Lab1

Assignment 1

1. The data is imported into R and divided into two sets; train and test data. This is because it is useful to use some data for creating a suitable model and then test this model on some other data. Naturally the model will perform better on the data used for training and therefore it is important to test the model on some other data.

2. The obtained confusion matrices with misclassification rates are shown below:

   Train data:

	FALSE	TRUE
0	804	127
1	93	346

   Misclass_train: 0.1605839

   Test data:

	FALSE	TRUE
0	808	143
1	92	327

   Misclass_test: 0.1715328

   As stated above it is reasonable that the model performs better on the train data compared to the test data. However, the fact that the miscalculations are similar indicates that the model performs similarly on two different data sets which is generally how you want your model to behave.

   By analyzing the confusion matrices, it is notable that the model is wrong more times proportionally when trying to classify an email that is spam than an email that is not spam.

3. The obtained confusion matrices with misclassification rates are shown below:

   Train data:

	FALSE	TRUE
0	921	10
1	333	106

   Misclass_train: 0.250365

   Test data:

	FALSE	TRUE
0	931	20
1	314	105

   Misclass_test: 0.2437956

   The misclassification rates have increases compared to the previous model which means that the model is wrong more times than before. This is due to the increased threshold of 0.8 in the model which results in fewer emails being classified as spam by the model. This results in a model which has a high accuracy regarding classifying actual spam emails as spam, but on the other hand classifies many emails as non-spam when they are spam emails. Choosing between the two models, the second one is preferred since much less emails which are not spam are classifies as spam (only 10 for train data and 20 for test data).

4. When applying the knn-method with K=30 on the data the following misclassification rates are obtained:

   Misclass_train: 0.1671533 Misclass_test: 0.3131387

Comparing with step 2, the misclassification rate for the training set is similar but it differs from the misclassification rate for the test set. The value obtained when applying the kknn-method is significantly higher than the one obtained from step 2. This could be due to the reason that the data is mixed in areas where the border between if an email is spam or not is not that clear. In the model this would for example mean that a data point with 16 closest data points classified as spam, will also be classified as spam (since 16/30 > 0.5). Since the mix of data points might be random in these areas it is reasonable that the model performs worse on the test data with the knn-method than when applying logistic regression.

5. When applying the knn-method with K=1 on the data, the following misclassification rates are obtained:

   Misclass_train: 0 Misclass_test: 0.35911241

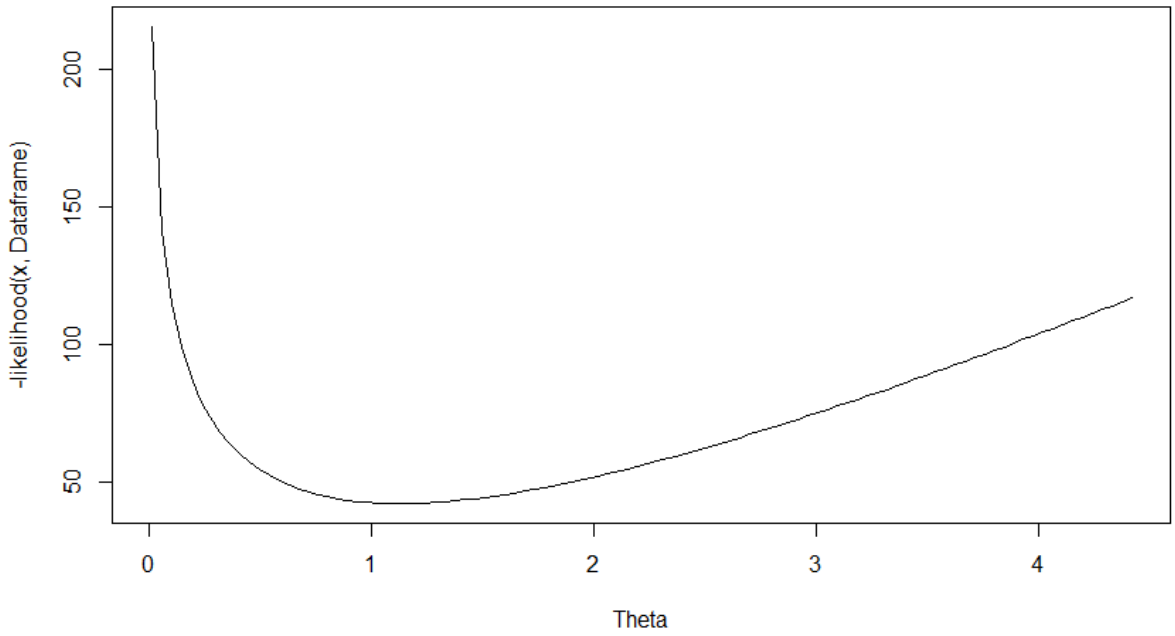
   The fact that the misclassification rate is 0 for the train data is reasonable and could be foreseen beforehand. Since the same data is applied for training the model and testing the model and because of that K=1 is used, the data point in the data that is applied to the model will be the same data point which the model has learned from. The result is that the closest neighbor for each data point is the data point itself. The result of this is that each data applied to the model will be classified correctly. The conclusion to be drawn from this is not that the model is perfect due to that it classified all emails correctly. When applying test data to the model the results are obvious – the model is not perfect and it was wrong about one third of the times classifying an email.

   For the test data the misclassification rate increased when using K=1 compared with when using K=30. This indicates that the model performs worse when using K=1 than when using K=30.

