Lab 1 TDDE01

Filip Cornell

2018-11-12

1 Assignment 1

First, we initialize some functions and load the data.

```
library(readxl)
data = read_excel("spambase.xlsx")
data$Spam = factor(data$Spam)
# Ensure this is correct when online again
log_reg = function(data, fitModel, pLimit = 0.5) {
  # Specify response in predict,
  fits = predict(fitModel, data)

  probabilities = fits

  classifications = apply(as.matrix(probabilities), 1, function(row) {
    if (row > pLimit) {
      return(1)
    } else {
      return(0)
    }
  })
  return(classifications)
}

classificationRate = function(confMatrix) {
  return(sum(diag(confMatrix))/sum(confMatrix))
}
```

1.1 Task 1

Loading in the data and setting the seed and dividing it into training and test sets is a trivial task.

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

1.2 Task 2

In this task, our mission was to classify the spam using logistic regression and report the confusion matrices, which can be found in the output below.

```r
#Task 2

fit = glm(Spam ~ ., data = train, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
#summary(fit)
fits = predict(fit, train)
classificationsTrain = log_reg(train, fit)
classificationsTest = log_reg(test, fit)

print("misclassification rate train")
```

```
## [1] "misclassification rate train"
```

```r
table(train$Spam, classificationsTrain)
```

```
##    classificationsTrain
##       0   1
##   0 864  81
##   1 146 279
```

```r
1 - classificationRate(table(train$Spam, classificationsTrain))
```

```
## [1] 0.1656934
```

```r
print("misclassification rate test")
```

```
## [1] "misclassification rate test"
```

```r
table(test$Spam, classificationsTest)
```

```
##    classificationsTest
##       0   1
##   0 846  91
##   1 163 270
```

```r
1 - classificationRate(table(test$Spam, classificationsTest))
```

```
## [1] 0.1854015
```

We get an accuracy of 83 % on the training set, and a test set accuracy of 81 %. This is quite a good accuracy, although not perfect. However, they are very alike, indicating that it is probably not overfitted.

1.3 Task 3

In task 3, we changed the classification probability to ensuring it has 0.9 as probability to be 1 to be classified as 1.

```r
## Task 3

classificationsTrain = log_reg(train, fit, pLimit = 0.9)
classificationsTest = log_reg(test, fit, pLimit = 0.9)

print("misclassification rate train")
```

```
## [1] "misclassification rate train"
```

```r
table(train$Spam, classificationsTrain)
```

```
##    classificationsTrain
##       0   1
##   0 917  28
##   1 217 208
```

```r
1 - classificationRate(table(train$Spam, classificationsTrain))
```

```
## [1] 0.1788321
```

```r
print("misclassification rate test")
```

```
## [1] "misclassification rate test"
```

```r
table(test$Spam, classificationsTest)
```

```
##    classificationsTest
##       0   1
##   0 894  43
##   1 250 183
```

```r
1 - classificationRate(table(test$Spam, classificationsTest))
```

```
## [1] 0.2138686
```

We see that the results are slightly worse, although not much worse. This is because some data points that lies within the interval $0.5 < p \leq 0.9$ are classified incorrectly, as they most likely actually are 1. However, the stricter rule prohibits these from being classified correctly, thus yield more false negatives.

1.4 Task 4

Here, the task was to use the standard classifier `kknn()` with K = 30. The misclassification rate here is higher, indicating that logistic regression is probably a better classification method in this case.

```r
# Task 4
#install.packages("kknn")
library("kknn")


kkn.fit = kknn(Spam ~., train = train, test = train, k = 30)

print("misclassification rate train")
```

```
## [1] "misclassification rate train"
```

```r
table(train$Spam, kkn.fit$fitted.values)
```

```
##
##       0   1
##   0 807 138
##   1  98 327
```

```r
1 - classificationRate(table(train$Spam, kkn.fit$fitted.values))
```

```
## [1] 0.1722628
```

```r
kkn.fit = kknn(Spam ~., train = train, test = test, k = 30)

print("misclassification rate test")
```

```
## [1] "misclassification rate test"
```

```r
table(test$Spam, kkn.fit$fitted.values)
```

```
##
##       0   1
##   0 672 265
##   1 187 246
```

```r
1 - classificationRate(table(test$Spam, kkn.fit$fitted.values))
```

```
## [1] 0.329927
```

1.5 Task 5

Changing to K = 1 worsened the results, yielding a slightly higher, although not very different, misclassification rate. We get more false positives than before.

```r
# Task 5

kkn.fit = kknn(Spam ~., train, train, k = 1)

print("misclassification rate train")
```

```
## [1] "misclassification rate train"
```

```r
table(train$Spam, kkn.fit$fitted.values)
```

```
##
##       0   1
##   0 945   0
##   1   0 425
```

```r
1 - classificationRate(table(train$Spam, kkn.fit$fitted.values))
```

```
## [1] 0
```

```r
kkn.fit = kknn(Spam ~., train, test, k = 1)

print("misclassification rate test")
```

```
## [1] "misclassification rate test"
```

```r
table(test$Spam, kkn.fit$fitted.values)
```

```
##
##       0   1
##   0 640 297
##   1 177 256
```

```r
1 - classificationRate(table(test$Spam, kkn.fit$fitted.values))
```

```
## [1] 0.3459854
```

We clearly overfit on the training data with K = 1, yielding a worse result on the test data.

2 Assignment 2

2.1 Task 1

First, we import the data.

```
# Task 1
library(readxl)
data = read_excel("machines.xlsx")
```

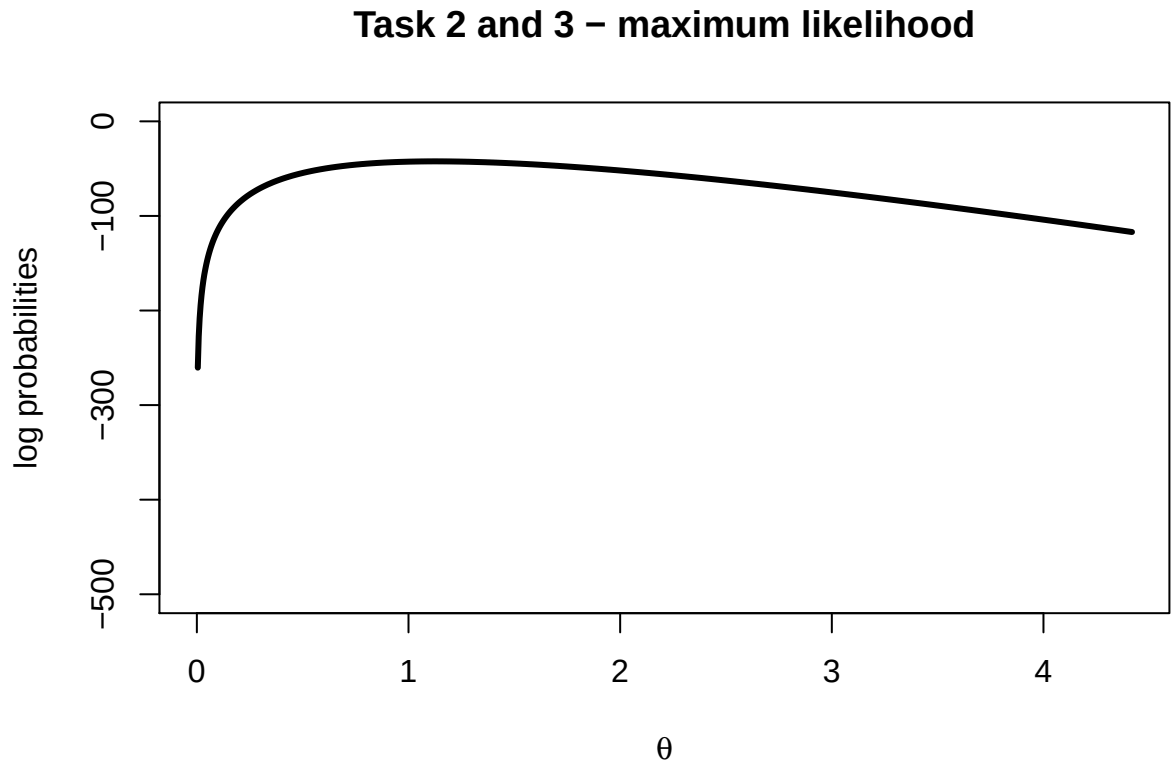
2.2 Task 2

Here, the task was to compute the log likelihood of $p(x|\theta) = \theta e^{-\theta x}$. The likelihood can be described as which can be described as

