TDDE01 - Lab 1

Ludvig Westerdahl (ludwe631)
Joel Almqvist (joeal360)
Styrbjörn Swensson (stysw526)

2019-11-22

Assignment 1: Spam classification with nearest neighbors

2. Classify training and test data

The classification principle can be seen in Eq. 1.

$$\hat{Y} = \begin{cases} 1 \text{ (Spam)}, & \textit{if } p(Y = 1 \mid X) > 0.5 \\ 0 \text{ (Not spam)}, & \textit{otherwise.} \end{cases} \tag{1}$$

Training data

Table 1: Training data confusion matrix. X is Predicted and Y is Gold.

	0	1
0	804	127
1	93	346

We can calculate the missclassification rate by dividing the total in the table with the sum of the diagonal.

```
Missclassification rate is 0.160583941605839
```

Test data

Now we apply the same principle but on the test set instead.

Table 2: Test data confusion matrix. X is Predicted and Y is Gold.

	0	1
0	808	143
1	92	327

```
Missclassification rate is 0.171532846715328
```

3. Classify training and test data with different classification principle

The new classification principle can be seen in Eq. 2.

$$\hat{Y} = \begin{cases} 1 \text{ (Spam)}, & \textit{if } p(Y = 1 \mid X) > 0.8 \\ 0 \text{ (Not spam)}, & \textit{otherwise.} \end{cases} \tag{2}$$

Training data

Table 3: Training data confusion matrix. X is Predicted and Y is Gold.

	0	1
0	921	10
1	333	106

```
Missclassification rate is 0.25036496350365
```

Test data

Table 4: Test data confusion matrix. X is Predicted and Y is Gold.

	0	1
0	931	20
1	314	105

```
Missclassification rate is 0.243795620437956
```

We can note that the classifier became a lot more careful in classifying mails as spam which resulted in a gross increase in the missclassification rate. Where the biggest went to missclassified spam mails almost by 200, while we did increase some correctly classified normal mails (Not spam). It can be argued whether or not this is a good thing. One one hand, it doesn't hide as many normal mails as spam. On the other hand, more spam mails gets through.

4. KNN classification

In this experiment we are using K = 30.

Training data

Table 5: Training data confusion matrix. X is Predicted and Y is Gold.

	0	1
0	779	152
1	77	362

```
Missclassification rate is 0.167153284671533
```

Test data

Table 6: Test data confusion matrix. X is Predicted and Y is Gold.

	0	1
0	702	249
1	180	239

```
Missclassification rate is 0.313138686131387
```

Here we can see that using KNN we have managed to have the same missclassification rate on the training data. However, the increase is missclassification increased by almost 15 percentage points while the logistic regression method only increased with a single percentage point. This means that the KNN model does not generalize as well on this data as the logistic regression model does.

5. KNN classification

In this experiment we are using K = 1.

Training data

Table 7: Training data confusion matrix. X is Predicted and Y is Gold.

	0	1
0	931	0
1	0	439

```
Missclassification rate is 0
```

Test data

Table 8: Test data confusion matrix. X is Predicted and Y is Gold.

	0	1
0	644	307
1	185	234

```
Missclassification rate is 0.359124087591241
```

Because we are only considering one neighbor, it is expected that we get 0 missclassifications when we consider the training data. We also managed to increase the test classification rate by around 5 percentage points, which means this model is even worse in generalizing on newly seen data. One can speculate that this information tells us that the data overlaps in some areas, that we actually should consider more neighbors to find out what class a data point belongs to.