Assignment 2

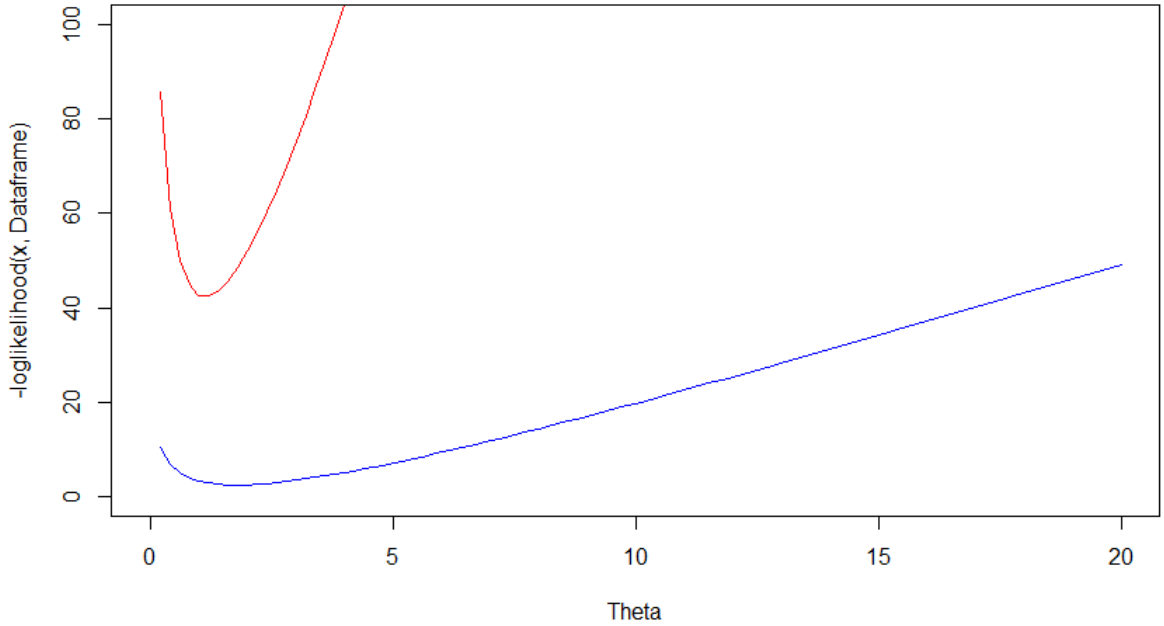
1. The data from “machines.xlsx” is imported which shows the lifetime of certain machines.

2. Assuming the probability model $p(x \mid \theta) = \theta e^{-\theta x}$ where x denotes Length of the lifetime of a machine. The distribution type of x is exponential with mean $\frac{1}{\theta}$. The loglikelihood-function obtained from the probability model above is $n * \log\theta - \sum_{i=1}^{n} x_i$ where n denotes the number of observations in the data set, θ is a parameters aimed to find and x_i denotes observation i. The curve obtained below shows the dependence of the loglikelihood-function on the parameter θ.



From the curve the optimal theta can be obtained which is the lowest point in the curve. Optimal theta was calculated to be 1.126217.

3. When repeating step 2 with but only using the 6 first observations a different curve is obtained. The curve from step 2 as well as the curve from step 3 is shown in the same graph below where the curve from step 2 is RED and the curve from step 3 is BLUE.