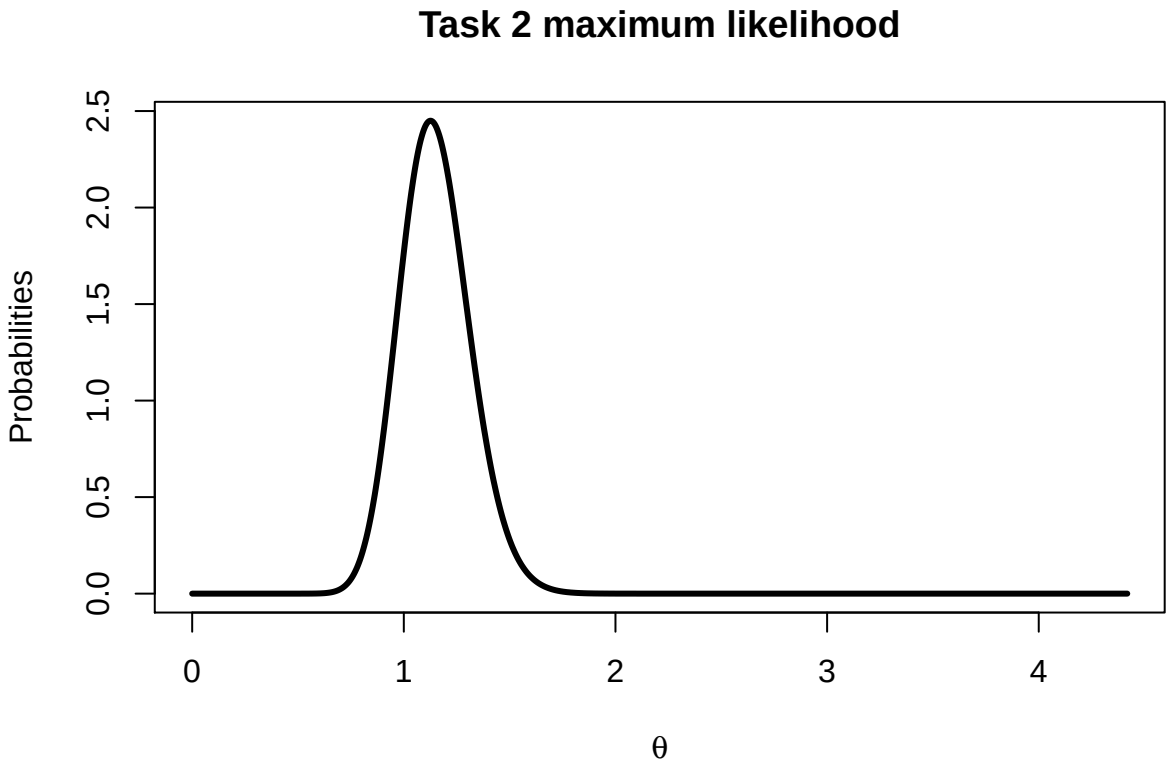
$$\sum_{i=1}^{n} log\big(p(x_i|\theta)\big) = \sum_{i=1}^{n} log\big(\theta e^{-\theta x_i}\big) = \sum_{i=1}^{n} log\big(\theta\big) - \theta x_i = nlog(\theta) - \theta \sum_{i=1}^{n} x_i$$

This results in the plot given below.

```
# Task 2

logLikelihood = function(theta, data) {
  return(sum(log(theta) - theta*data))
}

normalize = function(scale, probs) {
  return(scale*probs/sum(probs))
}

thetaGrid = seq(0,max(data), length = 1000)
scale = 1/(thetaGrid[2] - thetaGrid[1])
logProbs = apply(as.matrix(thetaGrid), 1, logLikelihood, data)

probsNormalizedTask2 = normalize(scale, exp(logProbs))
plot(x = thetaGrid, y = logProbs,
     type = "l", lwd = 3,
     xlab = expression(theta),
     main = "Task 2 and 3 - maximum likelihood", ylab = "log probabilities", ylim = c(-500,0))
```




```
plot(x = thetaGrid, y = probsNormalizedTask2,
     type = "l", lwd = 3,
     xlab = expression(theta),
     main = "Task 2 maximum likelihood",
     ylab = "Probabilities")
```



```r
maxTheta = thetaGrid[which.max(probsNormalizedTask2)]
print(paste("Max value of theta is ", maxTheta))
```

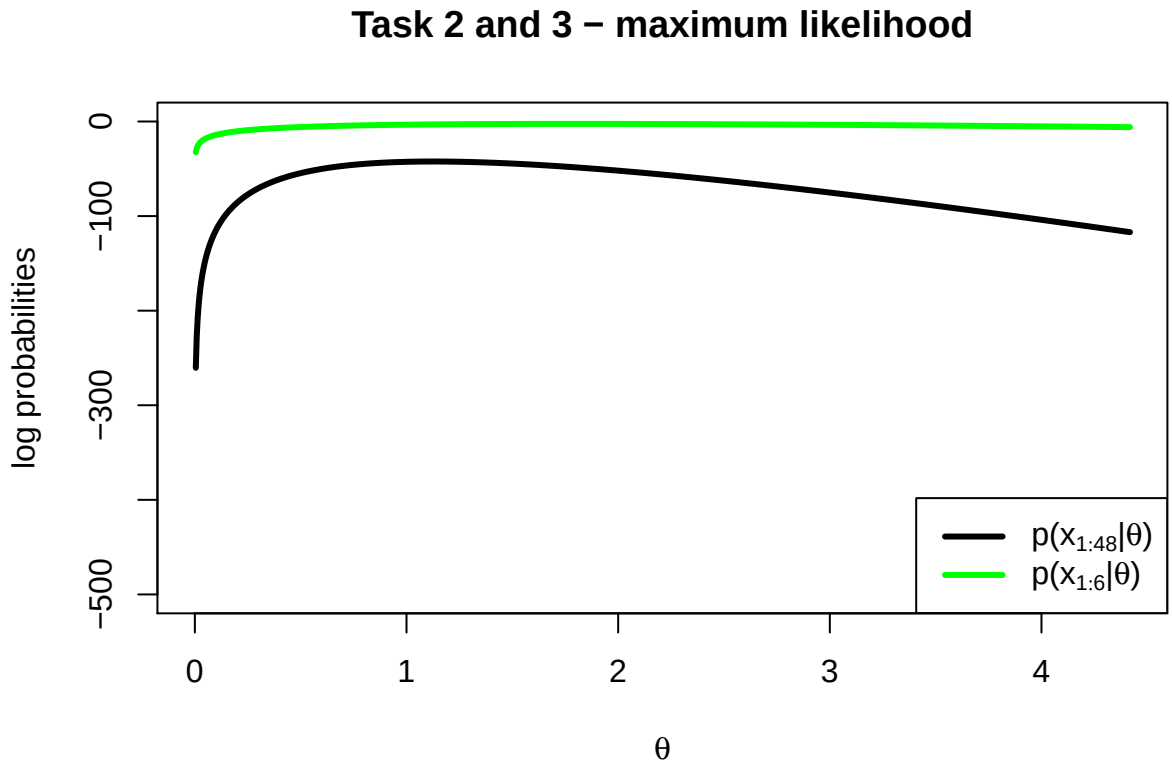
```
## [1] "Max value of theta is  1.12779300909544"
```

We also see that the most likely value of θ is around 1.128.