Assignment 2: Inference about lifetime of machines

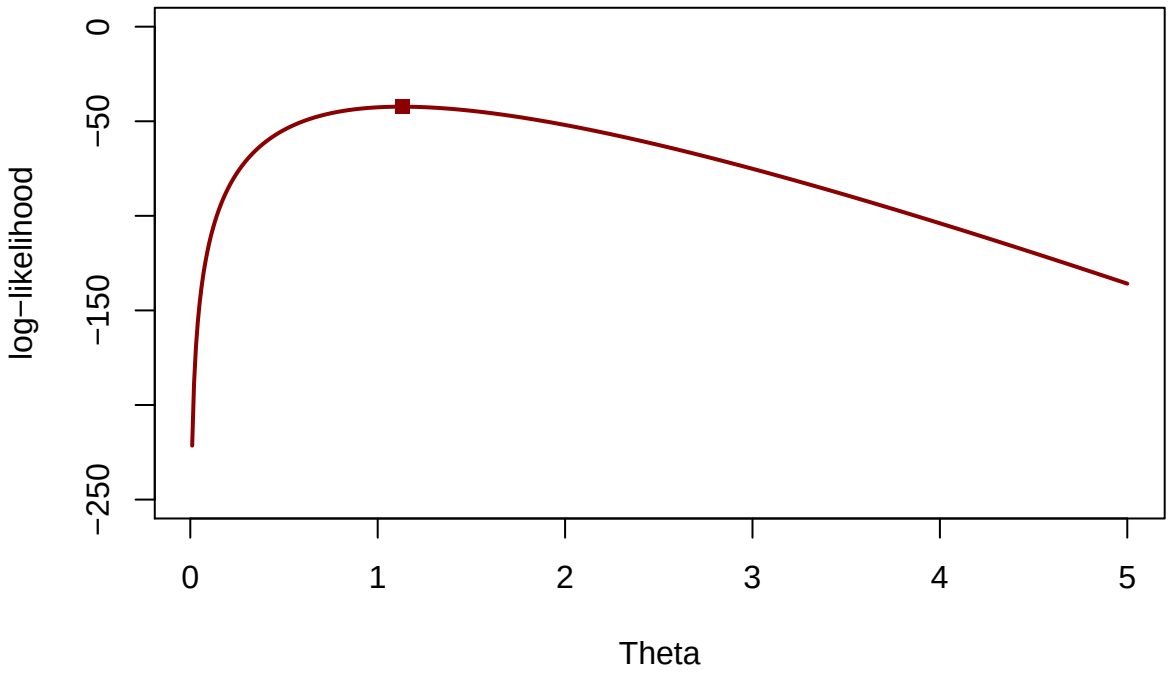
2. Probaility model and fitting it to data

Probability model

We are considering a model defined by Eq. 3.

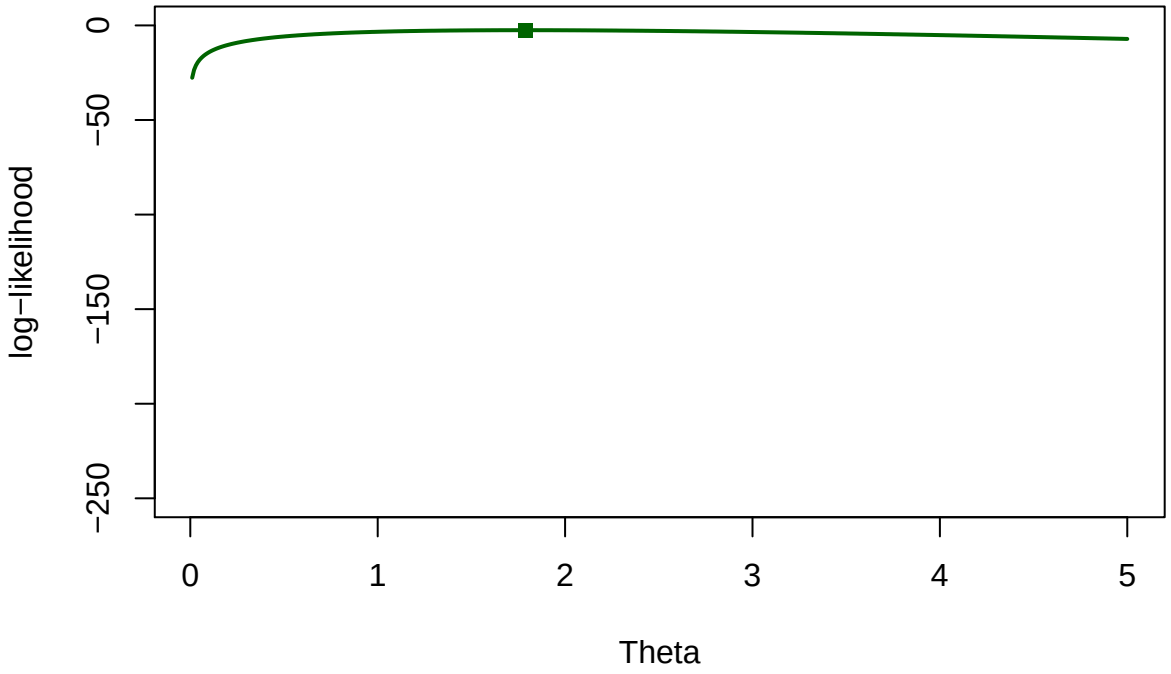
$$p(x|\theta) = \theta^{-\theta x} \tag{3}$$

Note that x is exponentially iid. In addition, below is a plot showing the dependency of log-likelihood on θ where the complete data set was used for fitting.




```
Maximum likelihood value of theta with all obs. is 1.13
```

The same procedure was done, but instead only using the first 6 observations.




```
Maximum likelihood value of theta with 6 obs. is 1.79
```

We can see on the green plot that there are many theta values which has a similar likelihood, which makes sense because we are using a very small data set. However, for the red one we can note that there are many thetas which are a lot worse than the maximum. We can note that maximum likelihood is only as good as the amount of data that we have and the underlying assumed distribution. For example, if we flip a coin 3 times and it ends up Heads all times, then maximum likelihood says that p(H) = 1. So, the θ value received when using all observations is more reiable than only using a subset.