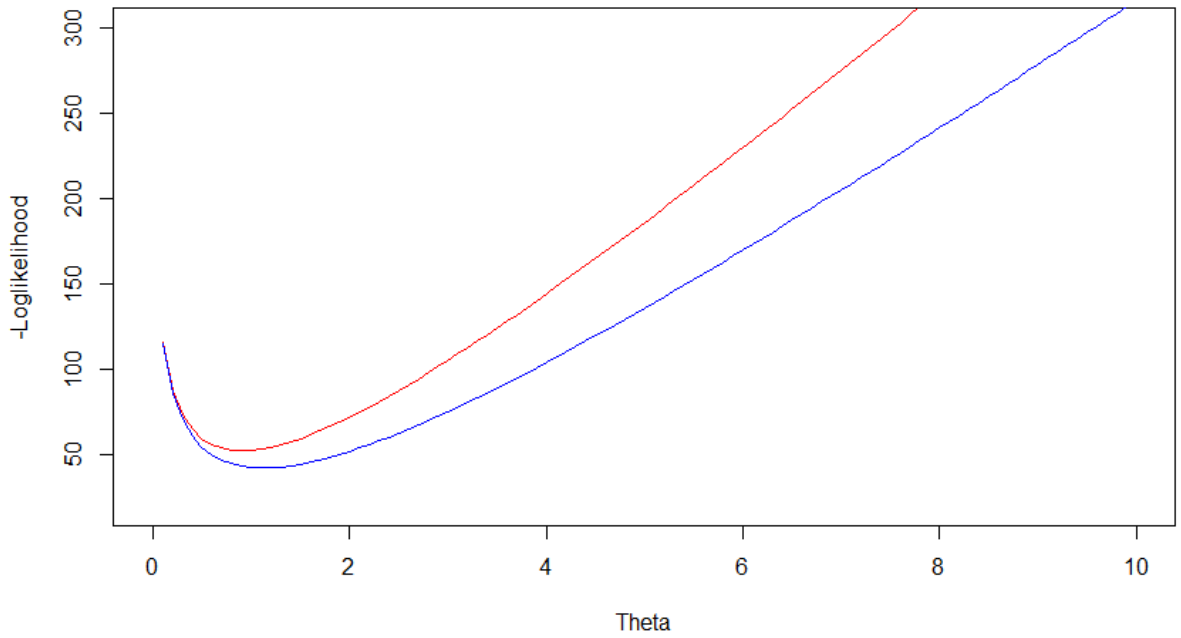


As seen in the graph above the RED curve is tighter and as θ increases the -loglikelihood increases which is not preferred. The aim is to minimize the -loglikelihood-function and it is evident in the graph that only a small interval of θ values results in a low value of the -loglikelihood function for the RED curve. This does not apply as much to the BLUE curve since it increases more slowly as θ increases. This indicates that the RED curve represents a more reliable model than what can be said about the model of the BLUE curve. This is because θ is connected to the mean of the distribution function and it does not seem reasonable that this theta value can range between 0 and 20 without any significant change in the corresponding dependency curve of loglikelihood on θ as in the case of the BLUE curve.

4. When assuming a Bayesian model with $p(x \mid \theta) = \theta e^{-\theta x}$ and a prior $p(\theta) = \lambda e^{-\lambda\theta}$ where $\lambda = 10$ the following loglikelihood-function is obtained:

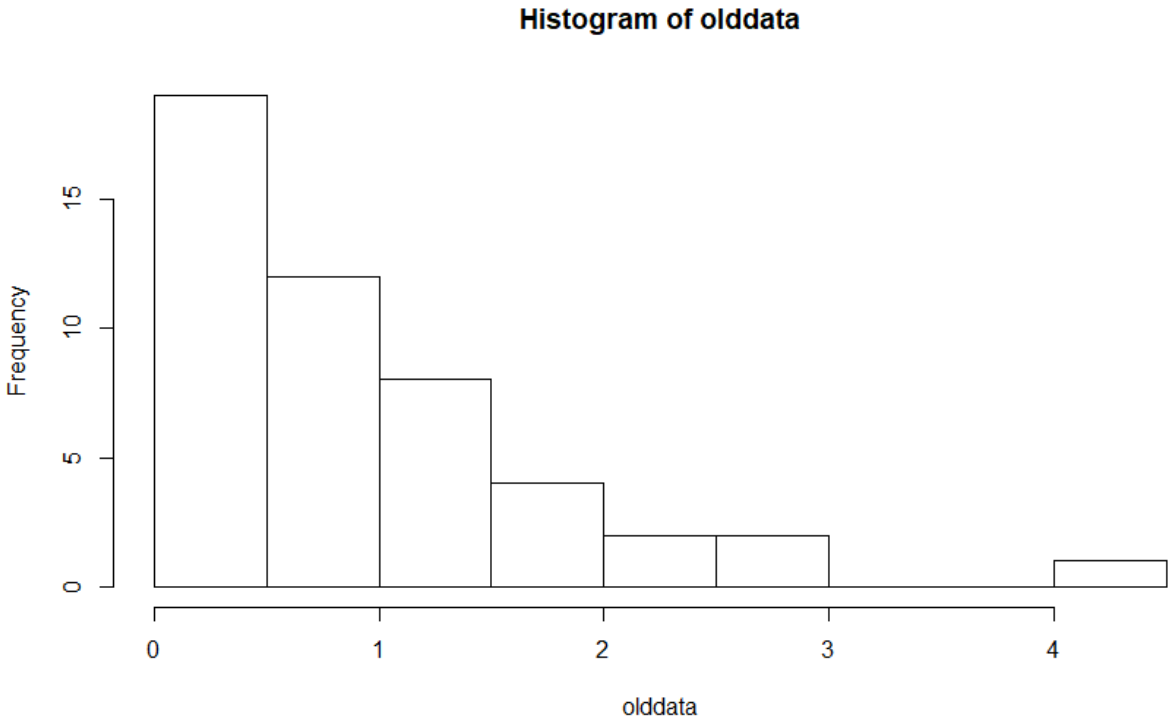
   $$f(\theta) = n * \log\theta - \theta * \sum_{i=1}^{n} x_i - 10 * \theta$$

