Introduction to Machine Learning

TDDE01 – Lab1

Author:
Christoffer Eduards - chred146

I. Introduction

This report is part 1 of the laborations in the course TDDE01 – Introduction to Machine Learning. This report consists of three mandatory exercises. The code required for solving these assignments is attached to this rapport and can be viewed under Appendix.

II. Assignment 1

Spam classification with nearest neigbors
This exercise is based around the dataset "spambase" which consists of information about 2740 e-mails. These e-mails have been classified as either spam (1) or not spam (0). Additional information about each e-mail is the frequency of various words.

1.1
In the first part of the exercise the data divided and stored into the sets train and test by the use of the given code.

1.2
In this task the confusion tables and misclassification rates are calculated for the train and test data. This is done by using logistic regression and the following classification principle:

$$\hat{Y} = 1 \; if \; p(Y = 1|X) > 0.5, otherwise \; \hat{Y} = 0$$

The confusion matrix and misclassification rate for **test data** is shown below:

```
     Prediction
Truth   0   1
    0 791 146
    1  97 336
Missclassification rate = 0.1773723
```

The confusion matrix and misclassification rate for **train data** is shown below:

```
     Prediction
Truth   0   1
    0 803 142
    1  81 344
Missclassification rate = 0.1627737
```

The result shows that the misclassification rate is slightly higher for the test data which is correct since the model is biased towards the training data since this was used to create it.

1.3
This exercise is the same as the previous one except for the classification principle. Here, the following classification principle is used:

$$\hat{Y} = 1 \; if \; p(Y = 1|X) > 0.9, otherwise \; \hat{Y} = 0$$

The confusion matrix and misclassification rate for **test data** can be seen below:

```
     Prediction
Truth   0   1
    0 936   1
    1 427   6
Missclassification rate = 0.3124088
```

For the **train data**, the confusion matrix and misclassification rate are:

```
    Prediction
Truth   0   1
    0 944   1
    1 419   6
Missclassification rate = 0.3065693
```

By changing the classification rule almost all predictions will be classified as 0 which explains the appearance for the confusion matrices. As in the previous assignment the misclassification rate is slightly higher for test data and can be explained in the same way as in the previous exercise.

1.4
In this part of the exercise the standard classifier kknn() with K = 30 is used to calculate the misclassification rates for training and testing data. This is presented below:

```
Test data:
Missclassification rate = 0.329927
Train data:
Missclassification rate = 0.1722628
```

As can be seen by the misclassification rates the kknn model provides a significantly lower rate for the training data. But the misclassification rate for test data is actually worse than by using logistic regression.

1.5
Here, we use K = 1 instead of K = 30. This results in the following change in the misclassification rates for training and testing data:

```
Test data:
Missclassification rate = 0.3459854
Train data:
Missclassification rate = 0
```

By using K = 1 the model gets extremely overfitted. This results in the model fitting the training data perfectly while applied on other data being far from optimal.

III. Assignment 2

Interference about lifetime of machines

This exercise is based around the dataset “machines” which contains information about the lifetime of certain machines. The only variable in this data is the lifetime measured in years.

2.1

Imports the data into the dataset “lifetimes”.

2.2

In this exercise we assume the probability model:

$$p(x|\theta) = \theta e^{-\theta x}, \qquad \text{x} = \text{Length}$$

The model is exponential, therefore it is an exponential distribution. The first exercise is to compute a log-likelihood function for a given θ and a given data vector x. The following is the result when the curve showing the dependence of log-likelihood on θ is plotted:

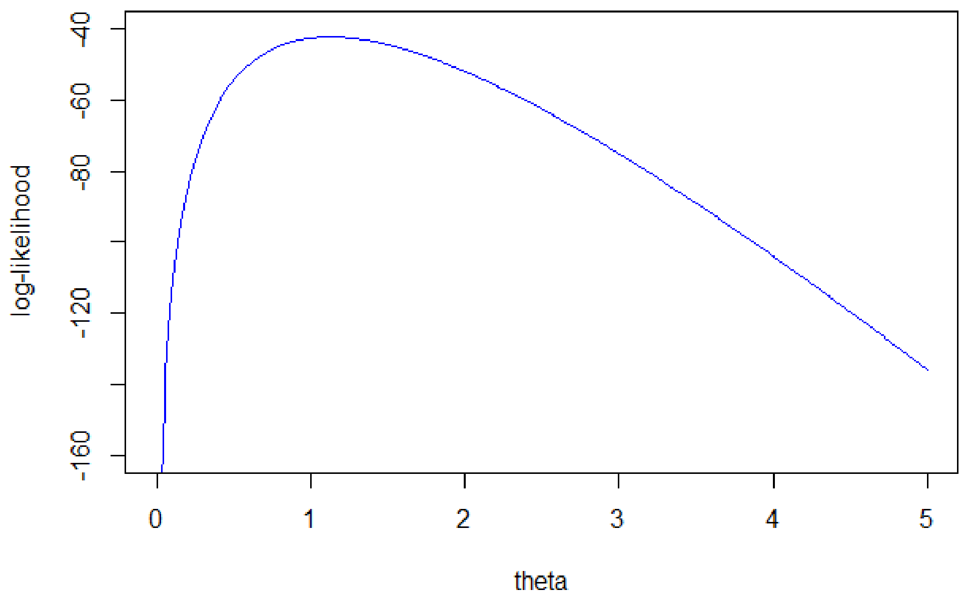

Figure 1: Log-likelihood on θ all data

According to the plot, the θ value that maximises the log-likelihood function is $\theta = 1.13$

2.3

In this exercise only the first six values from the data is being used to compute the log-likelihood function. This function paired with the plot from 2.2 is shown in Figure 2.

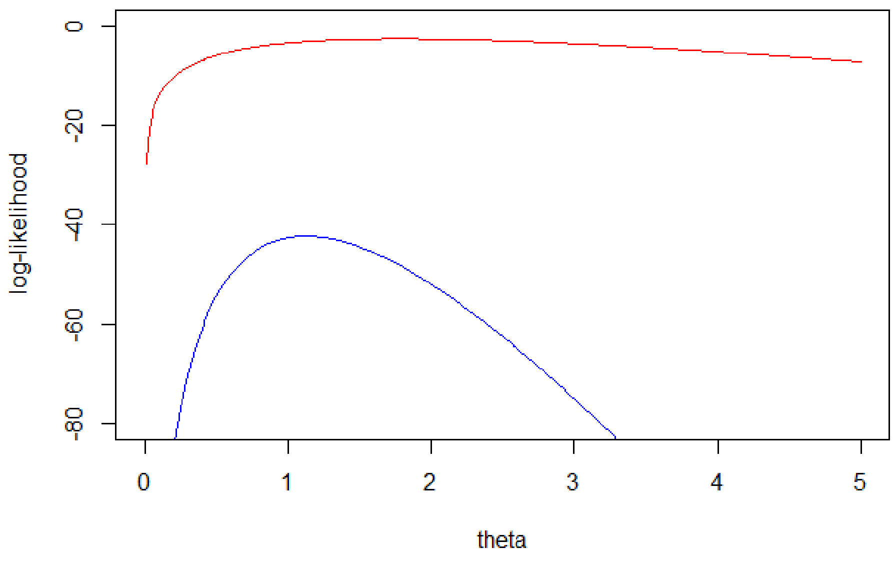

Figure 2: Log-likelihood on θ all data vs first six

The red curve that shows the result when only using the first six lifetimes gives us that the optimal $\theta = 1.79$ which differs quite much from the previous one (blue curve) where $\theta = 1.13$. This tells us that a log-likelihood model is dependent on having sufficient amount of data that represents the distribution correctly. We can see from the plots that the blue curve with all data used has a more defined peak in comparison to the red one. This results in that a small change in that a small change in theta for the red curve does not worsen the likelihood as much as it does when all data is used. The conclusion from this is that the reliability decreases with less data.