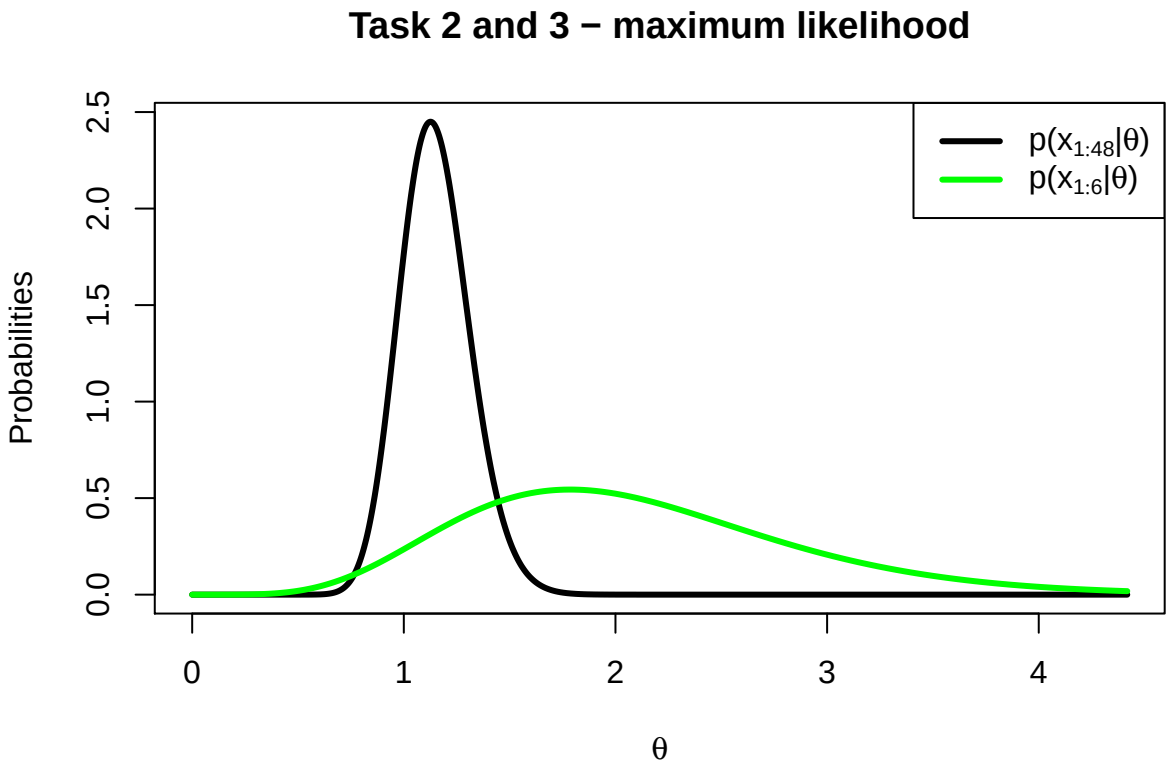
2.3 Task 3

In task 3, we use fewer observations, but use the maximum likelihood function just as before.

```r
# Task 3
cutData = data$Length[1:6]
logProbsTask3 = apply(as.matrix(thetaGrid), 1, logLikelihood, cutData)
probsNormalizedTask3 = normalize(scale, exp(logProbsTask3))

plot(x = thetaGrid, y = logProbs,
     type = "l", lwd = 3,
     xlab = expression(theta),
     main = "Task 2 and 3 - maximum likelihood", ylab = "log probabilities", ylim = c(-500,0))
lines(x= thetaGrid, y = logProbsTask3, type = "l", lwd = 3, col = "green")

legend("bottomright",
       lty = rep(1,2),
       col = c("black", "green"),
       lwd = c(3,3),
       legend = c(              expression(paste("p(",x[1:48],"|",theta,")")),
                  expression(paste("p(",x[1:6],"|",theta,")")))
       )
```



```r
plot(x = thetaGrid, y = probsNormalizedTask2,
     type = "l", lwd = 3,
     xlab = expression(theta),
     main = "Task 2 and 3 - maximum likelihood", ylab = "Probabilities")
lines(x= thetaGrid, y = probsNormalizedTask3, type = "l", lwd = 3, col = "green")

legend("topright",
       lty = rep(1,2),
       col = c("black", "green"),
       lwd = c(3,3),
       legend = c(              expression(paste("p(",x[1:48],"|",theta,")")),
                  expression(paste("p(",x[1:6],"|",theta,")")))
       )
```



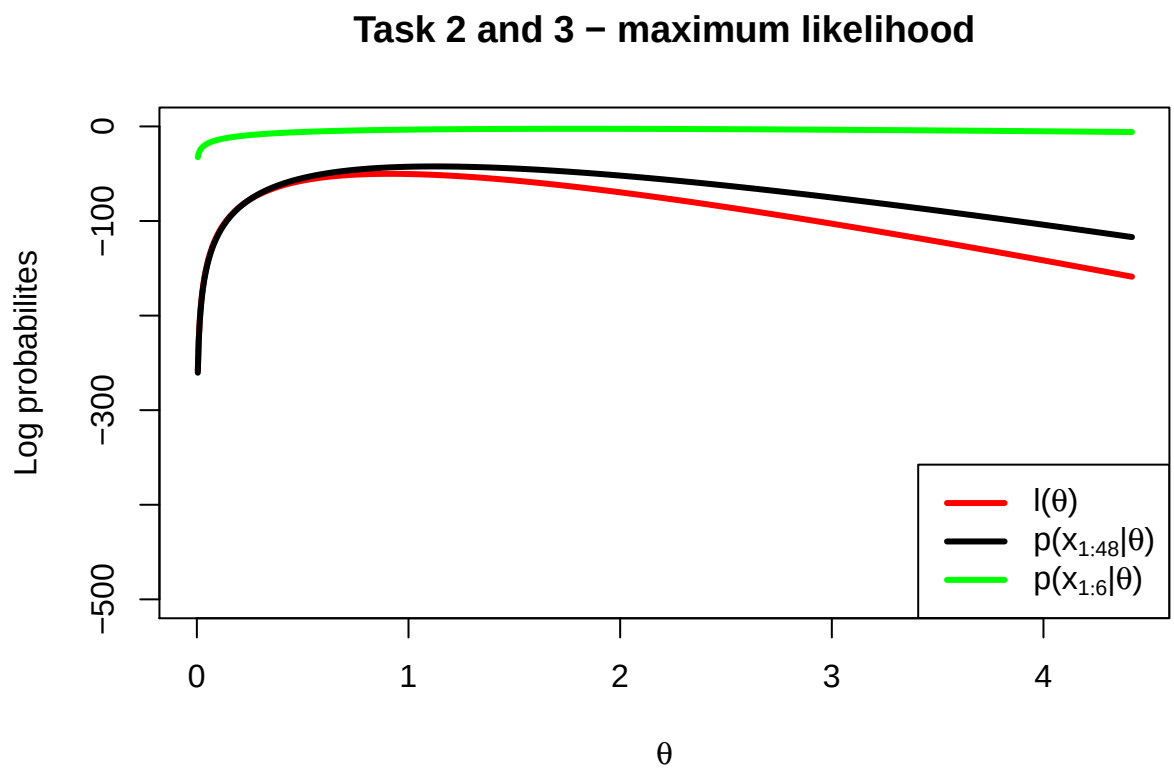
This yields a distribution around a different value of θ, with a lower probability. This is because we have fewer observation, and we can thus not be as sure as in task 2.