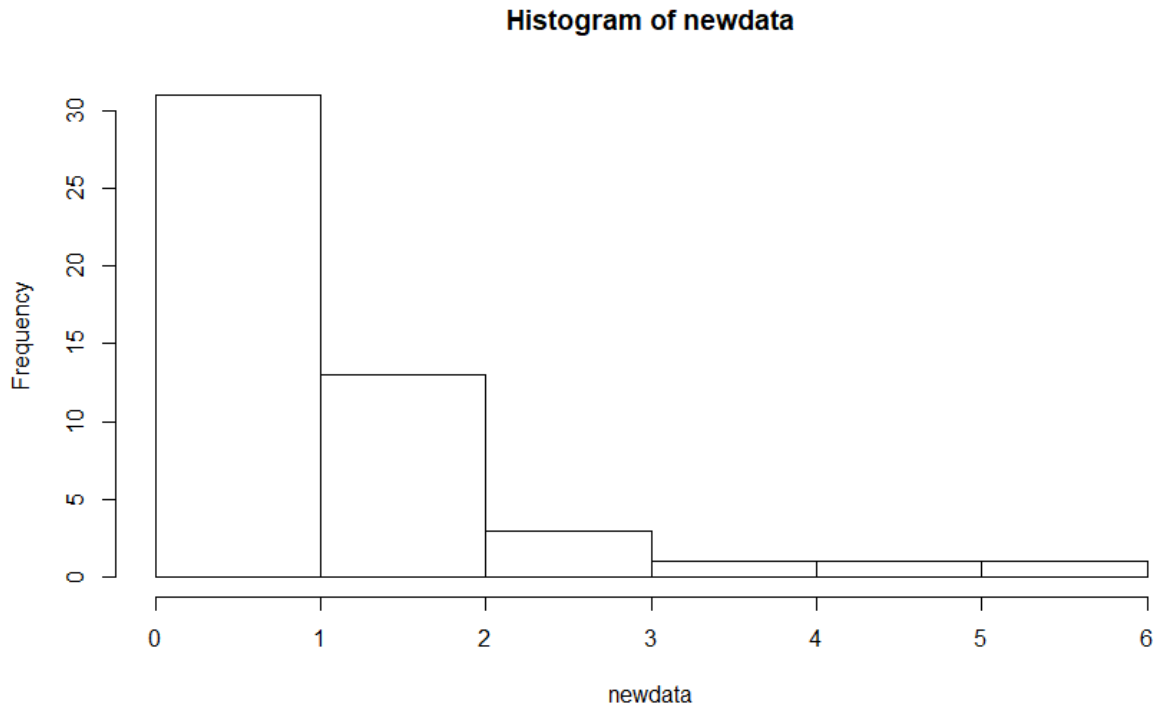
   The measure actually computed by this function is the logarithm of the probability that x and θ takes on certain values at the same time, i.e. $p(x \cap \theta)$. The prior $p(\theta)$ can be seen as data already observed which the Bayesian model is multiplied with to obtain the total probability.



Above is the graph showing the dependency of Bayesian model (RED) and the model from step 2 (BLUE), on θ. As seen in the graph the optimal value of θ for the Bayesian model is 0.9121907. This is a value closer to the actual mean of the data used and the Bayesian model is more reliable since the curve is tighter.

5. Now if using the model and optimal θ from step 2, 50 new observations can be generated. The histograms of the original data and the newly generated data with optimal θ are shown below.

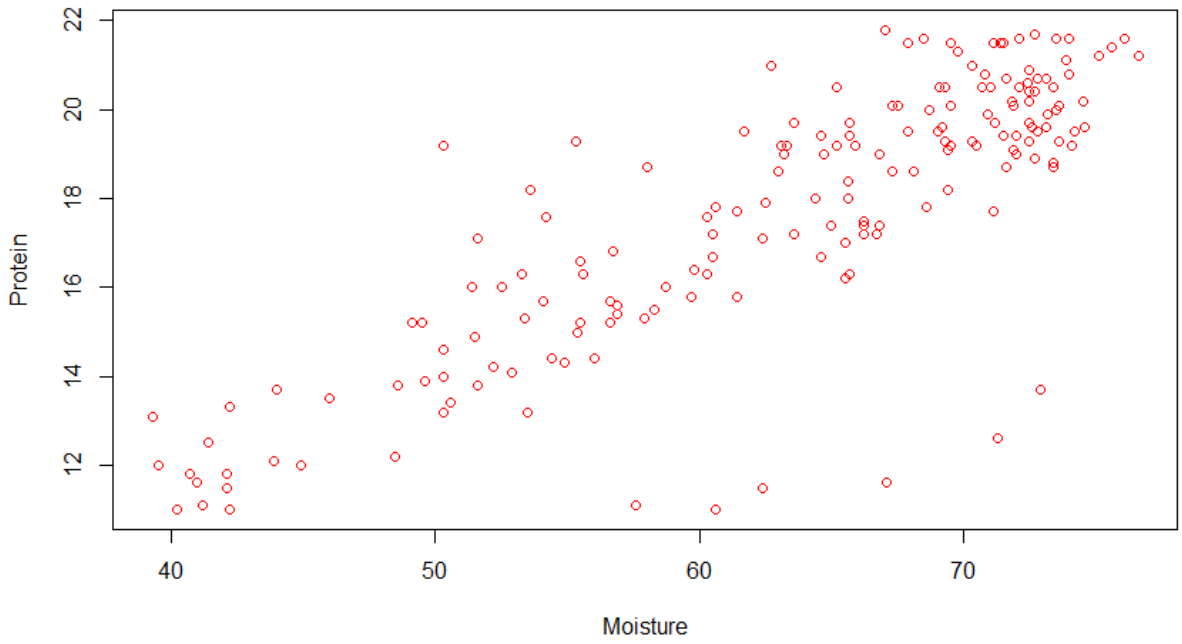




The histograms look similar with the difference that the generated values are more evenly distributed than the original data set which is reasonable since the generated values was created by using a distribution function. The histograms also show that it is reasonable to apply an exponential distribution function to them since they follow this pattern in the histograms above.