2.4

In this step of the assignment a Bayesian model is used:

$$p(x|\theta) = \theta e^{-\theta x}, \qquad \text{x = Length}$$

with the prior:

$$p(\theta) = \lambda e^{-\lambda\theta}, \qquad \lambda = 10$$

A function that computes: $l(\theta) = log(p(x|\theta)p(\theta)$ and depends on θ is plotted below. This is a function that is proportional to the posterior probability. It is scaled and compared with the plot from 2.2 below:

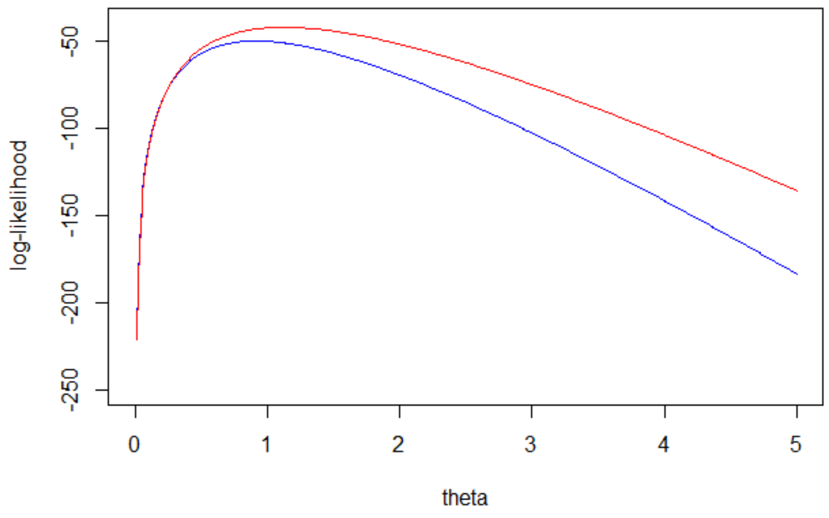

Figure 3: Bayesian model vs frequentist approach

The optimal theta when using the Bayesian model (blue curve) is $\theta = 0.91$. We can see that the curve using the Bayesian model (blue) is fairly similar to the frequentist approach. The main part that makes the function differ is that the Bayesian one also depends on the prior with the parameter λ which we in this exercise set to 10. What the posterior does is that it assumes the expected optimal log-likelihood for the data which in this case is E[λ] = 1 / λ = 0,1. In fact this value is approximately equal to 1. This shows us that if a large dataset is being used the posterior have little impact on the final result.

2.5

The final step of the exercise is to use theta value found in 1.2 and generate 50 new observations by using an exponential model. The histogram for the new data can be seen below:

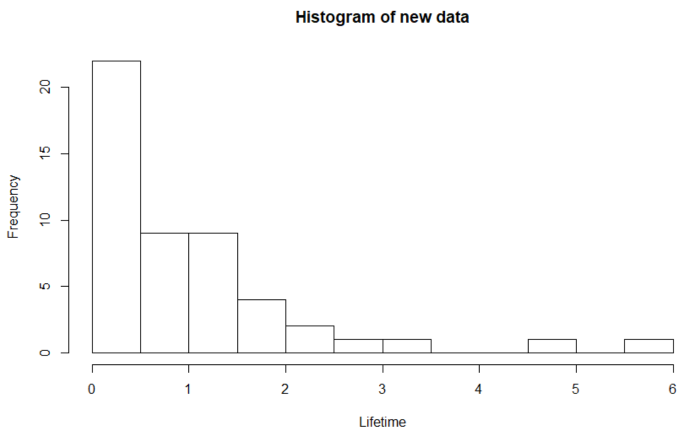

Figure 4: Histogram of new data

The histogram with the original data is presented below:

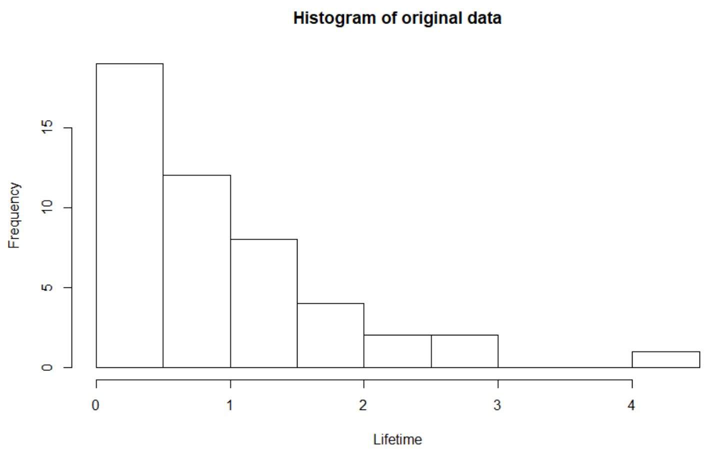

Figure 5: Histogram of original data

The histograms have great similarities which tells us that the theta value used to generate the new values is a good estimation.

IV. Assignment 4

Linear regression and regularization

In this exercise the task is to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat. To solve this exercise data from the file “tecator” is provided. It contains for each meat sample 100 channel spectrum of absorbance records and the levels of moisture, fat and protein.

4.1

In the first part of the assignment the data is imported, and moisture vs protein is plotted. The result from this can be seen below:

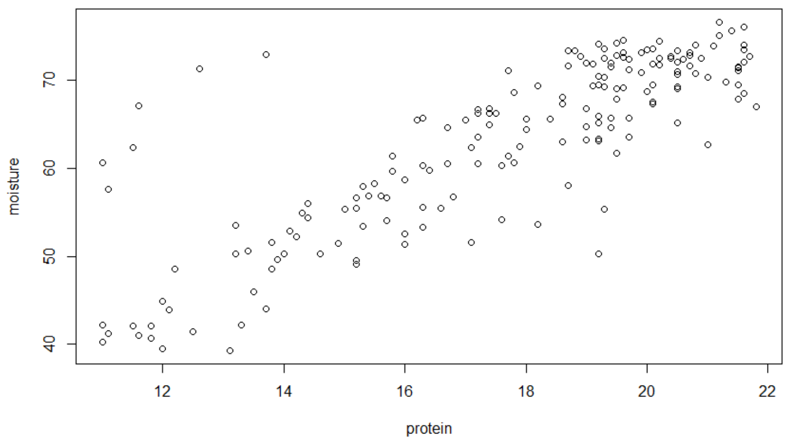

Figure 6: Moisture vs Protein

The data in the figure looks to have the possibility to be fitted with a linear model effectively since the data is located around a line and the error seems to be normally distributed. The problematic part with a linear model in this case could be the outliers in the top left corner in the figure that differs significantly from the rest of the data.

4.2

In the second part of the assignment a probabilistic model has been constructed that describes the moisture M_i which is normally distributed, as a polynomial function of Protein. The model can be seen below:

$$Y \sim N(w_0 + w_1x + w_2x^2 + \ldots + w_nx^n, \sigma^2) = N\left(\sum_{i=0}^{n} w_ix^i, \sigma^2\right), \qquad X = Protein, Y = Moisture$$

An MSE criterion makes it possible to compare different mean-squared errors in relation to different parameters such as the polynomial degree to decide on the best model.

4.3

In the third part of the assignment the data is divided into training and validation sets (50%/50%). Six models ($i = 1 \ldots 6$) are fitted to each data set. The mean-squared error in relation to the polynomial degree can be shown in the figure below. The blue line represents training data and the red line represents the validation data.