Bayesian model

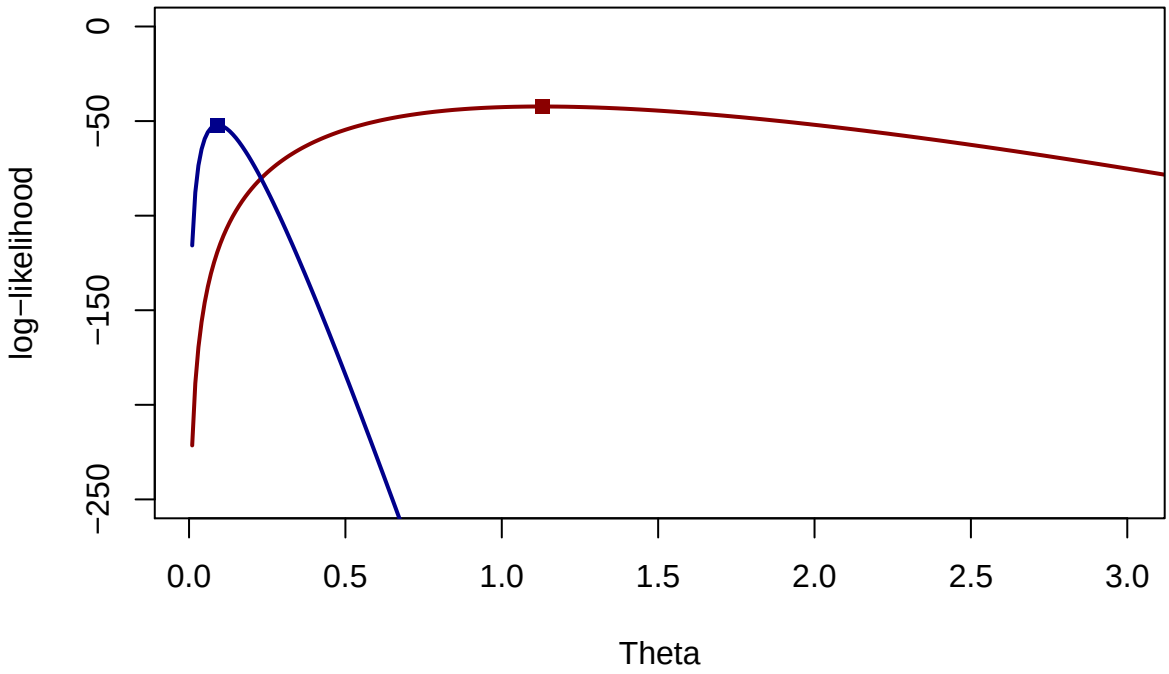
Now we change the model where the likelihood is defined according to Eq. 3 but we introduce a prior, see Eq. 4.

$$p(\theta) = \lambda e^{-\lambda\theta} \tag{4}$$

And we set $\lambda = 10$. From this we have written a function to calculate the function $l(\theta) = log(p(x|\theta)p(\theta))$ which is the log of the posterior, see below.

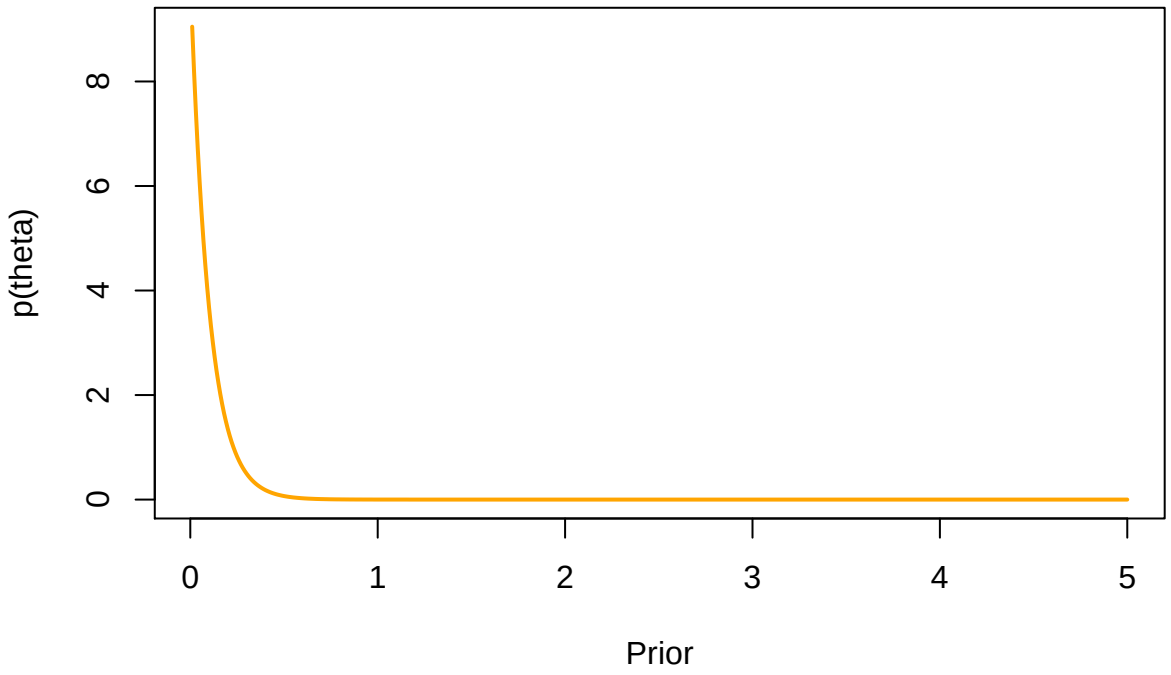
```
prior = function(lambda, theta) {
  val = lambda * exp(-lambda * theta)
  return(val)
}


log_posterior = function(x, theta) {
  val = exp_log_likelihood(x, theta)  + length(x) * log(prior(10, theta))
  return(val)
}
```

