TDDE01 Lab1

Richard Friberg, Ruben Hillborg, Anton Gefvert

11/26/2018

Assignment 1 - spam classification

1.2.

```
Classification principle:
1 if p(Y = 1|X) > 0.5 else 0
```

For training data:

```
##          Truth
## Predicted   0   1
##         0 803  81
##         1 142 344
```

```
## [1] "Missclassification rate: 0.162774"
```

For test data:

```
##          Truth
## Predicted   0   1
##         0 791  97
##         1 146 336
```

```
## [1] "Missclassification rate: 0.177372"
```

This is not at spam-filter I would like to use. A lot of emails that is not spam is predicted as spam and quite a high percentage of the emails that are spam gets through, which is not as bad but would be a bit irritating.

We can see that the classification rates are similar for both the training and test data. This implies that the regression model atleast made a good attempt at generalising its spam detection functionality and thus not overfitting the training data.

1.3.

```
Classification principle:
1 if p(Y = 1|X) > 0.9 else 0
```

For training data:

```
##          Truth
## Predicted   0   1
##         0 944 419
##         1   1   6
```

```
## [1] "Missclassification rate: 0.306569"
```

For test data:

```
##          Truth
## Predicted   0   1
##         0 936 427
##         1   1   6
```

```
## [1] "Missclassification rate: 0.312409"
```

The new stricter rule made it so that the filter behaves very cowardly, barely marking any of the emails as spam. Can't quite call it a spam filter if it doesn't filter out the spam.

4. knn with k = 30

For training data:

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

```
## [1] "Missclassification rate: 0.172263"
```

For test data:

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

```
## [1] "Missclassification rate: 0.329927"
```

In comparision to task 2 above, where we used linear regression, you can see that the missclassifcation rates are about the same using only the training set. But when we switch over to comparing the validation using the training data, we can see that in task 2 the model was a much better attempt at generalising the spam filtering. This as the missclassification rate only rose about 1 % compared to doubling we saw here, from 17% to 33%.

5. knn with k = 1

For training data:

```
##          Truth
## Predicted   0   1
##         0 945   0
##         1   0 425
```

```
## [1] "Missclassification rate: 0.000000"
```

For test data:

```
##          Truth
## Predicted   0   1
##         0 559 258
##         1 386 167
```

```
## [1] "Missclassification rate: 0.345985"
```

Using k=1 will result in the model overfitting the data, as it clearly does when "predicting" everything correctly using only the training data. The flaws of using k=1 only comes up when trying to validate the model using the test data, where the missclasification rate goes from zero to about 35% thus not producing a very good generalisation.

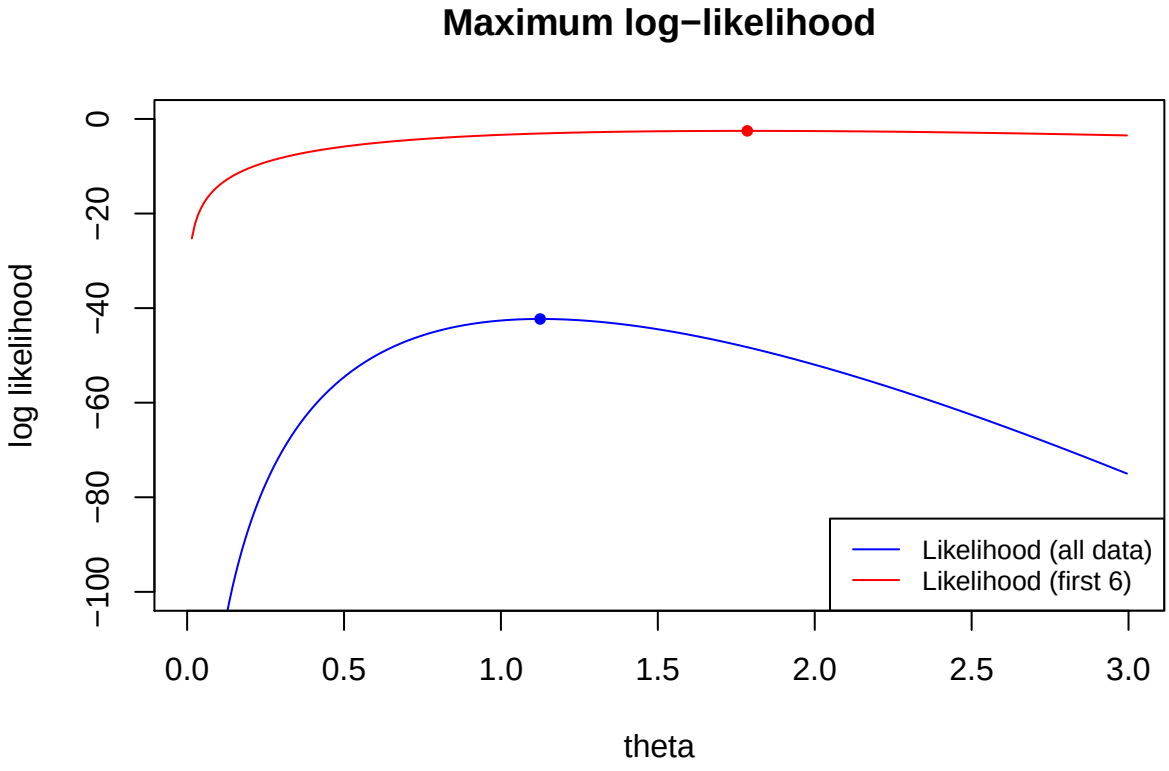
Assignment 2. Inference about lifetime of machines