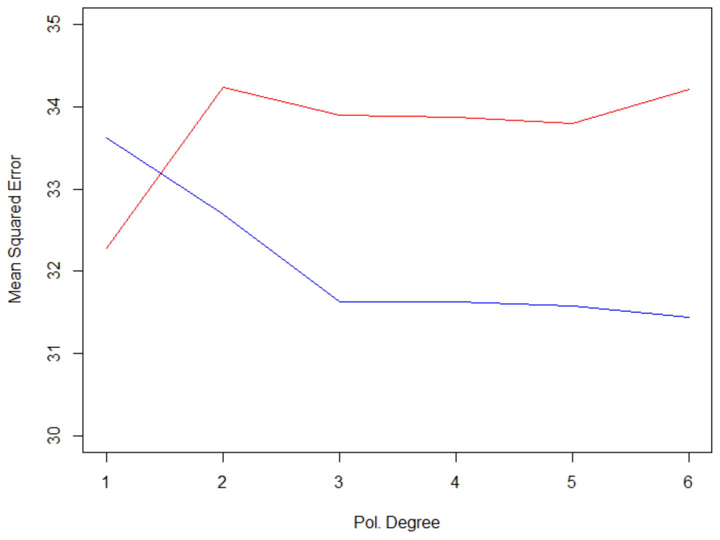

Figure 7: Mean-squared error vs pol. degree

According to the plot, the best model for the training data is when i = 6 and for the validation data the best model is when i = 1. This can be explained in terms of bias-variance trade off. Linear models (i = 1) tend to have a high bias but a low variance. For non-linear algorithms the case is often the opposite. Starting with the training data, we see that the mean-squared error decreases as i, or in general terms called the model complexity increases. This is because the model gets more and more overfitted on this data as the model complexity increases.
As i increases the model will have lower bias but higher variance. The effects of this can be seen on the validation data. We see that the linear estimation is the optimal one for the validation data. With i > 1 the mean-squared error is larger. It decreases slightly at between two and five since the bias decreases. What eventually happens at i = 5 is that the increasing variance play a larger part than the decreasing bias which makes the MSE grow.

4.4
In this task a variable selection of a linear model is performed using stepAIC. In the linear model, Fat is response and Channel1-Channel100 are predictors. The function returns 63 features. Therefore, 47 features did not improve the model to a statistically significant extent when they were being tested against the then current features.

4.5
In this exercise we use the same predictor and response variables and fit a Ridge regression model. The result is shown in the following figure.

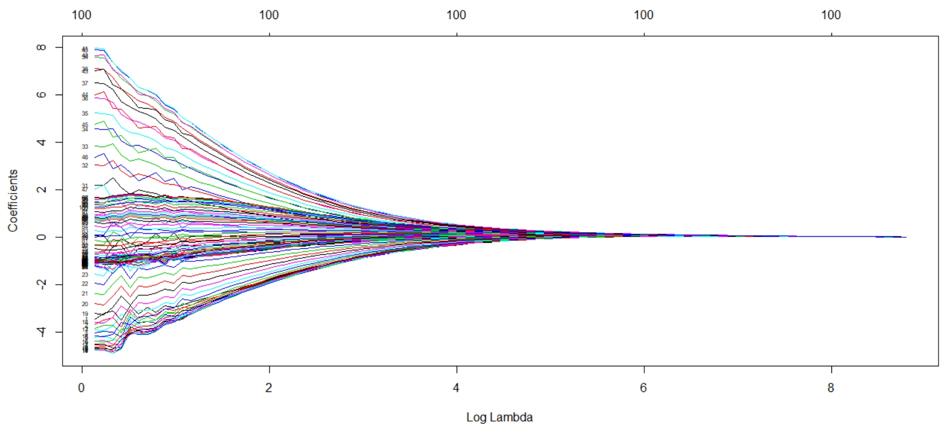

Figure 8: Ridge regression

The model shows us that as the penalty factor λ increases, all coefficients moves towards zero. This is due to the property of Ridge regression where the coefficients get shrunk quadratically.

4.6
In this task the same procedure as in the previous task is performed but for a Lasso regression model. The result can be seen below:

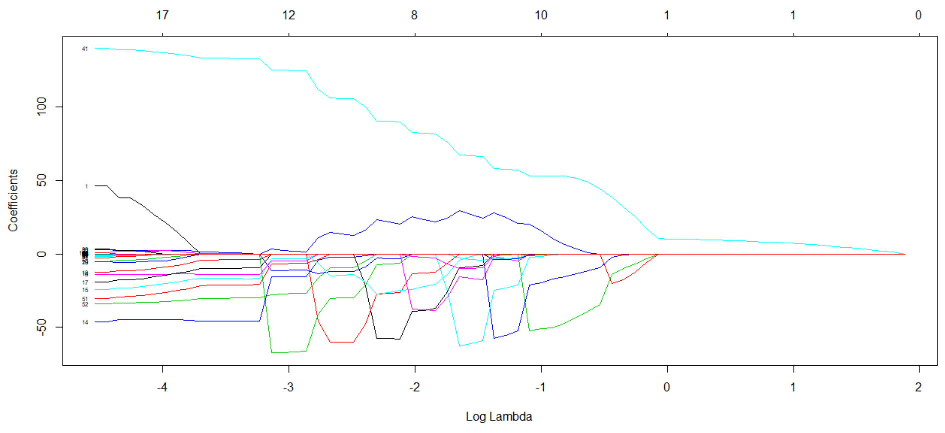

Figure 9: Lasso regression

For the lasso model some coefficients get penalized severely while other coefficients are untouched for different values of lambda. The difference from the Ridge model is that instead of using the squares, the model now utilises the absolute value to penalize different coefficients.

4.7
In this task cross-validation is used to find the optimal Lasso model. The plots below show us that the lowest possible lambda is preferable to minimize the mean-squared error. This value is lambda = 0. The interpretation of this is that all variables should be considered in the optimal solution. By choosing this value of lambda the model yields the same result as a least square model would give. Though, by looking at the figure below it is clear that there is no significant improvement in the mean-squared error with more than around 60 variables.

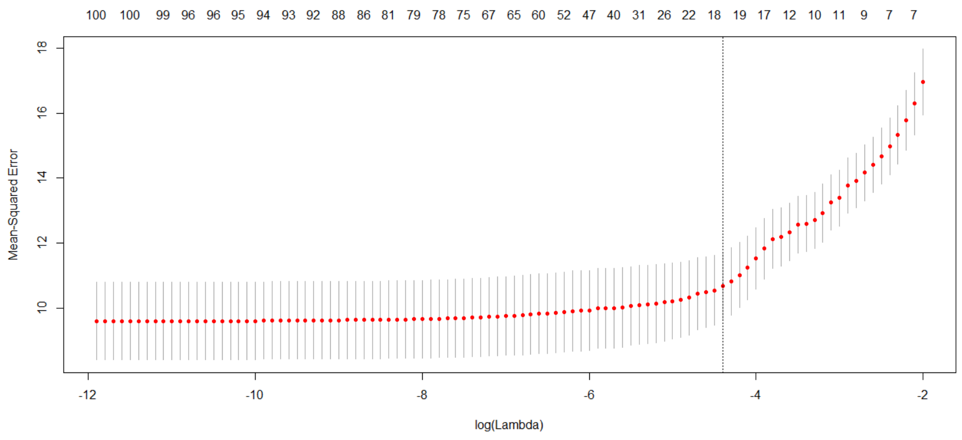

Figure 10: Variable selection Lasso

4.8

There are clear similarities between the variable selections in the stepAIC procedure and the Lasso model. The stepAIC returned 63 variables. A stepAIC procedure only includes features that result in a significant statistical improvement of the model. In the Lasso model there are no significant improvement at more than around 60 features which explains the result from the stepAIC variable selection.

