivity and specificity
# Adding true labels to kknn and knearest matrices
classify <- data.frame(classify)
classify$Spam <- test[,49]
Knear_class <- data.frame(Knear_class)
Knear_class$Spam <- test[,49]

sensitivity_kknn <- 0
specificity_kknn <- 0
sensitivity_knearest <- 0
specificity_knearest <- 0
for (i in 1:19){
  table_kknn <- table(classify[,20], classify[,i])
  sensitivity_kknn[i] <- table_kknn[2,2] /sum(table_kknn[,2])
  specificity_kknn[i] <- table_kknn[1,1] /sum(table_kknn[,1])
  table_knearest <- table(Knear_class[,20], Knear_class[,i])
  sensitivity_knearest[i] <- table_knearest[2,2] /sum(table_knearest[,2])
  specificity_knearest[i] <- table_knearest[1,1] /sum(table_knearest[,1])
}
kknn_ROC <- data.frame(cbind(1 - specificity_kknn, sensitivity_kknn))
knearest_ROC <- data.frame(cbind(1 - specificity_knearest, sensitivity_knearest))
```

```r
plot(kknn_ROC, type="l", col="blue", xlim=c(0.02,0.2), ylim=c(0.7,0.97),
     xlab="1-Specificity (false positive rate)", ylab="Sensitivity (True positive rate)",
     main="ROC curve for kknn and knearest function")
lines(knearest_ROC, type="l", col="red")
legend(0.15,0.9,c("kknn","knearest"),
lty=c(1,1),
lwd=c(2.5,2.5),col=c("blue","red"))

# 2
machine <- read.csv("machines.csv", sep=";", header = TRUE)
# 2.2
n <- length(machine[,1])
theta <- seq(0.01, 5, by=0.01)

loglike_Theta <- n * log(theta) - theta *sum(machine[,1])
plot(loglike_Theta, type="l")

theta[which(loglike_Theta==max(loglike_Theta))]

theta_zoom <- seq(1.08, 1.18, by=0.01)

loglike_Theta_zoom <- n * log(theta_zoom) - theta_zoom *sum(machine[,1])
plot(loglike_Theta_zoom, type="l")
# 2.3
machine_six <- data.frame(machine[1:6, 1])
n_six <- length(machine_six[,1])
theta <- seq(0.01, 5, by=0.01)
loglike_ThetaSix <- n_six * log(theta) + -theta *sum(machine_six[,1])
theta[which(loglike_ThetaSix==max(loglike_ThetaSix))]

par(mfrow=c(1,2))
plot(loglike_Theta, type="l", main = "Original data")
plot(loglike_ThetaSix, type="l", main = "First 6 observations")
par(mfrow=c(1,1))

# 2.4
lambda <- 0.5
theta <- seq(0.01, 3, by=0.01)
x <- machine[,1]

l_theta <- log(theta^n * exp(-theta*sum(x)) * (0.5*exp(-0.5*theta)))
plot(l_theta, type="l")
theta[which(l_theta==max(l_theta))]

# 2.5
set.seed(12345)
R_exp <-rexp(50, 1.13)

par(mfrow=c(1,2))
hist(machine[,1],breaks=8, main="Original data")
hist(R_exp, breaks=12, main = "Simulated data")
par(mfrow=c(1,1))
```