In this assignment a file containing the lifetime of 48 machines was given. The random variable for these lifetimes was called Length.

2.1-2.3 Maximum log-likelihood

In the assignment we were to assume that the probability model for the data was $p(x|\theta) = \theta e^{-\theta x}$ for $\mathbf{x}$=*Length*, and that the observations were independent and identically distributed (iid). Given the assumed model, and that the data is time until an event happens, we know that the data is exponentially distributed. A function that calculates the log-likelihood $log\ p(x|\theta)$ of an exponential distribution, for a given θ and a given vector $\mathbf{x}$, was written.

The dependence of log-likelihood on θ was then plotted and the maximum likelihood could be extracted from the graph. The same was also done for only the 6 first observations in Length. The results can be seen in the plot below, where the blue line is the likelihoods dependence on θ when using all observations from the data, and the red line is for the first 6 observations only. The maximum likelihood is marked with a dot.




```
## [1] "Maximum likelihood value of theta for blue line: 1.125000"
```

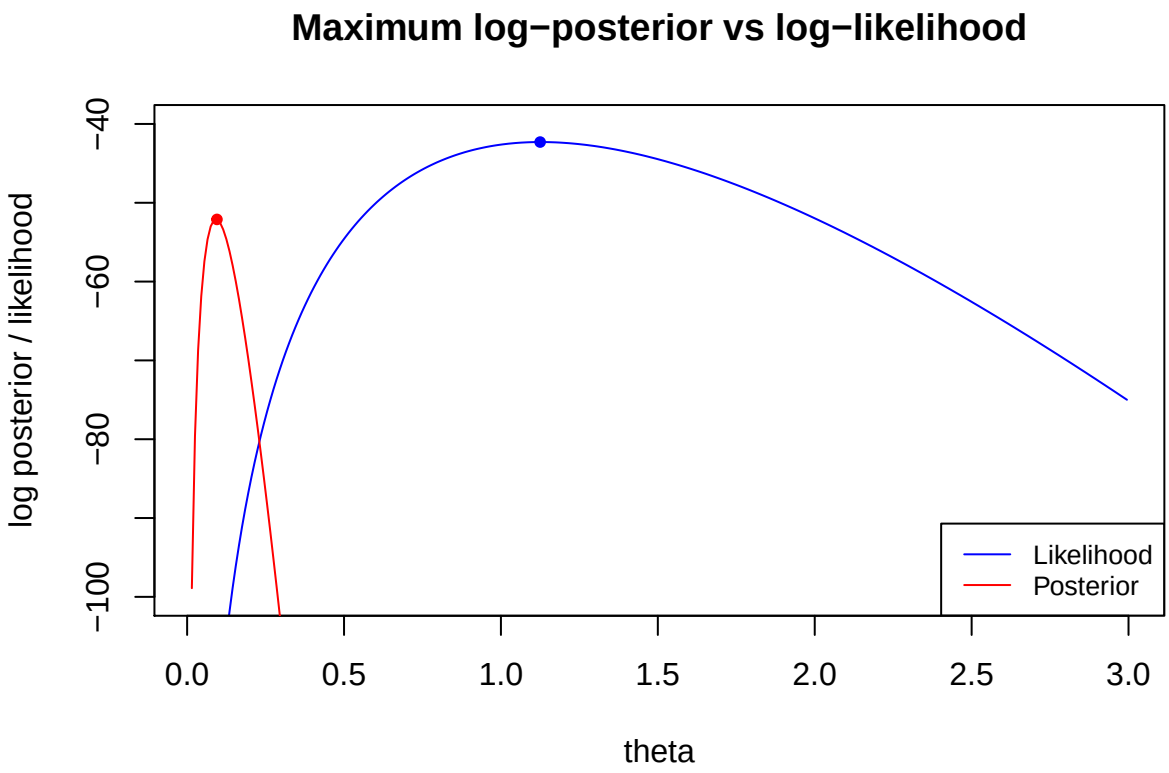
```
## [1] "Maximum likelihood value of theta for red line: 1.785000"
```

The estimated theta for all of the data is probably more reliable, since it's based on 48 data values while the other is only based on 6 values. These 6 values could give a missleading lifetime mean to represent the expected lifetime of all machines.