V. Appendix

```
#Assignment 1:
#1.1
setwd("C:/Users/Christoffer Eduards/OneDrive/R/Lab1/Assignment1")
library("kknn")

#Loads data and divides it into train and test sets
Dataframe=read.csv2("spambase.csv")
n=dim(Dataframe)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=Dataframe[id,]
test=Dataframe[-id,]

#1.2
#Classify is a function that divides the predictions according to the
exercise
classify = function(pred, const){
  classified = c()
  for(i in 1:length(pred)){
    classified[i] = if(pred[i] > const) 1 else 0
  }
  return(classified)
}
#Fit the logistic regression model
glm.fit = glm(Spam~., data=train, family=binomial)

#Prediction with test data
preds_test = predict(glm.fit, test, type="response")

#Classifies according to exercise 1.2
classified5_test = classify(preds_test, 0.5)

#Confusion matrix test data:
conf5_test = table(Truth = test$Spam, classified5_test)

#Missclassification function
```

```
missclass_rate = function(preds, truth){
  return(mean(preds != truth))
}

#Missclassification rate test data
missclass5_test = missclass(classified5_test, test$Spam)

#Prediction for train data
preds_train = predict(glm.fit, train, type="response")

#Classified train data
classified5_train = classify(preds_train, 0.5)

#Confusion matrix train data
conf5_train = table(Truth = train$Spam, Prediction = classified5_train)

#Missclassification rate train data
missclass5_train = missclass(classified5_train, train$Spam)

#Printout for confusion matrixes and missclassification rates
print(conf5_test)
print(missclass5_test)
print(conf5_train)
print(missclass5_train)

#1.3
#Confusion matrixes and missclassification rates for test and train data
with classification according to exercise 1.3
classified9_test = classify(preds_test, 0.9)
conf9_test = table(Truth = test$Spam, Prediction = classified9_test)
missclass9_test = missclass(classified9_test, test$Spam)
classified9_train = classify(preds_train, 0.9)
conf9_train = table(Truth = train$Spam, Prediction = classified9_train)
missclass9_train = missclass(classified9_train, train$Spam)

#Printout
print(conf9_test)
print(conf9_train)
print(missclass9_train)
print(missclass9_test)

#1.4
#Fits model with package kknn() and K=30
fit.kknn30 = train.kknn(Spam~., train, 30)

#Make predictions with test data
preds30_test = predict(fit.kknn30, test)

#Classifies data as in 1.2
classified_test_kknn30 = classify(preds30_test, 0.5)
#Confusion matrix and missclassification rate for test data
conf_test_kknn30 = table(classified_test_kknn30, test$Spam)
missclass_test_kknn30 = missclass(classified_test_kknn30, test$Spam)

#Same steps as previously but for training data
preds30_train = predict(fit.kknn30, train)
classified_train_kknn30 = classify(preds30_train, 0.5)
conf_train_kknn30 = table(classified_train_kknn30, train$Spam)
missclass_train_kknn30 = missclass(classified_train_kknn30, train$Spam)

#Printout for confusion matrixes and missclassification rates
print(conf_test_kknn30)
print(conf_train_kknn30)
print(missclass_test_kknn30)
print(missclass_train_kknn30)

#1.5
#Same exercise as 1.4 but with K=1
fit.kknn1 = train.kknn(Spam~., train, 1)
preds1_test = predict(fit.kknn1, test)
classified_test_kknn1 = round(preds1_test)
conf_test_kknn1 = table(classified_test_kknn1, test$Spam)
missclass_test_kknn1 = missclass(classified_test_kknn1, test$Spam)
preds_train_kknn1 = predict(fit.kknn1, train)
classified_train_kknn1 = round(preds_train_kknn1)
```

```
conf_train_kknn1 = table(classified_train_kknn1, train$Spam)
missclass_train_kknn1 = missclass(classified_train_kknn1, train$Spam)
print(conf_test_kknn1)
print(conf_train_kknn1)
print(missclass_test_kknn1)
print(missclass_train_kknn1)

#Assignment 2:
#2.1
setwd("C:/Users/Christoffer Eduards/OneDrive/R/Lab1/Assignment2")
lifetimes=read.csv2("machines.csv")
thetas <- seq(0.0, 5.0, 0.01)
set.seed(12345)
#2.2
#The loglikelihood function which is used in the assignment