2.4 Task 4

In task 4, we want to use a prior $p(\theta) = \lambda e^{-\lambda\theta}$, in other words, that θ also follows an exponential distribution. With this and the maximum likelihood function, we create the posterior distribution $l(\theta)$ and obtain a new distribution for θ

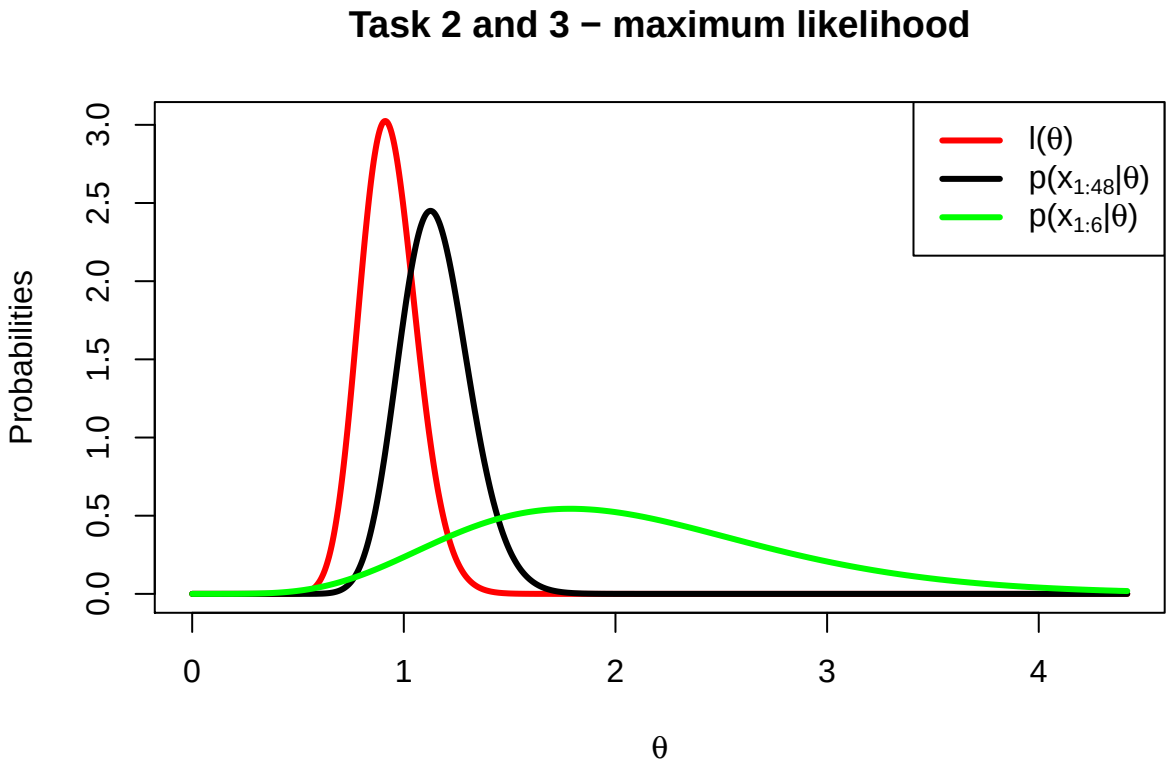
```r
# Task 4


l_theta = function(theta, data, lambda = 10) {
```

```
  log_dpois = log(lambda*exp(-lambda*theta))
  return(logLikelihood(theta, data) + log_dpois)
}

logProbsTask4 = apply(as.matrix(thetaGrid), 1, l_theta, data)

probsNormalizedTask4 = normalize(scale, exp(logProbsTask4))

plot(x = thetaGrid, y = logProbsTask4,
     type = "l", col = "red",
     lwd = 3, xlab = expression(theta),
     main = "Task 2 and 3 - maximum likelihood", ylab = "Log probabilites",
     ylim = c(-500,0))
lines(x=thetaGrid, y = logProbs, lwd = 3)
lines(x= thetaGrid, y = logProbsTask3, type = "l", lwd = 3, col = "green")

legend("bottomright",
       lty = rep(1,3),
       col = c("red", "black", "green"),
       lwd = c(3,3,3),
       legend = c(expression(paste("l(",theta,")")),
                  expression(paste("p(",x[1:48],"|",theta,")")),
                  expression(paste("p(",x[1:6],"|",theta,")")))
       )
```




```
plot(x = thetaGrid, y = probsNormalizedTask4,
     type = "l", col = "red",
     lwd = 3, xlab = expression(theta),
     main = "Task 2 and 3 - maximum likelihood", ylab = "Probabilities")
```

```
lines(x=thetaGrid, y = probsNormalizedTask2, lwd = 3)
lines(x= thetaGrid, y = probsNormalizedTask3, type = "l", lwd = 3, col = "green")

legend("topright",
       lty = rep(1,3),
       col = c("red", "black", "green"),
       lwd = c(3,3,3),
       legend = c(expression(paste("l(",theta,")")),
                  expression(paste("p(",x[1:48],"|",theta,")")),
                  expression(paste("p(",x[1:6],"|",theta,")")))
       )
```



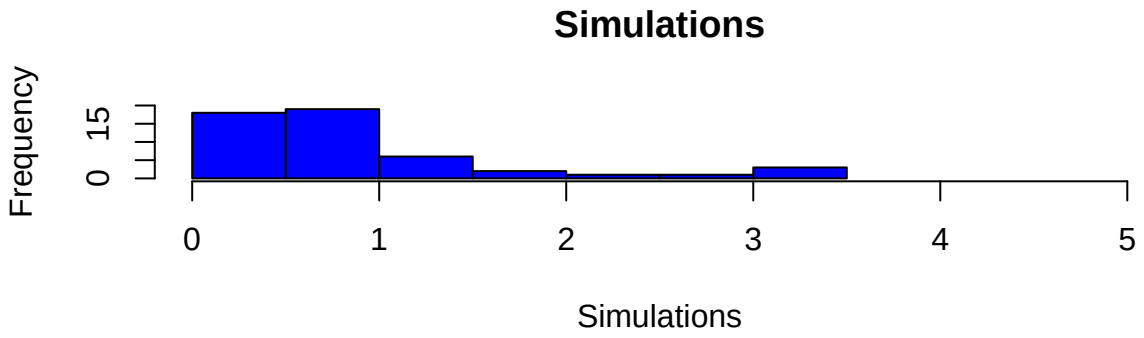
This is has an even higher probability to be true. Do note that these are not completely correctly normalized, but the relation between the three is correct.