The dependency of this function on θ can be seen in the plot below.

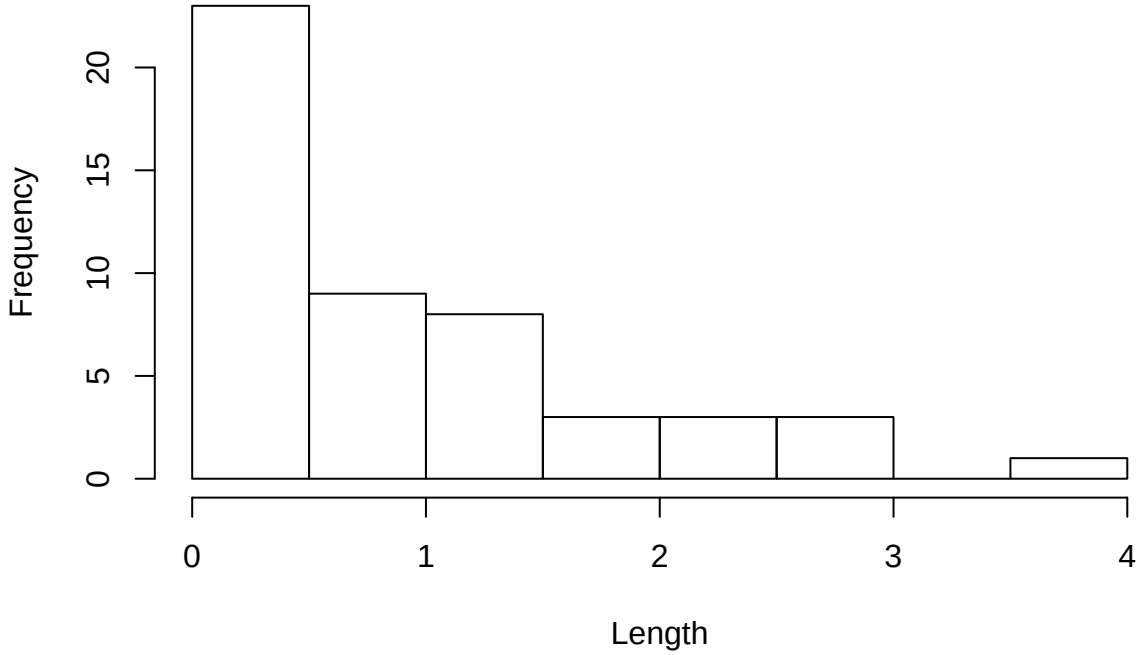



```
Optimal value of theta with all obs. using posterior is 0.09
```