loglikelihood <- function(x, theta) {
  p = log(theta*exp(-theta*x))
  return(sum(p))
}

#Predict values with the loglikelihood function, and inputs lifetimes and
thetas
predsLog = sapply(thetas, loglikelihood, x=lifetimes)



#Plots the log-likelihood function on theta
plot(thetas, predsLog, xlab="theta",
     ylab = "log-likelihood",
     xlim=c(0,5), ylim=c(-160,-40), col="blue", type="l")

#Finds the maximum theta
max_predsLog = thetas[which(predsLog==max(predsLog))]
print(max_predsLog)

#2.3
#Chose only the first six values in the dataset then do the same as in 2.2
lifetimes_first_six = lifetimes[1:6,]
predsLog_firstSix = sapply(thetas, loglikelihood, x=lifetimes_first_six)
```

```
plot(thetas, predsLog, xlab="theta",
     ylab = "log-likelihood",
     xlim=c(0,5), ylim=c(-80,0), col="blue", type="l")
points(thetas, predsLog_firstSix, col="red", type="l")
max_predsLog_firstSix =
thetas[which(predsLog_firstSix==max(predsLog_firstSix))]
print(max_predsLog_firstSix)

#2.4
#The bayeshian model used in the assignment
bayesian_model = function(x, theta){
  p = prod(theta*exp(-theta*x))
  prior = 10*exp(-10*theta)
  l = log(p*prior)
  return(l)
}

predsBay = sapply(thetas, bayesian_model, x=lifetimes)
#Plots the predictions from 2.4 and 2.2
plot(thetas, predsBay, xlab="theta",
     ylab = "log-likelihood",
     xlim=c(0,5), ylim=c(-250,-40), col="blue", type="l")
points(thetas, predsLog, col="red", type="l")
#Theta that maximises the function
max_predsBay = thetas[which(predsBay==max(predsBay))]
print(max_predsBay)

#2.5
#Theta from 2.2
theta = max_predsLog
#Creates a histogram of the original data
hist(as.matrix(lifetimes),15, main= "Histogram of original data",
xlab="Lifetime")
#Creates a histogram of the new random generated data
hist(rexp(50,theta),15, main= "Histogram of new data", xlab="Lifetime")

#Assignment 4
setwd("C:/Users/Christoffer Eduards/OneDrive/R/Lab1/Assignment4")
```

```
data=read.csv2("tecator.csv")
data_matrix = as.matrix(data)
library("glmnet")
library("MASS")

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

#4.1
#Plot moisture vs protein
moisture=data_matrix[,ncol(data)]
protein=data_matrix[,ncol(data)-1]
plot(protein,moisture)

#4.3
#Create six different models each for training and validation data sets.
mse_train = numeric(6)
mse_validation = numeric(6)
for (i in 1:6) {
  lm_model = lm(Moisture ~ poly(Protein, i), data = train)
  predict_train = predict.lm(lm_model, train)
  predict_validation = predict.lm(lm_model, validation)
  mse_train[i] = mean((train$Moisture - predict_train) ^ 2)
  mse_validation[i] = mean((validation$Moisture - predict_validation) ^ 2)
}
x = seq(1, 6)
#Plot the MSE vs pol. degree
plot(
  x,
  mse_train,
  ylim = c(30, 35),
  col = "blue",
  ylab = "Mean Squared Error",
  xlab = "Pol. Degree",
  type = "l"
)
points(x, mse_validation, col = "red", type = "l")

#4.4
#Perform the stepAIC variable selection
lm_step = lm(data$Fat~., data[,2:102] )
step = stepAIC(lm_step)

#4.5
#Ridge regression
predictors = data[, 2:101]
response = data[, 102]
ridge = glmnet(as.matrix(predictors),
               response,
               alpha = 0,
               family = "gaussian")
plot(ridge, xvar = "lambda", label = TRUE)

#4.6
#Lasso regression
lasso = glmnet(as.matrix(predictors),
               response,
               alpha = 1,
               family = "gaussian")
plot(lasso, xvar = "lambda", label = TRUE)

#4.7
#Perform cross-validation to find optimal lasso-model on lambda
lambda = as.vector(exp(seq(from=-12, to=-2, by=0.1)))
lambda[1] = 0
cross_validation = cv.glmnet(as.matrix(predictors),
                             response,
                             alpha = 1,
                             family = "gaussian",
                             lambda = lambda)
cross_validation$lambda.min
```

```
plot(cross_validation)
coef(cross_validation, s = "lambda.min")
print(cross_validation$lambda.min)
```