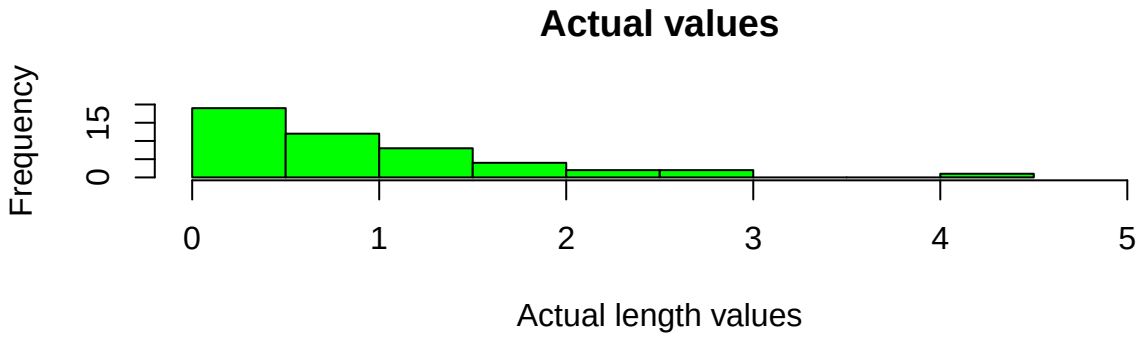
2.5 Task 5

In this task, we were supposed to simulate some observations from $p(x|\theta) = \theta e^{-\theta x}$. Since this is an exponential distribution, we can simply generate sample from the built-in function `rexp()` in R.

```
# TASK 5

# here we use maxTheta

sims = rexp(n=50, rate=maxTheta)

par(mfrow=c(2,1))
hist(sims, main = "Simulations",
     col = "blue", xlim = c(0,5),
```

```
     ylim = c(0,20), xlab = "Simulations")
hist(data$Length, main = "Actual values",
     col = "green", xlim = c(0,5),
     ylim = c(0,20), xlab = "Actual length values")
```






```
par(mfrow=c(1,1))
```

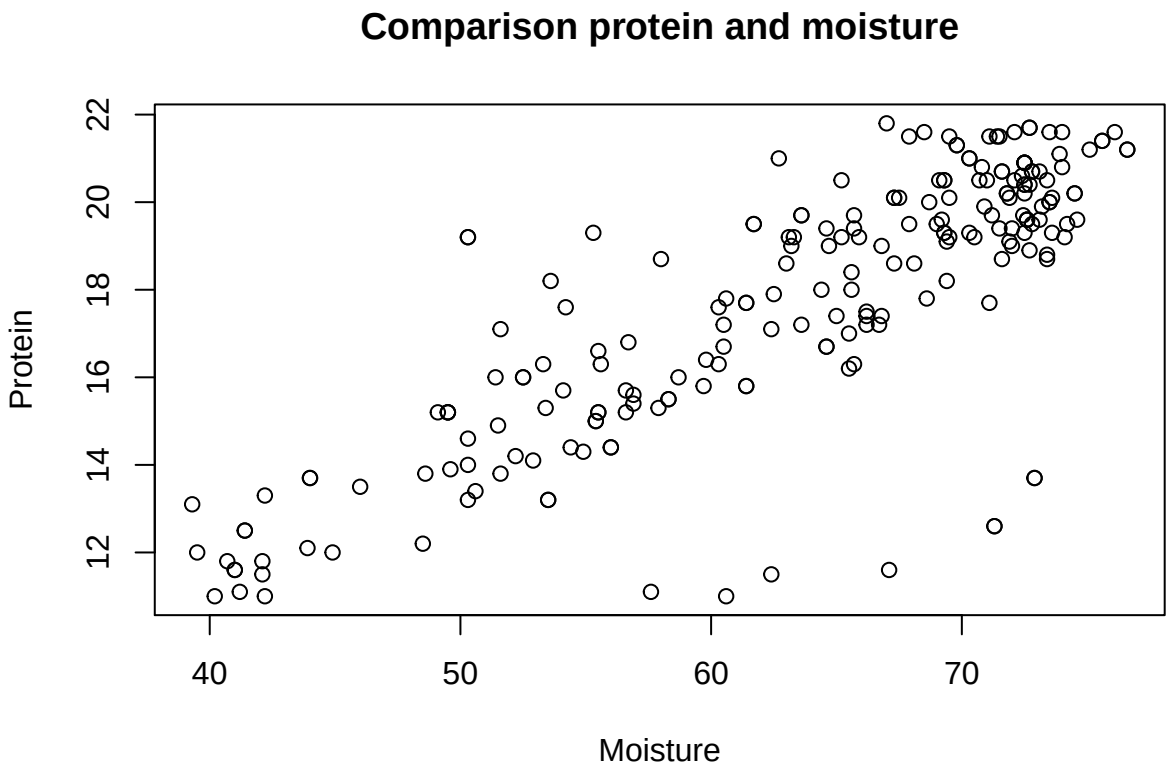
We can see that the actual values and the simulated values follows a similar distribution. This would be clearer with more samples, but is still quite clear with only 50 samples for each.

3 Assignment 4

3.1 Task 1 - Plotting

Plotting the protein versus the moisture clearly shows that the relation between these can be described well using a linear model.

```
# Task 1
library(readxl)
data = read_excel("tecator.xlsx")

plot(x=data$Moisture, y = data$Protein,
     xlab = "Moisture",
     ylab = "Protein",
     main = "Comparison protein and moisture")
```



3.2 Task 2 - Describing a probabilistic model

In this task, we are supposed to describe a probabilistic model that describes M_i. This can be described as Here, it is appropriate to use the MSE criterion, since . The MSE criterion can be described as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

and it is appropriate to use for linear regression, since it comes from the maximum likelihood function if the error is normally distributed, which it is, according to instructions in this case.

To be able to describe different probabilistic models $M_i \quad \forall i \in \{1, .., 6\}$ I created a few functions to be able to generate these models. The models can mathematically be described as

$$M_i = \left\{ y_m = \sum_{j=0}^{i} \beta_0 x^j + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2) \quad \forall m \in \{1, .., m\} \right\}$$

where m is the number of datapoints.

```
# Task 2

prob_model = function(train, test, i) {
  n_train = length(train$Moisture)
  x_train = train$Moisture
  X_train = matrix(c(x_train,
                     x_train^2,
                     x_train^3,
                     x_train^4,
```

```
                    x_train^5,
                    x_train^6),
                  nrow = n_train, ncol = 6)


  n_test = length(test$Moisture)
  x_test = test$Moisture
  X_test = matrix(c(x_test,
                    x_test^2,
                    x_test^3,
                    x_test^4,
                    x_test^5,
                    x_test^6),
                  nrow = n_test, ncol = 6)

  if (i != 6) {
    X_train = X_train[,-((i+1):7)]
    X_test = X_test[,-((i+1):7)]
  }


  y_train = train$Protein
  y_test = test$Protein



  return(list(y_train,X_train, y_test, X_test))
  #return misclassification rate and confusion matrix



}

MSE = function(y, y_hat) {
  n = length(y)
  return((1/n)*sum((y - y_hat)^2))
}

convert_to_poly = function(x, i) {
  n = length(x)
  x_matrix = matrix(c(rep(1,n), x, x^2, x^3, x^4, x^5, x^6), nrow = n, ncol = 7)

  if (i != 6) {
    x_matrix = x_matrix[,-((i+2):7)]
  }


  return(x_matrix)
}
```