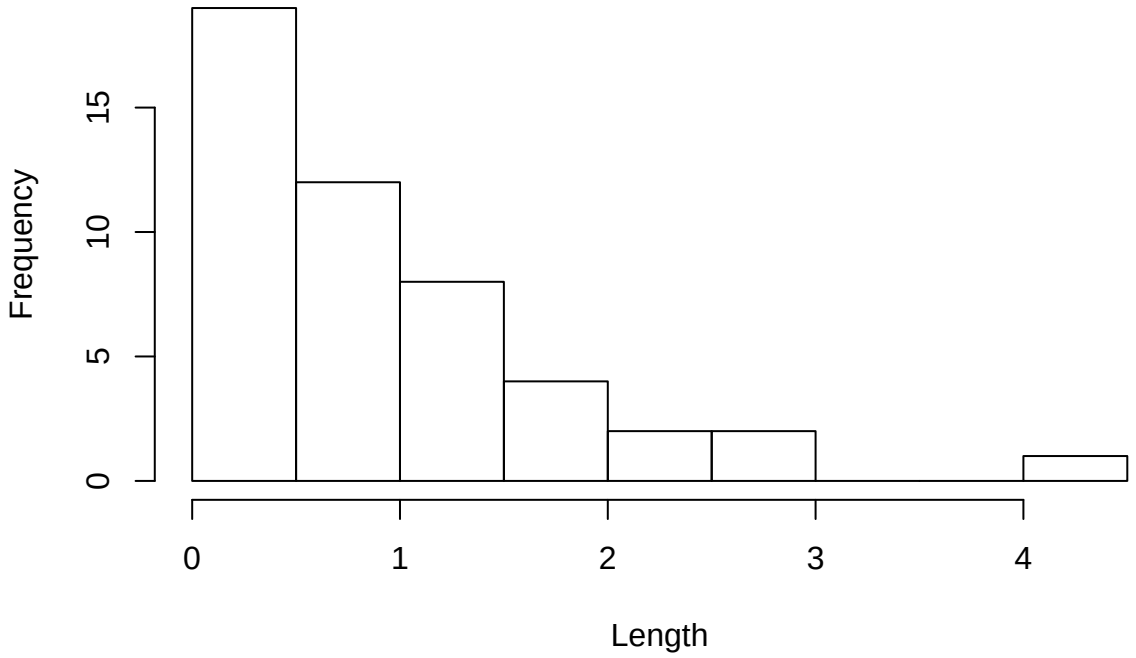
The likelihood was moved to the left (blue curve) due to our previous knowledge about theta. We can plot the prior knowledge about theta and see that it favours low theta values, see below. Hence, we expect θ to decrease, which it did.



We generated 50 datapoints from our first model, see Eq. 3 and plotted them in a histogram, see below. We can use these plots to determine if the found maximum likelihood value of θ is reliable or not.



We can compare the generated data with the original data, see below.

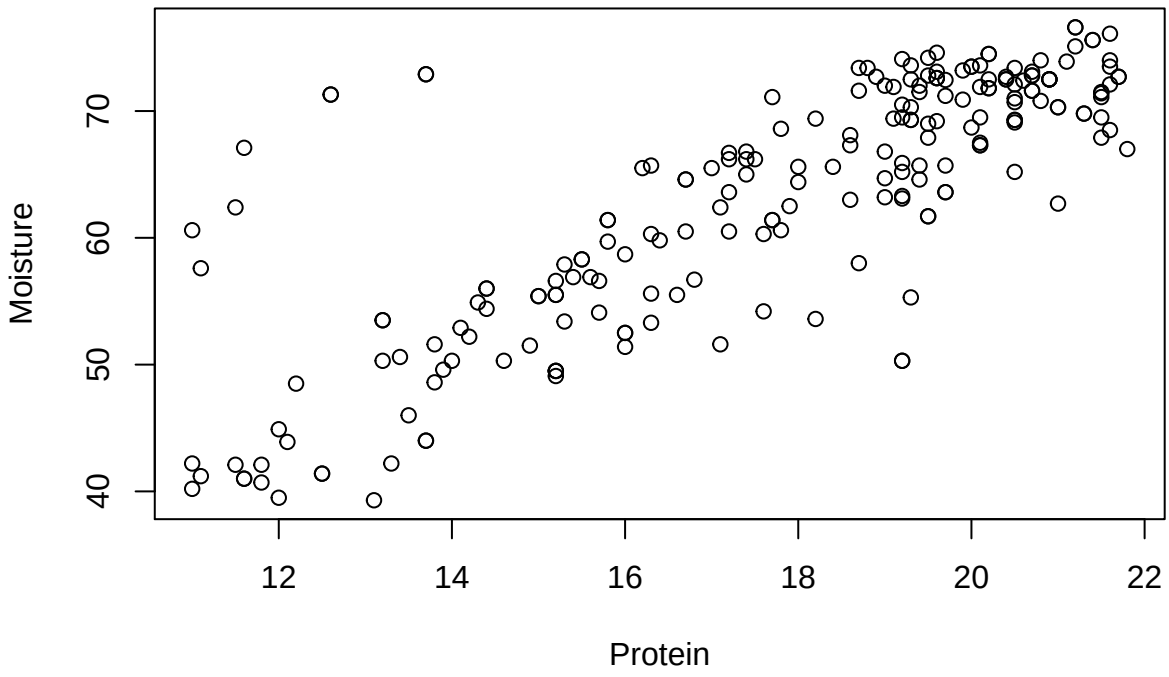


We can note that the two histograms look very similar. Indicating the the found θ is probably a good value representing the underlying distribution.

Assignment 4: Linear regression and regularization

MSE

The plot below depicts Protein versus Moisture.