2.4 Bayesian maximum log likelihood

We now assumed the Bayesian model $p(x|\theta) = \theta e^{-\theta x}$ with the prior $p(\theta) = \lambda e^{-\lambda\theta)}, \lambda = 10$. A new function was written that calculates $l(\theta) = log(p(x|\theta)p(\theta))$ for a given θ and $\mathbf{x}$.

This function computes the log of the posterior, and the resulting dependence of $l(\theta)$ on θ when used on all of the data can be seen in the plot below. The line is plotted next to the line for log-likelihood on all data for the last section.



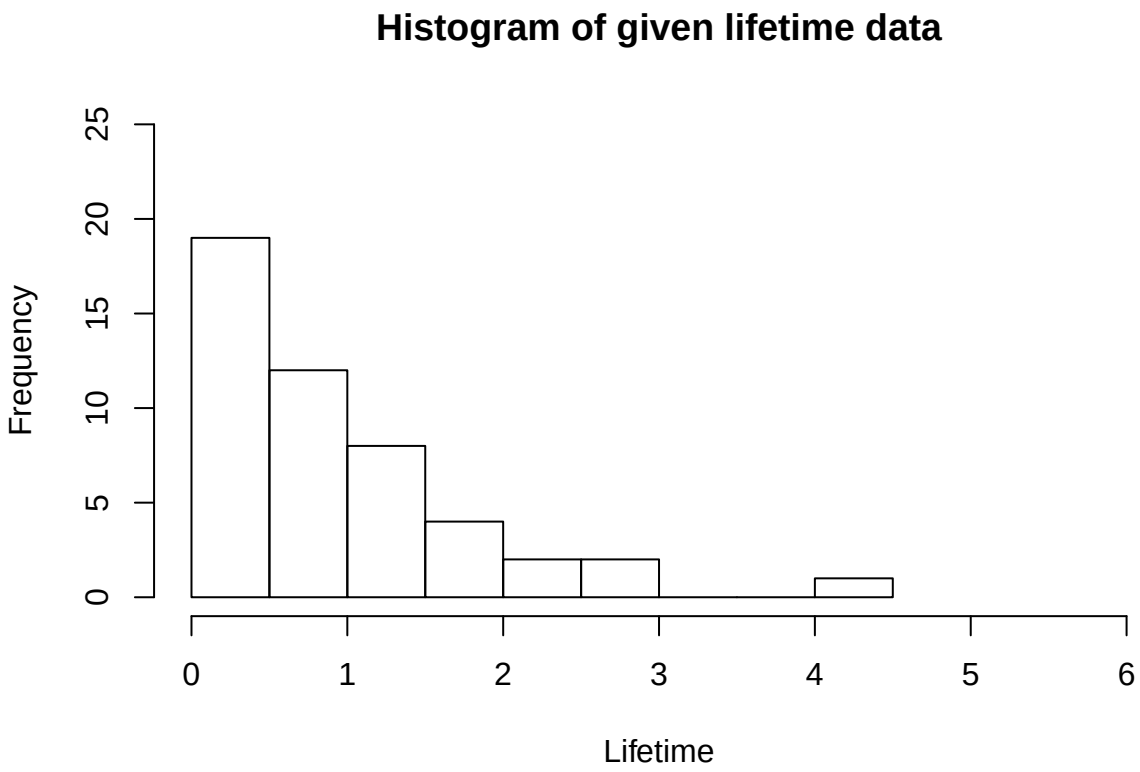

```
## [1] "Maximum likelihood value of theta for blue line: 1.125000"
```

```
## [1] "Maximum likelihood value of theta for red line: 0.095000"
```