Assignment 4

1. When plotting the Moisture vs Protein the following graph is obtained.



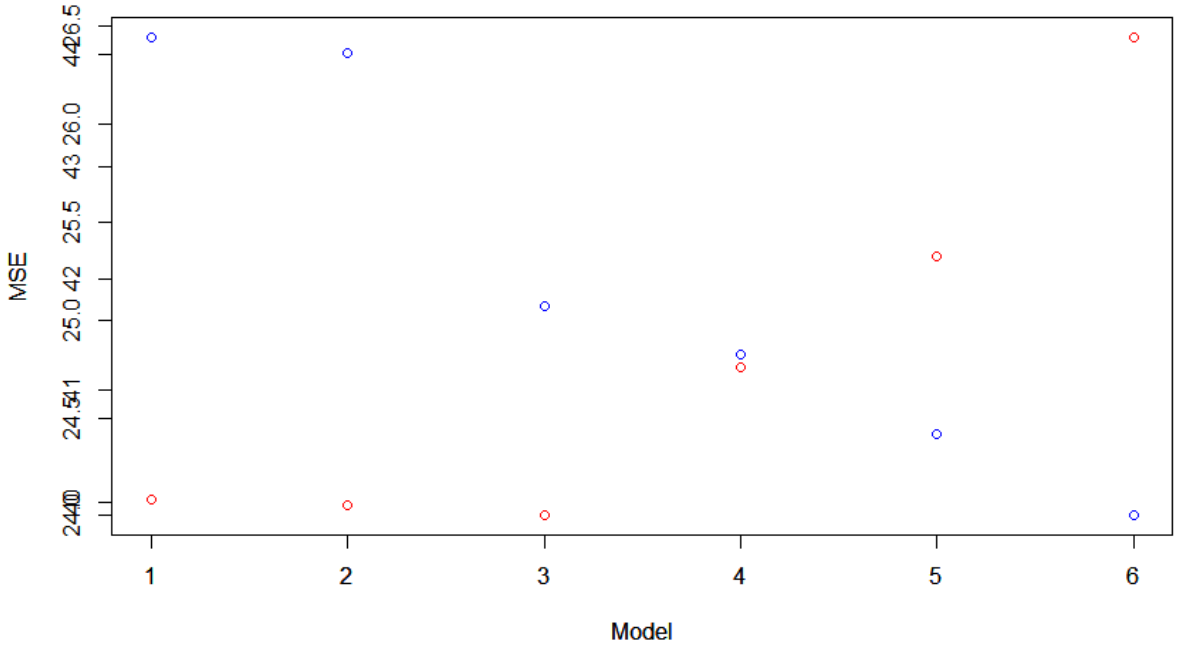
It looks like it might be a linear relation between moisture and protein.

2. Below is the probabilistic model which describes the relation of Moisture to Protein.

$$M_i \sim N(\sum_{i=0}^{n} w_i x_i, \sigma^2)$$

MSE is an appropriate measure to use since it calculates how big the errors are in the model (also called residuals). By using MSE it is easy to see which model that is preferred.

3. The plot below shows the relation between MSE and the different polynomial models.

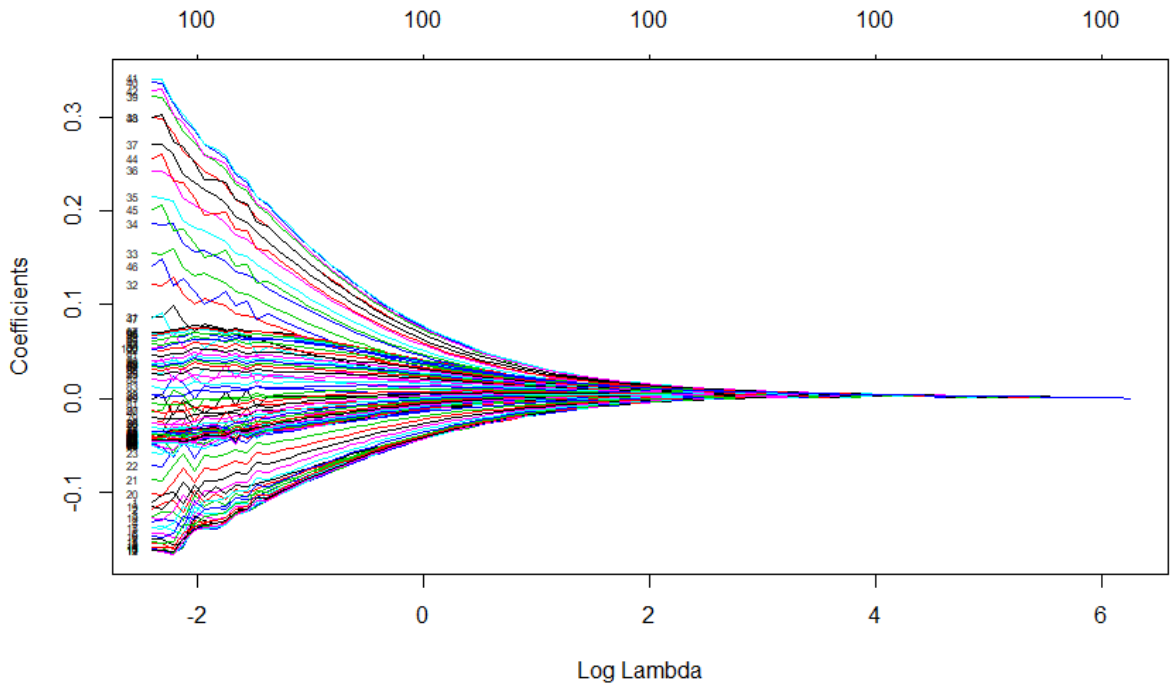


The red dots describe the test data and the blue dots describe the training data. The best model to use is model 3 since it has the lowest MSE for the test data. When complexity increases the variance between test and training increases. And the MSE decreases for training data as the complexity increases, this means that the bias decrease as it is more able to fit the data.