It can be seen that the data seem to follow a linear pattern. Hence, we can try and model this distribution

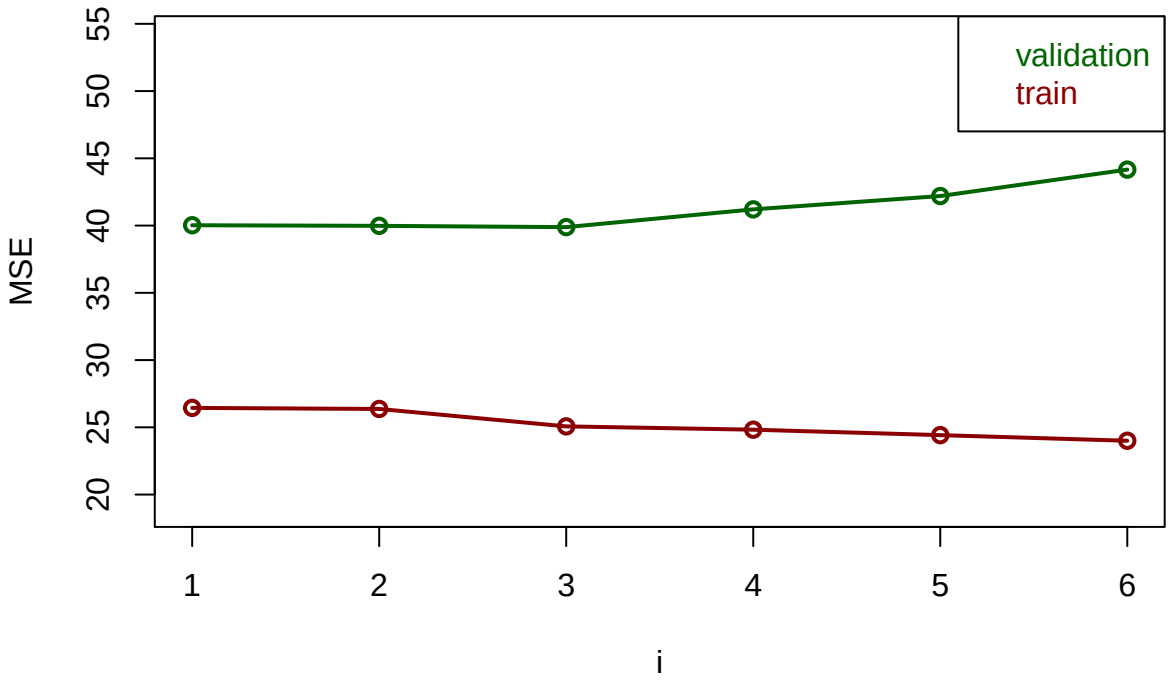
with a linear model.

We assumed the data follows the probability distribution seen in Eq. 5, where Y is Moisture and x is Protein. This model can be adapted by choosing different values of i.

$$Y \sim N(\sum_{n=0}^{i} w_i x^i, \sigma^2) \tag{5}$$

This means that we assume the data follows a Normal distribution. Now, we need a meassure to test this model to find a good value for i that seems to agree with our data. We can use MSE (Mean Squared Error) for this which works out well because if we minimize MSE we are also maximizing the log-likelihood of the model. MSE is also good becaues it exponentially punishes data the further away it is. And our data looks like it only has a few outliers and most are normally distributed a long a line.

Now we divide our data into two sets, training and test data, split 50/50 and fit the model using i = 1...6. We fit the training data on the model and evaluate the MSE against both subsets, see below.



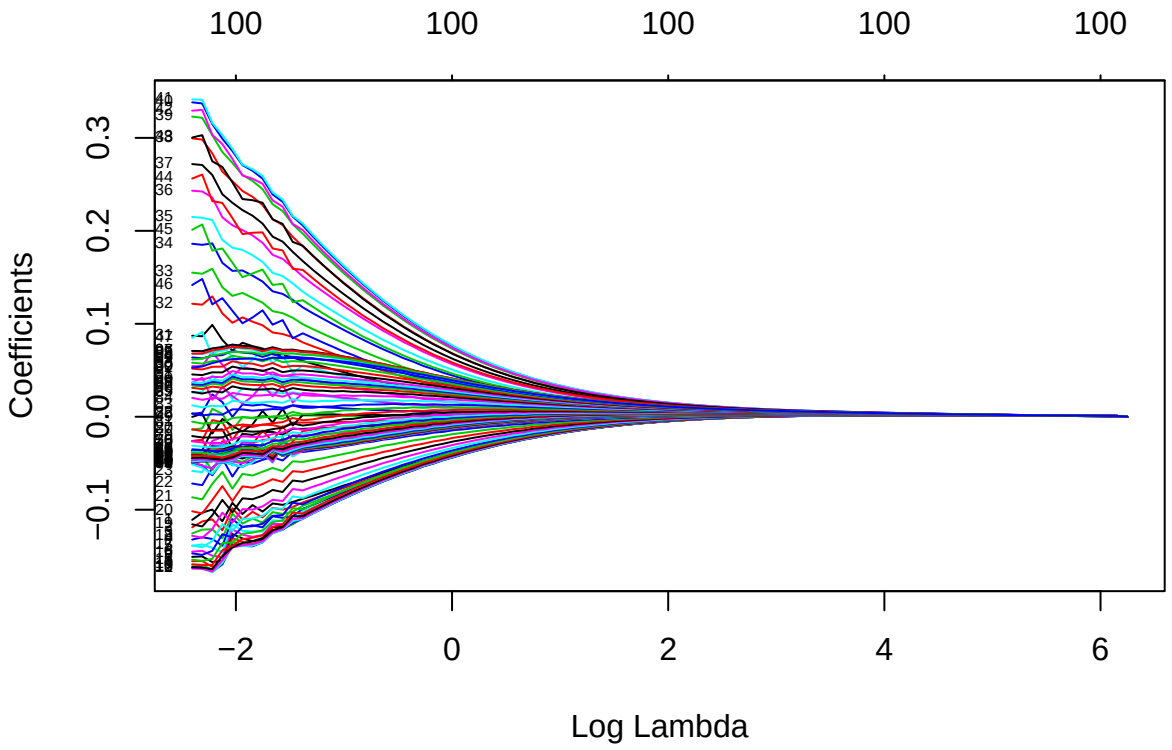
Because a more complex model has less bias but more variance we can note that in the plot, the train MSE goes down while the validation MSE goes up as the model becomes more complex indicating a higher variance. So the best i is the smallest one, the most simpel model in this case.