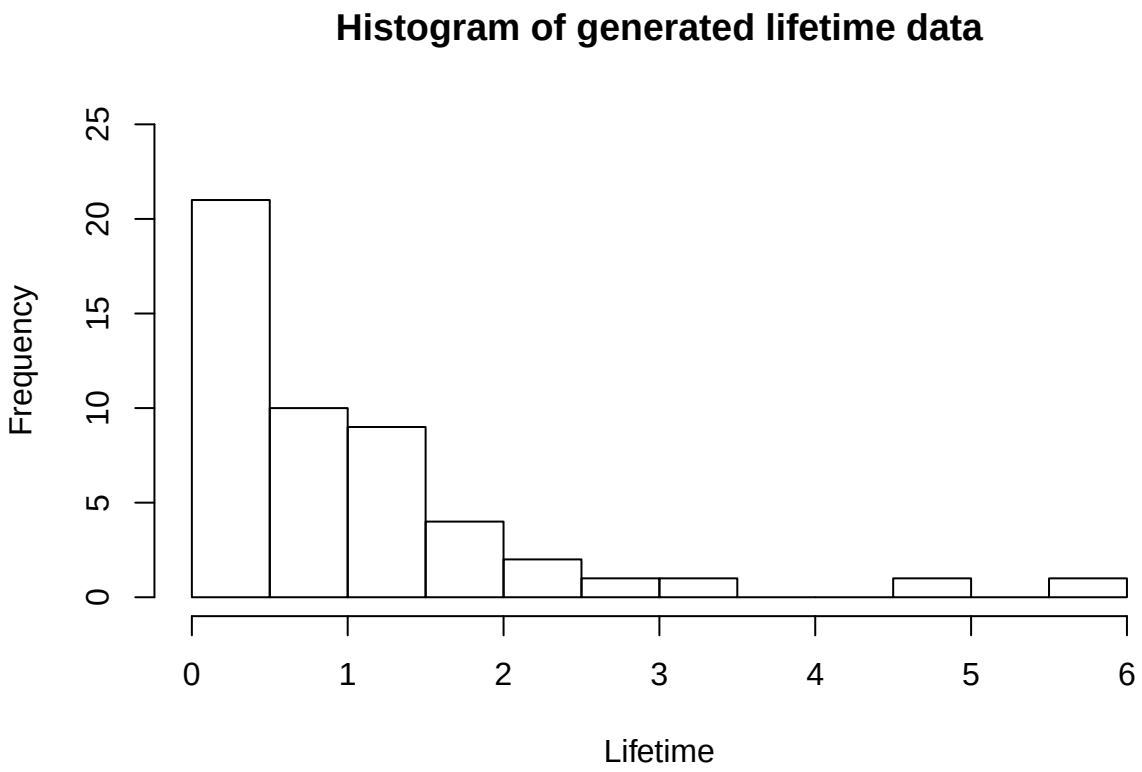
From the plot we can see that given the new prior knowledge of the probability for θ, the maximum likelihood of θ has been shifted to the left. This happens because the prior promotes lower values of θ to be more likely.

2.5 Testing theta

To see if the estimated theta found in section 2.2 (using all data and no prior), 50 observations were generated from the exponential distribution using the estimated theta ($Exp(\hat{\theta})$). The distribution of the generated data was then compared to the actual data given in the assignment using histograms. The histograms can be ob-



served below:



The distribution of the histograms looks very similar to each other, indicating that the estimated theta is close to the real

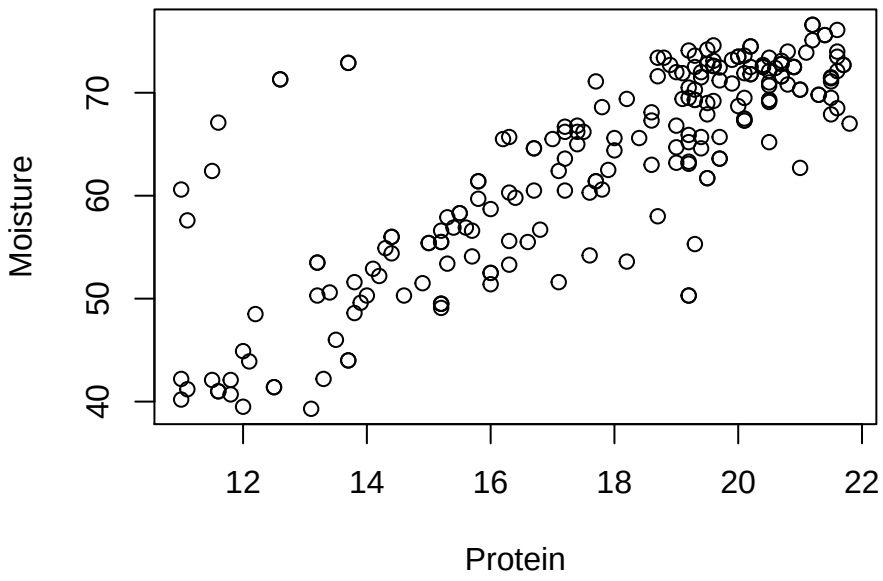


Figure 1: Moisture vs protein

Assignment 4

4.1

Looking at Figure 1, most of the datapoints are located in a basically linear line, which implies the model is described well by a linear model.

4.2

A model that describes M_i can be written as

$$y \sim N\left(\sum_{n=0}^{i} w_i x^i, \sigma^2\right)$$

where x is the Protein data, y is the Moisture and w_0 to w_i are weights.

MSE is appropriate because MSE gives exponentially higher error the bigger the error, this means an overfitted model which work bad on validation data will give a larger error, a decently fitted model will give a smaller error. Also, for a gaussian model, minimizing the MSE is equivalent to maximizing loglikelihood!