3.3 Task 3

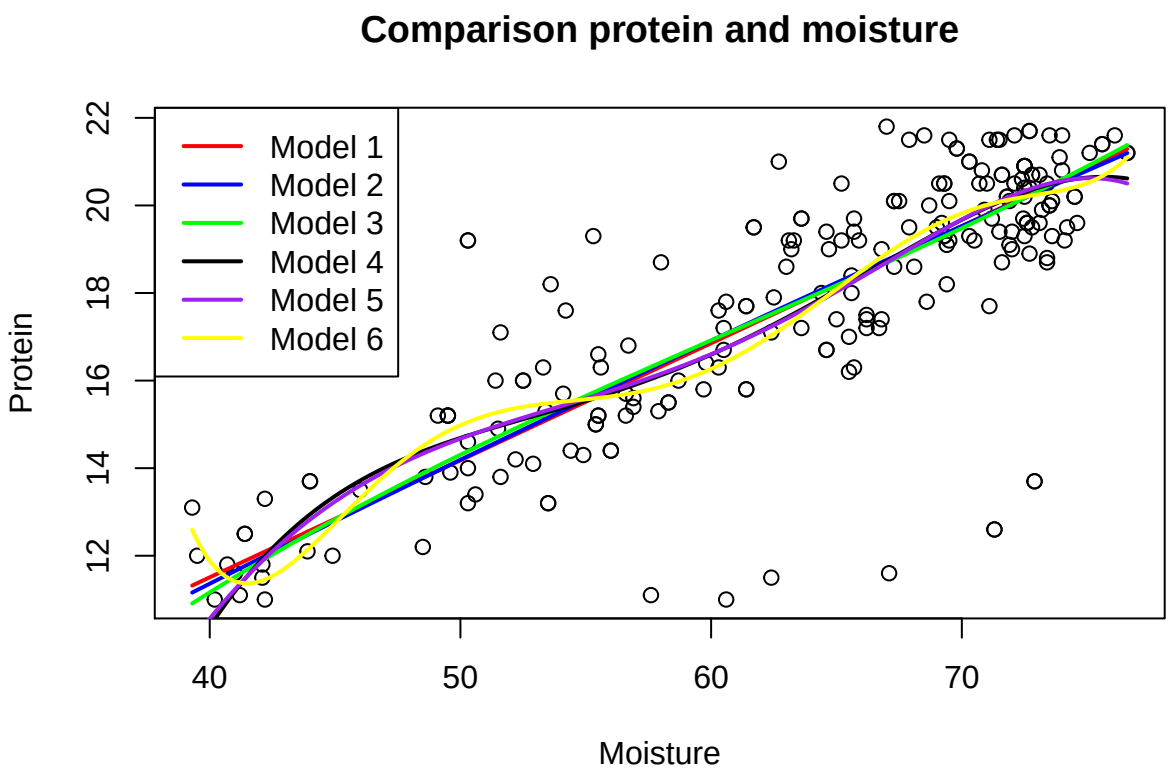
In this task, we use the functions created previously to generate the models and get their MSEs for the training and the data set.

```
plot(x=data$Moisture, y = data$Protein,
     xlab = "Moisture",
     ylab = "Protein",
     main = "Comparison protein and moisture")
# Task 3
# Copying from Assignment 1
n=dim(data)[1]
set.seed(1)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]


MSE_vals = matrix(NA, nrow = 6, ncol = 2)
rownames(MSE_vals) = seq(1,6,1)
colnames(MSE_vals) = c("Training MSE", "Test MSE")


x_seq = seq(min(data$Moisture), max(data$Moisture), length = 1000)
colors = c("red", "blue", "green", "black", "purple", "yellow")
for (i in 1:6) {
  print(i)
  Model = prob_model(train, test, i)
  y_train = Model[[1]]
  X_train = data.frame(Model[[2]])
  y_test = Model[[3]]
  X_test = data.frame(Model[[4]])
  if (i == 1) {
    names(X_test) = c("Model..2..")
  }


  lm.fit = lm(y_train ~., data = X_train)
  trainPredict = predict(lm.fit, X_train)
  testPredict = predict(lm.fit, X_test)

  betas = lm.fit$coefficients


  x_matrix = convert_to_poly(x_seq, i)



  lines(x=x_seq, y = t(as.matrix(betas))%*%t(x_matrix), col = colors[i], lwd = 2)
  MSE_vals[i, 1] = MSE(train$Protein ,trainPredict)
  MSE_vals[i, 2] = MSE(test$Protein, testPredict)
}
```

```
## [1] 1
## [1] 2
```

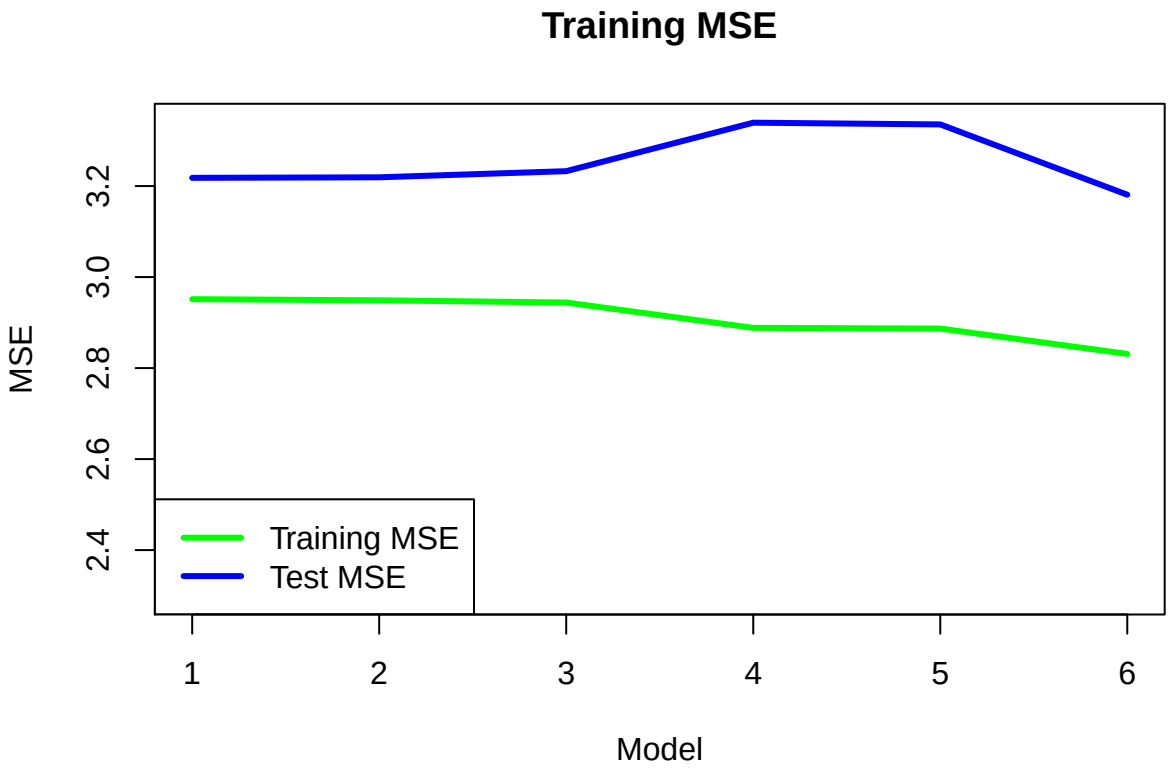
```
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