Variable selection

The next step is about how we can choose the variables to be included in our model. We used the function stepAIC which performs a greedy search where one parameter is added or removed depending on which direction is chosen, then it adds the that makes the AIC (Akaike Information Criteria) smaller.

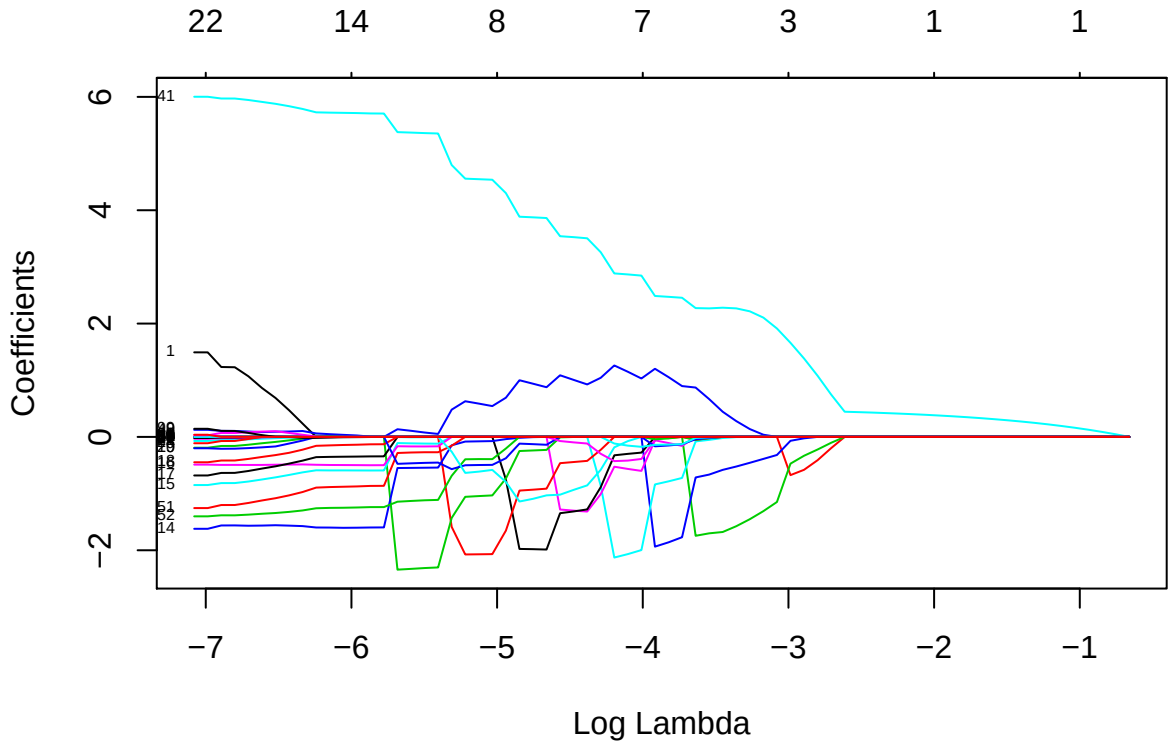
```
Number of parameters chosen by stepAIC is  63
```

Now we perform the task of using Ridge regression on all variables.



We can see that the coefficients goes to 0 as lambda is increased.