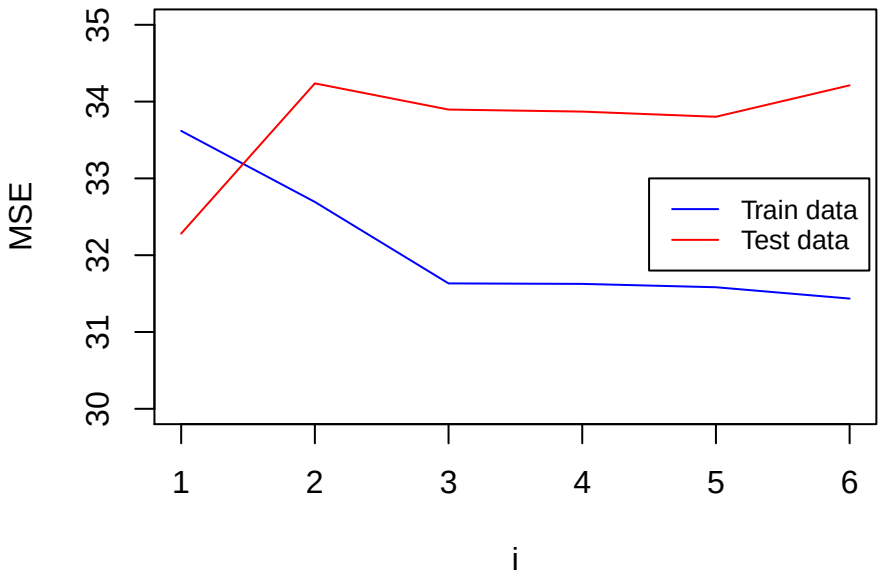


Figure 2: MSE for diffrent i in M_i

4.3

As seen if Figure 2 we see that model $i = 1$ has lowest MSE for the test data, and should thus be selected.

We can see that after $i = 2$ the MSE for test data becomes higher while train data gets lower, this is probably due to the more advanced models overfitting the training data (As we can see in Figure **??**, the solution looks very linear)

4.4

The amount of variables selected is when using stepAIC is 63.

4.5

As seen in Figure 3, we see that when lambda gets larger, the coeffiecents converge towards zero.

4.6

When we compare LASSO in Figure 4 with ridge regression, we see that when using ridge, no coefficents are removed (top axis of graph), these merely get very close to zero, whereas when using lasso, some coeffiecents gets set to 0 and are thus not used at all (which simplifies the model).