```
legend("topleft",
       lty = rep(1,6),
       col = colors,
       lwd = c(2,2,2,2,2,2),
       legend = c("Model 1",
                  "Model 2",
                  "Model 3",
                  "Model 4",
                  "Model 5",
                  "Model 6"))
```




```
plot(x=seq(1,6,1), y = MSE_vals[,1],
     col = "green",
     type = "l",
     xlab = "Model",
     ylab = "MSE",
     lwd = 3,
     main = "Training MSE", ylim = c(2.3, max(MSE_vals)))
lines(x=seq(1,6,1), y = MSE_vals[,2],
     col = "blue",
     lwd = 3,
     xlab = "Model",
     ylab = "MSE",
     main = "Test MSE")
legend("bottomleft",
```

```
       lty = rep(1,6),
       col = c("green", "blue"),
       lwd = c(3,3),
       legend = c("Training MSE",
                  "Test MSE"))
```



We clearly see that with a higher polynomial degree, we manage to fit the data better. However, the test set does not necessarily improve, but rather do not follow a clear pattern. Depending on which seed one sets, the MSE results varies, but it does not necessarily improve with higher polynomials. This is because the more polynomials we have, we risk having a greater bias towards the training data.

3.4 Task 4

In task 4, we use variable selection using `stepAIC()`.

```
#Task 4 - use entire data set.

# Perform variable selection of a linear model
# in which fat is a response and Channel1-Channel100 are predictors.
# Comment on how many variables were selected.

library(MASS)
dataToUse = data
#Remove protein and
dataToUse$Protein = NULL
dataToUse$Moisture = NULL
dataToUse$Sample = NULL
lm.fit = lm(Fat ~ ., data = dataToUse)
```

```
step = stepAIC(lm.fit, direction = "both")

nrVarsChosen = length(step$coefficients)
```

Printing the number of variables chosen, we get that 64 variables have been chosen. This is quite a high number, but we managed to at least remove $\frac{1}{3}$ of the variables using the variable selection.

```
print(nrVarsChosen)
```

```
## [1] 64
```

3.5 Task 5

Using the model retrieved through `stepAIC()`, we do a ridge regression.

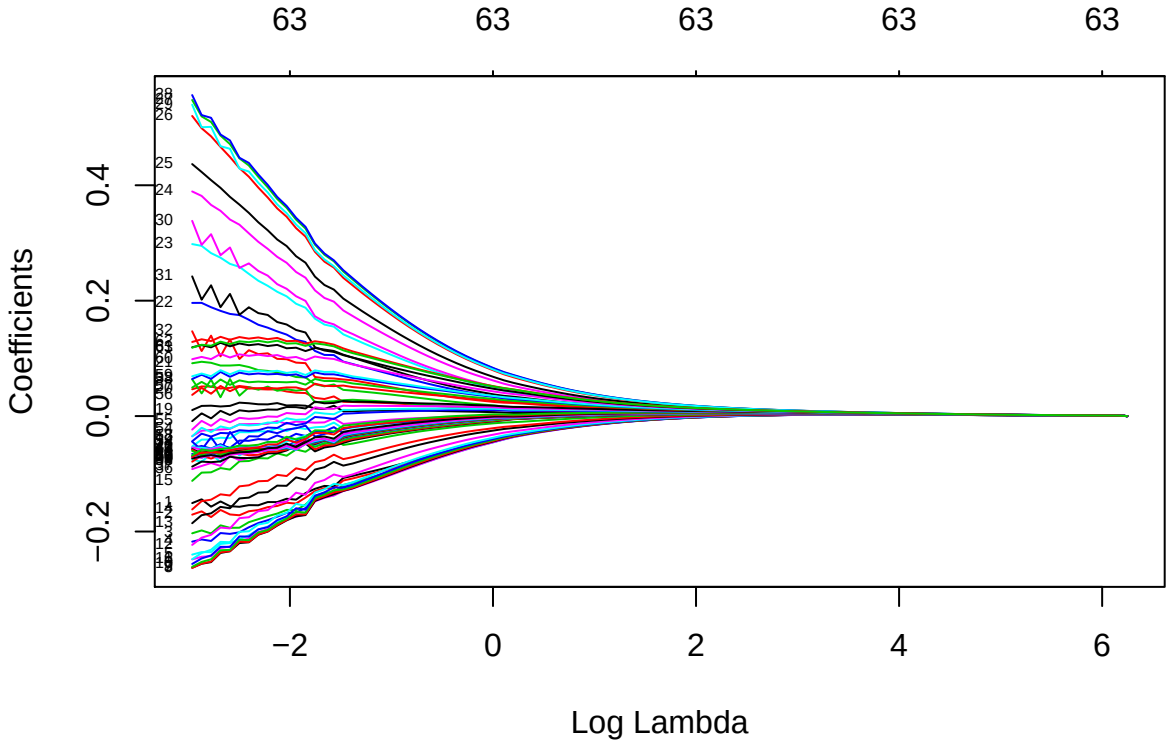
```
# Task 5 - ridge regression
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
variablesSelected = names(step$coefficients)
variablesSelected = variablesSelected[-c(1)]
dataSteps = dataToUse[,variablesSelected]
dataSteps = scale(dataSteps)
response = scale(dataToUse$Fat)
ridgeRegModel = glmnet(as.matrix(dataSteps), response, alpha = 0, family="gaussian")
plot(ridgeRegModel, xvar="lambda", label=TRUE)
```

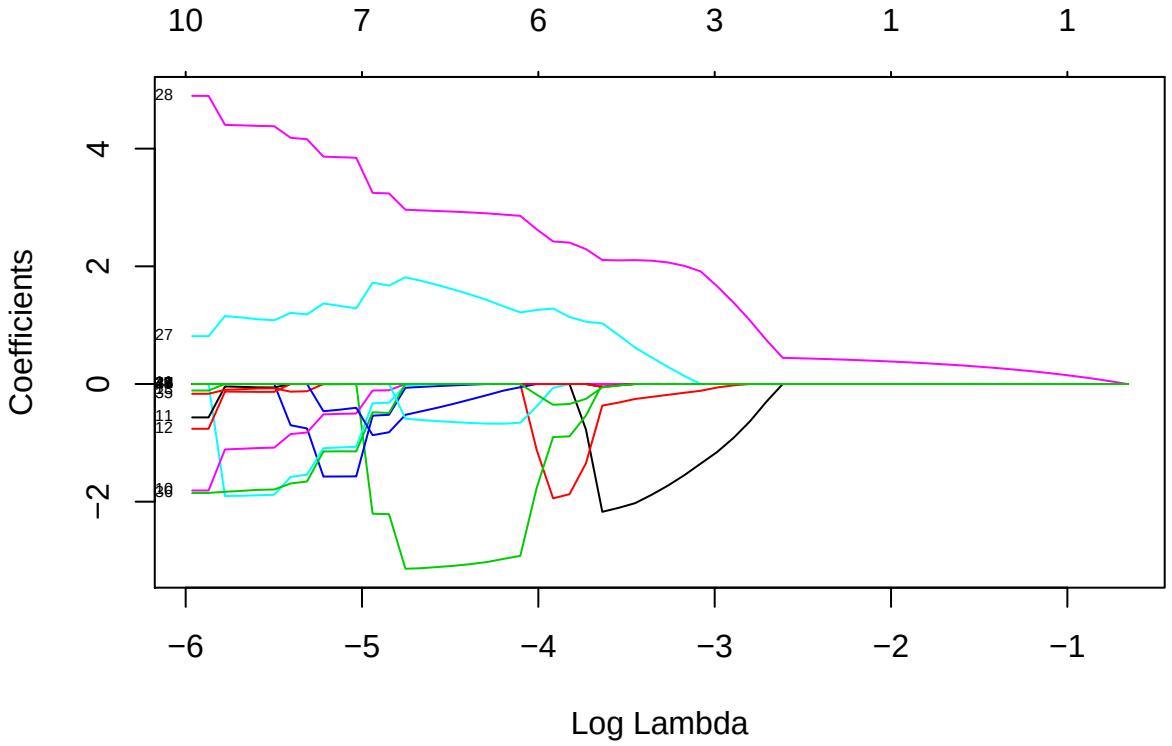


We can clearly that the coefficients are minimized, tending towards 0 the more we regularize. This is because the regularization makes them affect less, limiting their possibility to overfit.

3.6 Task 6

In task 6, we fit a LASSO regression instead of Ridge regression.