4. Below is the obtained model from using stepAIC. Resulting in the best model with 63 variables selected.

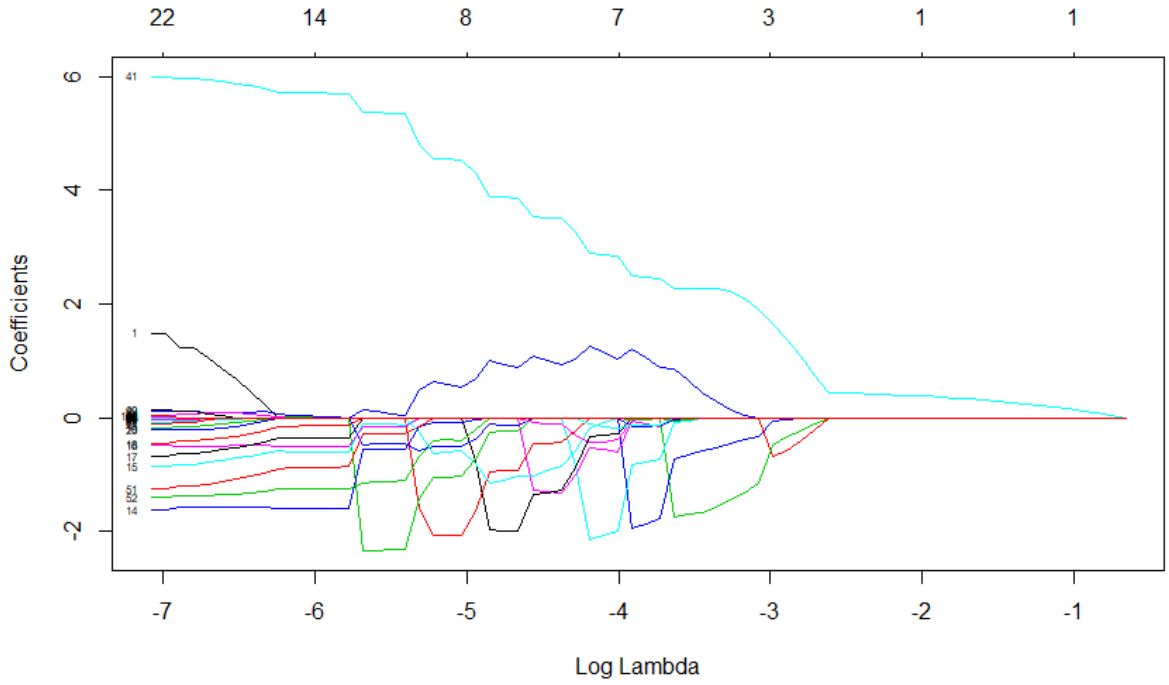
```
Call:
lm(formula = fat ~ Channel1 + Channel2 + Channel4 + Channel5 +
    Channel7 + Channel8 + Channel11 + Channel12 + Channel13 +
    Channel14 + Channel15 + Channel17 + Channel19 + Channel20 +
    Channel22 + Channel24 + Channel25 + Channel26 + Channel28 +
    Channel29 + Channel30 + Channel32 + Channel34 + Channel36 +
    Channel37 + Channel39 + Channel40 + Channel41 + Channel42 +
    Channel45 + Channel46 + Channel47 + Channel48 + Channel50 +
    Channel51 + Channel52 + Channel54 + Channel55 + Channel56 +
    Channel59 + Channel60 + Channel61 + Channel63 + Channel64 +
    Channel65 + Channel67 + Channel68 + Channel69 + Channel71 +
    Channel73 + Channel74 + Channel78 + Channel79 + Channel80 +
    Channel81 + Channel84 + Channel85 + Channel87 + Channel88 +
    Channel92 + Channel94 + Channel98 + Channel99, data = channel_values)
```

5. Below is the graph showing the relation between the coefficients and lambda for the RIDGE-model.



As lambda increases the coefficients become more penalized and go towards the value 0. Since Ridge regression uses quadratic penalty it is a smooth curve for the decreasing of the coefficients values. This lets us make sure that the model is not overfitted, it also penalized outliers more than others due to quadratic penalty.