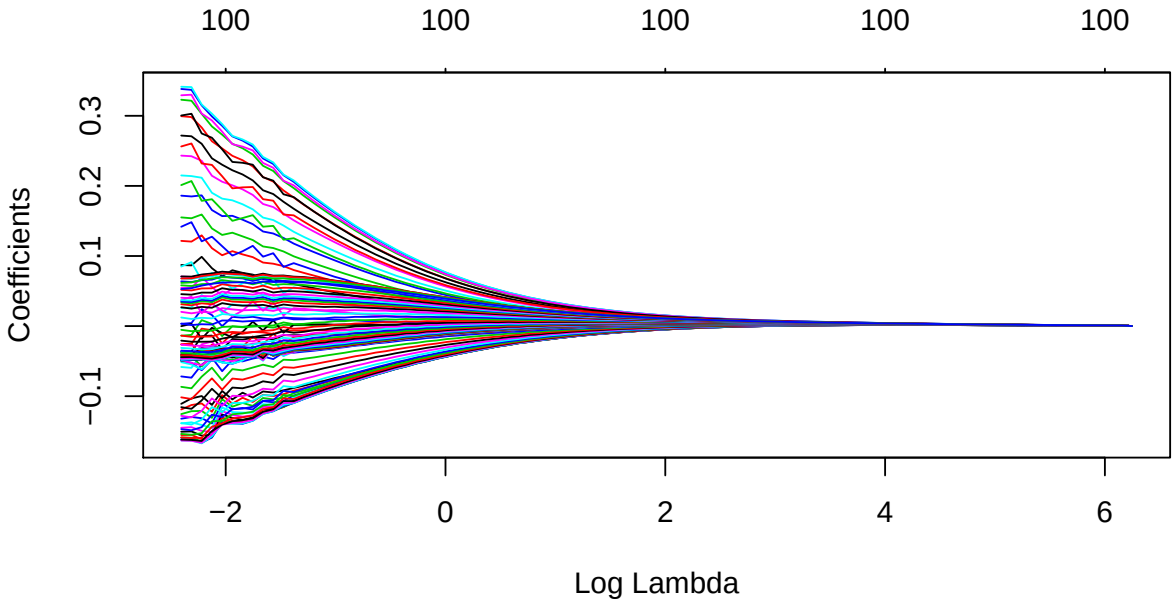


Figure 3: ridge coefficients over lambda

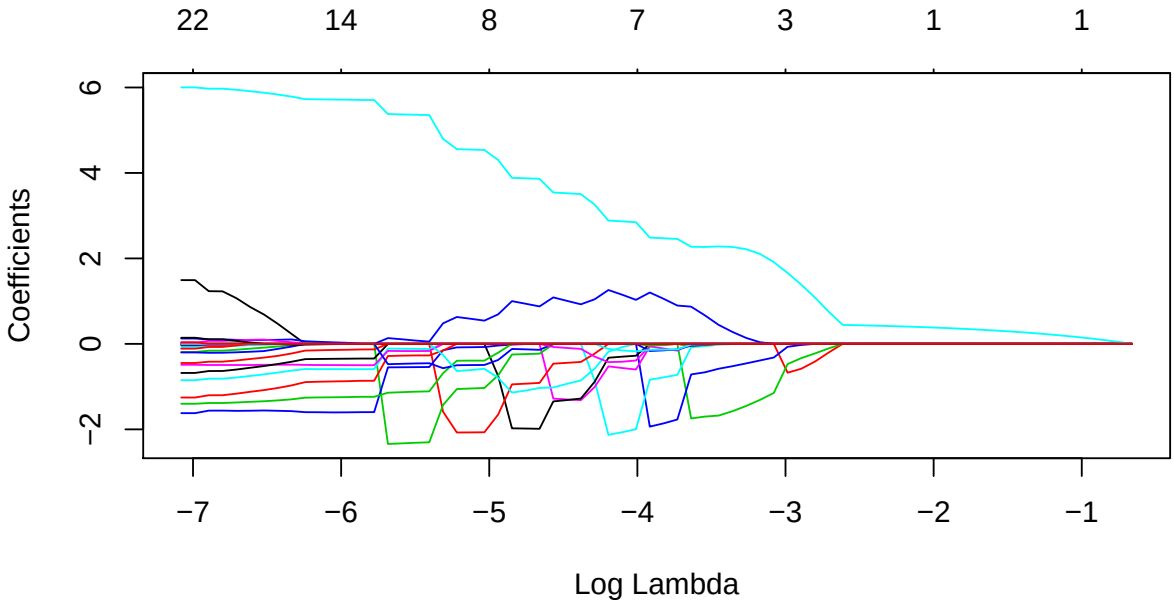


Figure 4: lasso coefficients over lambda

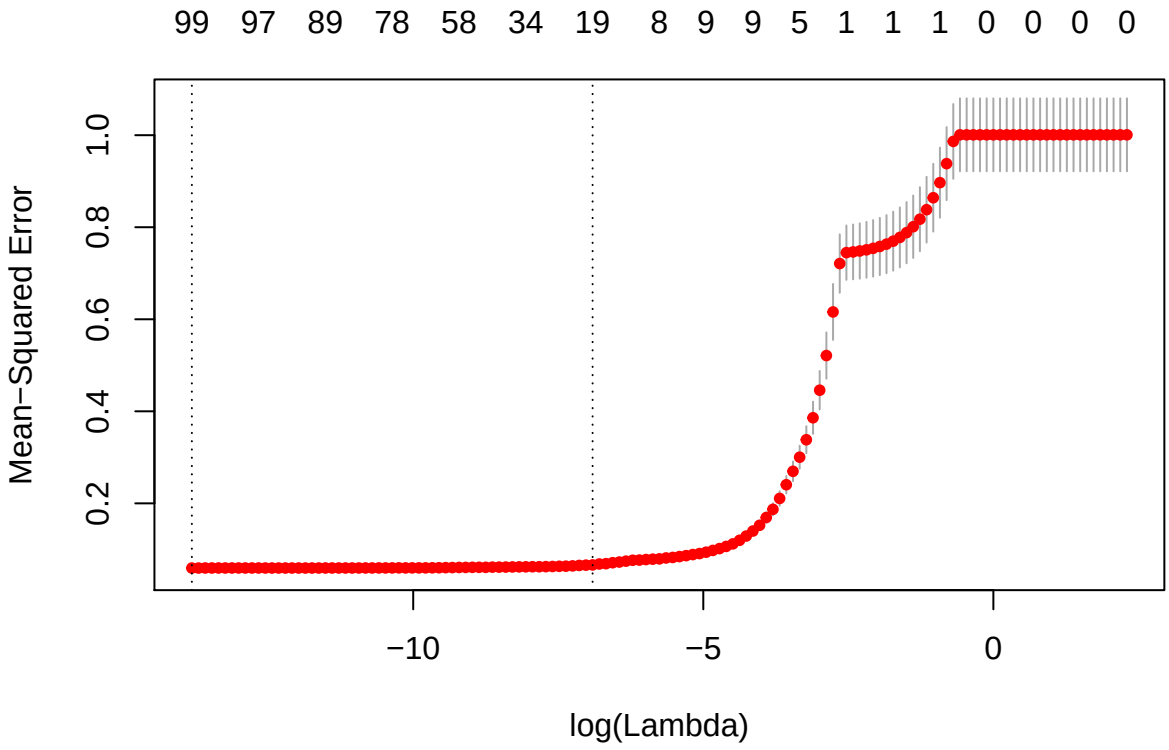


Figure 5: Cross-validation of LASSO

4.7

When doing cross-validation, the optimal value (minimum MSE), is given when $\lambda = 0$ and it gives 100 parameters.

If we look at Figure 5, we see that MSE increases with the increase of λ. As we saw earlier, the optimal $\lambda = 0$, is also indicative of this trend (as the minimium MSE should be the lowest value possible, in this case 0).

4.8

When using CV on LASSO, we use all coefficents, whereas in stepAIC we only use 63 coeffiecents.