```
# Task 6 - Lasso regression
#Here, alpha = 1 instead
lassoModel = glmnet(as.matrix(dataSteps), response, alpha = 1, family="gaussian")
plot(lassoModel, xvar="lambda", label=TRUE)
```



We see here as well that LASSO regression also makes the variables

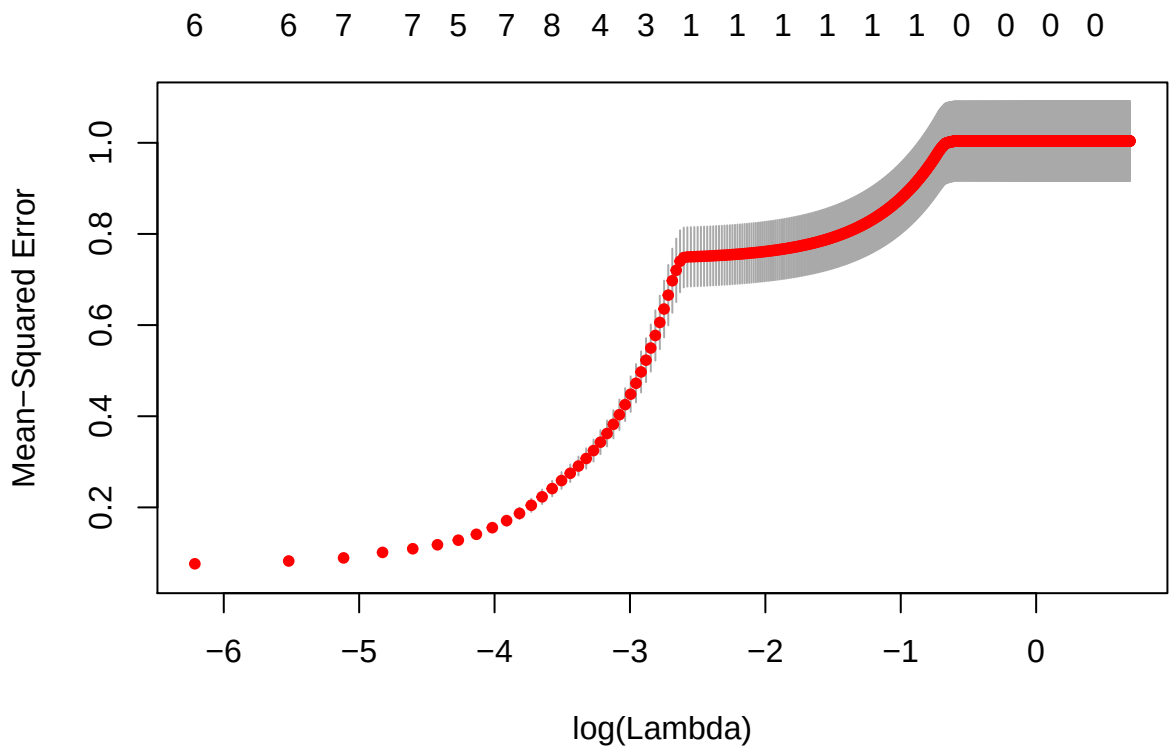
3.7 Task 7

Here, we use the LASSO model combined with cross-validation, to see whether we can reduce the number of variables.

```
# Task 7 - use cross-validation to find the optimal lasso model

lambda = seq(0,2, length = 1000)
lassoCVModel = cv.glmnet(as.matrix(dataSteps), response, lambda = lambda, alpha = 1, family="gaussian")
lassoCVModel$lambda.min
```

```
## [1] 0
```

```
plot(lassoCVModel)
```




```
coef(lassoCVModel, s = "lambda.min")
```

```
## 64 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  4.842088e-15
## Channel1     5.256494e-01
## Channel2     2.321935e-01
## Channel4     6.216016e-01
## Channel5     2.376907e-01
## Channel7     3.065258e+00
## Channel8     8.905658e-02
## Channel11   -2.195373e+00
## Channel12    3.173341e-01
## Channel13   -1.598259e-01
## Channel14   -2.266648e-01
## Channel15    9.888194e-01
## Channel17   -2.452527e+00
## Channel19   -1.999273e+00
## Channel20   -5.810896e-01
## Channel22   -1.730800e+00
## Channel24   -2.117293e-01
## Channel25    1.025985e-01
## Channel26    1.369203e-01
## Channel28    8.954074e-02
## Channel29   -5.670383e-02
## Channel30   -1.653193e-01
## Channel32   -7.042347e-01
## Channel34   -5.421359e-01
## Channel36    2.227688e-01
## Channel37    3.226586e-01
## Channel39    5.751797e-01
```

```
## Channel40   -3.197614e-02
## Channel41    6.697044e+00
## Channel42    1.081395e-01
## Channel45   -6.873826e-02
## Channel46   -3.740014e-01
## Channel47   -4.685819e-01
## Channel48   -3.565384e-01
## Channel50    4.847936e-01
## Channel51    6.563343e-01
## Channel52   -4.475029e+00
## Channel54   -4.199649e-01
## Channel55    7.574294e-01
## Channel56    9.247664e-01
## Channel59    9.858683e-01
## Channel60    5.550758e-02
## Channel61   -8.188875e-02
## Channel63   -4.529709e-01
## Channel64    3.919912e+00
## Channel65   -5.485933e-01
## Channel67   -9.367555e-01
## Channel68   -3.629482e-01
## Channel69   -4.893397e-01
## Channel71   -1.097997e+00
## Channel73   -8.031498e-01
## Channel74   -5.296899e-01
## Channel78   -8.293671e-01
## Channel79   -1.030879e-01
## Channel80   -2.450470e-02
## Channel81   -2.861823e-02
## Channel84   -7.440048e-02
## Channel85    1.924310e-02
## Channel87    1.768470e-01
## Channel88    1.472838e-01
## Channel92    5.477630e-01
## Channel94    2.885280e-01
## Channel98    6.280698e-01
## Channel99    1.304106e-01
```

The LASSO model manages to reduce the variables to only 11, compared to 64 by `stepAIC`.

3.8 Task 8

While we saw that `stepAIC` reduced the number of variables to 64, LASSO combined with cross-validation managed to reduce it to only 11, indicating that the LASSO with crossvalidation is better at reducing the number of features.