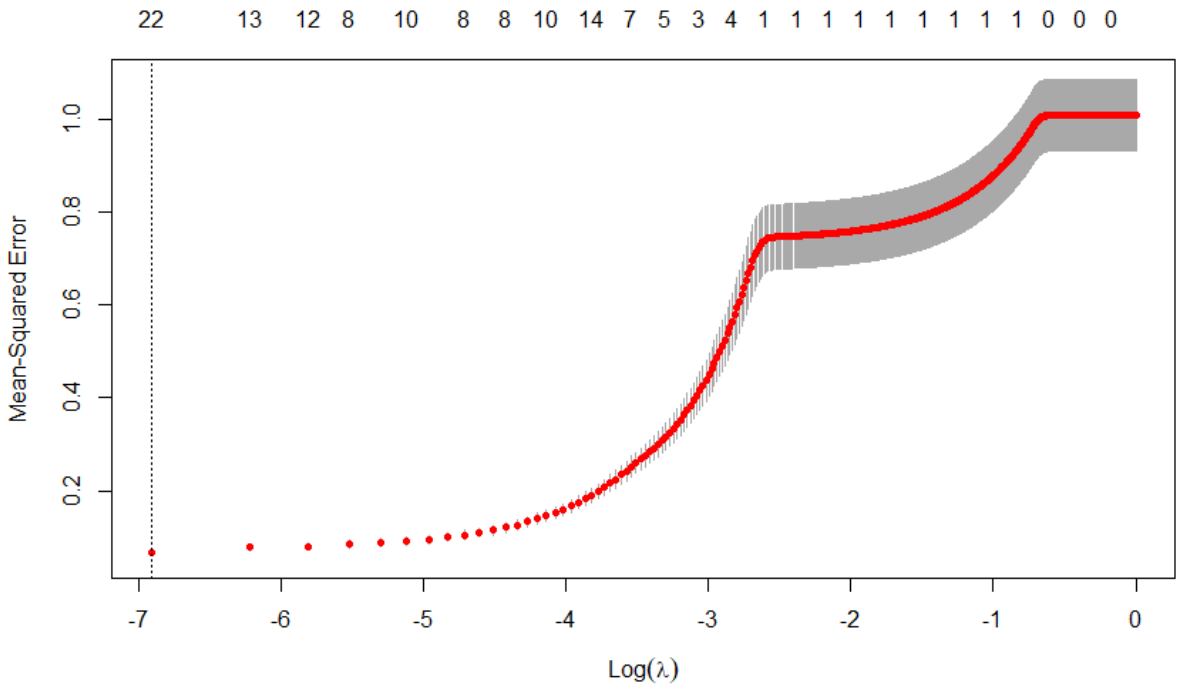
6. Below is the graph showing the relation between coefficients and lambda for the LASSO-model.



In this plot we can see that the coefficients value also go towards 0 when lambda increases. It penalizes the coefficients linearly and have hard edges because the feasible area for the

model is restricted by a polygon.

7.



The optimal lambda for the LASSO model is 0. It is because there is no penalty then and since the model is created from the training data and tested on same data it wants to overfit to decrease towards minimal MSE.

Appendix

```
#Read data and divide into test and train 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,]

#Create model for prediction
spammodel = glm(Spam~., family='binomial', data=train)
summary(spammodel)

#Predict values and create confusion matrix for traindata
predicted_values_train = predict(spammodel, newdata=train, type='response')
confusion_matrix_train = table(train$Spam, predicted_values_train>0.5)
print(confusion_matrix_train)

#Predict values and create confusion matrix for testdata
predicted_values_test = predict(spammodel, newdata=test, type='response')
confusion_matrix_test = table(test$Spam, predicted_values_test>0.5)
print(confusion_matrix_test)

#Create function for misclassification rate
missclass=function(conf_matrix, fit_matrix){
 n=length(fit_matrix[,1])
 return(1-sum(diag(conf_matrix))/n)
}

#Calculate missclassification rate for train and test data
```

```
missclass_train = missclass(confusion_matrix_train, train)
print(missclass_train)
missclass_test = missclass(confusion_matrix_test, test)
print(missclass_test)

#Create confusion matrix where classification is based on threshold 0.8
confusion_matrix_train2 = table(train$Spam, predicted_values_train>0.8)
confusion_matrix_test2 = table(test$Spam, predicted_values_test>0.8)
print(confusion_matrix_train2)
print(confusion_matrix_test2)

#Calculate missclassification rate for train and test data with threshold 0.8
missclass_train2 = missclass(confusion_matrix_train2, test)
print(missclass_train2)
missclass_test2 = missclass(confusion_matrix_test2, train)
print(missclass_test2)

#Fetch package kkm
#install.packages("kknn")
library("kknn")

#Classify according to kknn with k=30 for test and train data sets
kknn_30_train = kknn(formula = Spam~., train, train, k=30)
kknn_30_test = kknn(formula = Spam~., train, test, k=30)
confusion_matrix_kknn30_train = table(train$Spam, kknn_30_train$fitted.values>0.5)
missclass_kknn30_train = missclass(confusion_matrix_kknn30_train, train)
confusion_matrix_kknn30_test = table(test$Spam, kknn_30_test$fitted.values>0.5)
missclass_kknn30_test = missclass(confusion_matrix_kknn30_test, test)
print(confusion_matrix_kknn30_train)
print(missclass_kknn30_train)
print(confusion_matrix_kknn30_test)
```

```
print(missclass_kknn30_test)


#Classify according to kknn with k=1 for test and train data sets
kknn_1_train = kknn(formula = Spam~., train, train, k=1)
kknn_1_test = kknn(formula = Spam~., train, test, k=1)
confusion_matrix_kknn1_train = table(train$Spam, kknn_1_train$fitted.values>0.5)
missclass_kknn1_train = missclass(confusion_matrix_kknn1_train, train)
confusion_matrix_kknn1_test = table(test$Spam, kknn_1_test$fitted.values>0.5)
missclass_kknn1_test = missclass(confusion_matrix_kknn1_test, test)
print(confusion_matrix_kknn1_train)
print(missclass_kknn1_train)
print(confusion_matrix_kknn1_test)
print(missclass_kknn1_test)
```