Code appendix

```
knitr::opts_chunk$set(echo = FALSE)
library(readxl)
library(glmnet)
library(kknn)
library(knitr)
require(MASS)

# 1.1 import data
data = read_excel("spambase.xlsx")

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

#1.2 training
model_train = glm(formula = Spam ~., family = binomial(link = "logit"), data = train)
prediction_train = predict(model_train, type = "response")
result_train_5 = ifelse(prediction_train > 0.5, 1, 0) # 1 om större än 0.5, 0 annars. (räknas som spam
with(train, table(result_train_5, Spam, dnn = c("Predicted", "Truth"))) # confusion matrix
missclassification_rate_train_5 = mean(result_train_5 != train$Spam)
sprintf("Missclassification rate: %f", missclassification_rate_train_5)

#1.2 test
prediction_test = predict(model_train, test, type = "response")
result_test_5 = ifelse(prediction_test > 0.5, 1, 0) # 1 om större än 0.5, 0 annars. (räknas som spam om
with(test, table(result_test_5, Spam, dnn = c("Predicted", "Truth"))) # confusion matrix
missclassification_rate_test_5 = mean(result_test_5 != test$Spam)
sprintf("Missclassification rate: %f", missclassification_rate_test_5)

#1.3 training
result_train_9 = ifelse(prediction_train > 0.9, 1, 0) # 1 om större än 0.5, 0 annars. (räknas som spam
with(train, table(result_train_9, Spam, dnn = c("Predicted", "Truth"))) # confusion matrix
missclassification_rate_train_9 = mean(result_train_9 != train$Spam)
sprintf("Missclassification rate: %f", missclassification_rate_train_9)

#1.3 test
result_test_9 = ifelse(prediction_test > 0.9, 1, 0) # 1 om större än 0.5, 0 annars. (räknas som spam om
with(test, table(result_test_9, Spam, dnn = c("Predicted", "Truth"))) # confusion matrix
missclassification_rate_test_9 = mean(result_test_9 != test$Spam)
sprintf("Missclassification rate: %f", missclassification_rate_test_9)

#3.4 training
kknn_model = kknn::kknn(formula = Spam ~., train = train, test = train, k = 30)
result_train_5k = ifelse(kknn_model$fitted.values > 0.5, 1, 0)
with(test, table(result_train_5k, train$Spam, dnn = c("Predicted", "Truth"))) # confusion matrix
missclassification_rate_train_k30 = mean(result_train_5k != train$Spam)
sprintf("Missclassification rate: %f", missclassification_rate_train_k30)

#3.4 test
```

```r
knn_model = kknn::kknn(formula = Spam ~., train = train, test = test, k = 30)
result_test_5k = ifelse(knn_model$fitted.values > 0.5, 1, 0)
with(test, table(result_test_5k, test$Spam, dnn = c("Predicted", "Truth"))) # confusion matrix
missclassification_rate_test_k30 = mean(result_test_5k != test$Spam)
sprintf("Missclassification rate: %f", missclassification_rate_test_k30)

#3.5 training
kknn_model_2 = kknn::kknn(formula = Spam ~ ., train = train, test = train, k = 1)
result_test_5 = ifelse(kknn_model_2$fitted.values > 0.5, 1, 0)
with(train, table(result_test_5, Spam, dnn = c("Predicted", "Truth"))) # confusion matrix
missclassification_rate_test_5 = mean(result_test_5 != train$Spam)
sprintf("Missclassification rate: %f", missclassification_rate_test_5)

#3.5 test
kknn_model_2 = kknn::kknn(formula = Spam ~ ., train = train, test = test, k = 1)
result_test_5 = ifelse(kknn_model_2$fitted.values > 0.5, 1, 0)
with(train, table(result_test_5, Spam, dnn = c("Predicted", "Truth"))) # confusion matrix
missclassification_rate_test_5 = mean(result_test_5 != test$Spam)
sprintf("Missclassification rate: %f", missclassification_rate_test_5)

#2.1
data = read_excel("machines.xlsx")

#2.3
log_likelihood = function(theta, x) {
  n = length(x)
  n*log(theta) - theta*sum(x)
}

x = seq(0.015, 3, 0.01)
y1 = log_likelihood(x, data$Length)
y2 = log_likelihood(x, data$Length[1:6])
plot(x, y1,
     ylim = c(-100, 0),
     type="l",
     col="blue",
     main = "Maximum log-likelihood",
     xlab = "theta",
     ylab = "log likelihood")
lines(x, y2, col="red")
theta_hat1 = x[which.max(y1)]
theta_hat2 = x[which.max(y2)]
points(x = theta_hat1, y = max(y1), col = "blue", pch = 20)
points(x = theta_hat2, y = max(y2), col = "red", pch = 20)
legend("bottomright", legend=c("Likelihood (all data)", "Likelihood (first 6)"),
       col=c("blue", "red"), lty=1:1, cex=0.8)
sprintf("Maximum likelihood value of theta for blue line: %f", theta_hat1)
sprintf("Maximum likelihood value of theta for red line: %f", theta_hat2)

#2.4
log_posterior = function(theta, x) {
  n = length(x)
  lambda = 10
```

```
  n*log(lambda) + n*log(theta) - theta*(sum(x)+lambda*n)
}

y3 = log_posterior(x, data$Length)
plot(x, y1,
     ylim = c(-100, -40),
     type="l",
     col="blue",
     main = "Maximum log-posterior vs log-likelihood",
     xlab = "theta",
     ylab = "log posterior / likelihood")
lines(x, y3, col="red")
theta_hat1 = x[which.max(y1)]
theta_hat3 = x[which.max(y3)]
points(x = theta_hat1, y = max(y1), col = "blue", pch = 20)
points(x = theta_hat3, y = max(y3), col = "red", pch = 20)
legend("bottomright", legend=c("Likelihood", "Posterior"),
       col=c("blue", "red"), lty=1:1, cex=0.8)
sprintf("Maximum likelihood value of theta for blue line: %f", theta_hat1)
sprintf("Maximum likelihood value of theta for red line: %f", theta_hat3)

#2.5
set.seed(12345)
rdata = rexp(50, theta_hat1)
hist(data$Length,
     main = "Histogram of given lifetime data",
     xlab = "Lifetime",
     xlim = c(0, 6),
     ylim = c(0, 25),
     breaks = 10)
hist(rdata,
     main = "Histogram of generated lifetime data",
     xlab = "Lifetime",
     xlim = c(0, 6),
     ylim = c(0, 25),
     breaks = 10)

#4.1
data3 <- read_excel("tecator.xlsx")


plot(data3$Protein, data3$Moisture, type="n", xlab="Protein", ylab="Moisture")
points(data3$Protein, data3$Moisture)

#4.3
n=dim(data3)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train3 = data3[id,]
test3 = data3[-id,]

mses = data.frame(Train_MSE=double(),
                  Test_MSE=double(),
```

```
                  stringsAsFactors=FALSE)
for(i in seq(1,6)){
    trained = lm(Moisture ~ poly(Protein, i), data=train3)
    mses[i,"Train_MSE"] = mean((train3$Moisture - predict(trained))^2)
    mses[i,"Test_MSE"] = mean((test3$Moisture - predict(trained, newdata=test3))^2)
}

plot(seq(1,6), mses$Test_MSE, type="n", ylim=c(30,35), ylab="MSE", xlab="i")
lines(seq(1,6), mses$Train_MSE, col="blue")
lines(seq(1,6), mses$Test_MSE, col="red")
legend(4.5, 33, legend=c("Train data", "Test data"),
       col=c("blue", "red"), lty=1, cex=0.8)

#4.4
data3[,1:102]
trained = glm(Fat~., data=data3[,2:102])
stepped = stepAIC(trained)
num_params = length(stepped$model - 1)

#4.5
covariates = scale(data3[2:101])
response = scale(data3$Fat)
ridge = glmnet(as.matrix(covariates), response, alpha=0, family="gaussian")
plot(ridge, xvar="lambda")

#4.6
lasso = glmnet(as.matrix(covariates), response, alpha=1, family="gaussian")
plot(lasso, xvar="lambda")

#4.7
cv1 = cv.glmnet(as.matrix(covariates), response, alpha=1,
                family="gaussian", lambda=c(0, 10^seq(-6, 1, 0.05)))
plot(cv1)
```