Assignment 2

```
#Read data and divide into test and train sets
Dataframe=read.csv2("machines_csv.csv")


#Compute a function for calculating the maximum likelihood of a function
loglikelihood=function(theta, x){
  n = length(x[,1])
  return(n*log(theta)-theta*sum(x))
}


#Plot curve for different theta values
theta_curve = curve(-loglikelihood(x, Dataframe), xlab="Theta", from=min(Dataframe),
to=max(Dataframe))


#Find maximum likelihood value of theta
theta_max = function(x){
```

```
 n=length(x[,1])
 return(n/sum(x))
}

#Find maxtheta
max_theta = theta_max(Dataframe)
print(max_theta)

#New vector with first 6 values
y = matrix(Dataframe[1:6,1], nrow=length(Dataframe[1:6,1]), ncol=1)
print(y)

#Plot new curve on top of each other
curve(-loglikelihood(x, Dataframe), xlab="Theta", from=0, to=20, add=FALSE, col="red",
ylim=c(0,100))
curve(-loglikelihood(x, y), xlab="Theta", from=0, to=20, add=TRUE, col="blue", ylim=c(0,100))

#Compute a function for calculating the likelihood of the bayesian function
bayesian_likelihood=function(theta, lambda, x){
 n = length(x[,1])
 return(n*log(theta)-theta*sum(x)-lambda*theta)
}

#Find maximum likelihood value of theta
bayesian_theta_max = function(lambda, x){
 n=length(x[,1])
 return(n/(sum(x)+lambda))
}

#Find maxtheta
bayesian_max_theta = bayesian_theta_max(10, Dataframe)
```

```
print(bayesian_max_theta)


#Plot new curve on top of each other

curve(-bayesian_likelihood(x, 10, Dataframe), ylab="-Loglikelihood", xlab="Theta", from=0, to=10, add=FALSE, col="red", ylim=c(20,300))

curve(-loglikelihood(x, Dataframe), ylab="-Loglikelihood", xlab="Theta", from=0, to=10, add=TRUE, col="blue", ylim=c(20,300))


#Generate 50 new observation using theta value from step 2

set.seed(12345)

newdata = rexp(50, rate = max_theta)

print(newdata)


#Plot new data and old data in histogram

olddata = Dataframe$Length

print(olddata)

hist(newdata)

hist(olddata)
```

Assignment 4

```
#Read data and plot Moisture vs Protein

Dataframe=read.csv2("tecator_csv.csv")

n = length(Dataframe[,1])

print(Dataframe)

moisture = Dataframe$Moisture

protein = Dataframe$Protein

fat = Dataframe$Fat

plot(moisture, protein, type="p", ylab="Protein", xlab="Moisture", col="red")


#Conclusion: Looks like a linear relation so a linear regression model is appropriate


#Divide data into training and validation sets
```

```
colno_protein = which(colnames(Dataframe)=="Protein")
colno_moisture = which(colnames(Dataframe)=="Moisture")
moisture_protein = Dataframe[colno_protein:colno_moisture]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=Dataframe[id,]
test=Dataframe[-id,]
moisture_train = train$Moisture
moisture_test = test$Moisture

#Create function for fitting linear regression models
fit_moisture_model = function(x) {
  return(lm(formula = Moisture ~ poly(Protein, degree=x), data=train))
}

#Create predict function for training set
predict_train = function(model){
  return(predict(model, newdata = train))
}

#Create predict function for test set
predict_test = function(model){
  return(predict(model, newdata = test))
}

#Create function for calculating MSE, input parameters are vectors containing original data and
predicted data
calcMSE = function(y, yhat){
  return(sum((y-yhat)^2)/length(y))
}
```

```
#Create models and predictions and store MSE values in a vector for training and test data
vector_train = c()
vector_test = c()
for (i in 1:6){
  fit = fit_moisture_model(i)
  predicted_train = predict_train(fit)
  predicted_test = predict_test(fit)
  vector_train[i] = calcMSE(moisture_train, predicted_train)
  vector_test[i] = calcMSE(moisture_test, predicted_test)
}

#Create numeric vector 1 through 6
models = c(1:6)

#Plot MSE for each model
plot(models, vector_train, col="blue", xlab="Model", ylab="MSE")
par(new=TRUE)
plot(models, vector_test, col="red", xlab="Model", ylab="MSE")

#Print MSE values
print(vector_train)
print(vector_test)

#Assignment 4: Fat response and Channel1-100 are predictors. Use stepAIC. How many variables were selected?

#Fetch package stepAIC
#install.packages("MASS")
library("MASS")

#Perform stepAIC on fat
```

```
colno_fat = which(colnames(Dataframe)=="Fat")

channel_values = Dataframe[1:(colno_fat-1)]

fit_fat = lm(fat~., data = channel_values)

step = stepAIC(fit_fat, direction="both")

step$anova

summary(step)


#Fit a Ridge regression model with same predictor and response variables. Plot model coefficient depend on the log

# of the penalty factor lambda.

#install.packages("glmnet")

library("glmnet")

covariates = scale(Dataframe[,2:(colno_fat-1)])

response = scale(Dataframe[,colno_fat])

ridge_model = glmnet(as.matrix(covariates), response, alpha=0, family="gaussian")

plot(ridge_model, xvar="lambda", label=TRUE)


#Coefficients goes towards 0 when lambda goes towards infinity


#Repeat last step but with LASSO instead of Ridge. Differences?

lasso_model = glmnet(as.matrix(covariates), response, alpha=1, family="gaussian")

plot(lasso_model, xvar="lambda", label=TRUE)


#Choose the best model by cross validation for LASSO model.

lasso_model_optimal = cv.glmnet(as.matrix(covariates), response, alpha=1, family="gaussian", lambda=seq(0,1,0.001))

lasso_model_optimal$lambda.min

plot(lasso_model_optimal)

coef(lasso_model_optimal, s="lambda.min")

print(lasso_model_optimal$lambda.min)
```