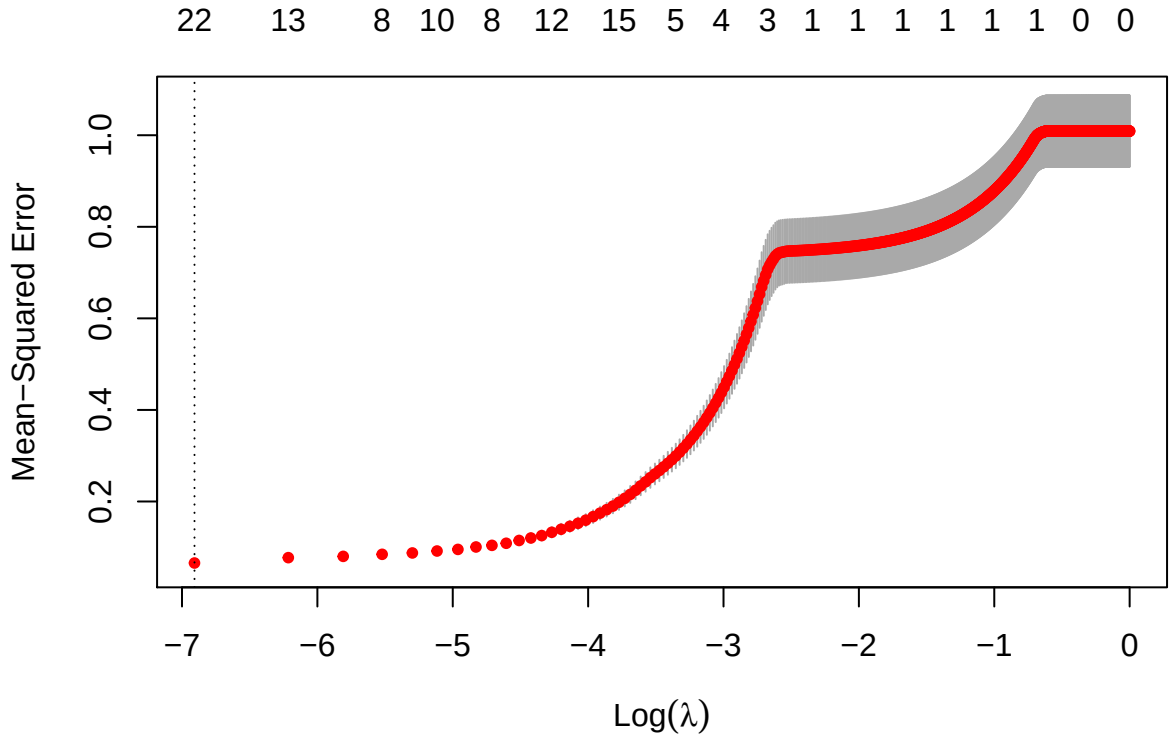
A similar method is called LASSO but where the penalty term is not quadrated.



It chooses fewer features than ridge where some are completely removed. This basically simplifies the model and can sometimes make it easier to understand. This can be beneficial in the case where we have some features and we aren't sure if they really contribute anything or not. However, for ridge, they are never

completely gone, they just converge to 0.

Then we use cross validation to find the optimal λ value for the LASSO model, see below.



According to the plot we can see that the lowest MSE is achieved when $\lambda = 0$. And the score is increasing as we increase λ.

```
Number of coefficients chosen by CV on LASSO model is 100
```

When we use CV on the LASSO model we actually end up using all variables, whereas the stepAIC removes plenty.

Appendix A: All code

```
knitr::opts_chunk$set(echo = TRUE)

if(!require(gdata)) {
  install.packages("gdata")
}

if (!require(kknn)) {
  install.packages("kknn")
}

if (!require(MASS)) {
  install.packages("MASS")
}

if (!require(glmnet)) {
  install.packages("glmnet", dependencies = TRUE)
```

```
}

if (!require(knitr)) {
  install.packages("knitr")
}


missclass = function(x, x_fit) {
  n = length(x)
  t = table(x, x_fit)
  sum = sum(diag(t))
  return(1 - sum/n)
}

if (!exists("spambase")) {
  spambase = read.xls("~/Documents/ml-tdde01/lab1/spambase.xlsx")
}

set.seed(12345)
data_rows = dim(spambase)[1]
train_indexes = sample(1:data_rows, floor(data_rows * 0.5))
train_data = data.frame(spambase[train_indexes, ])
test_data = data.frame(spambase[-train_indexes, ])


logistic = glm(Spam~., data = train_data, family = "binomial")


train_fitted = data.frame(Prob=predict(logistic, newdata = train_data, type = "response"))
test_fitted = data.frame(Prob=predict(logistic, newdata = test_data, type = "response"))

train_spam = train_data["Spam"]
test_spam = test_data["Spam"]

# Train > 50
train_fitted$Spam50 = ifelse(train_fitted$Prob > 0.5, 1, 0)
table_train_50 = table(Gold=train_spam$Spam, Pred=train_fitted$Spam50)
#print(table_train_50)
kable(table_train_50,
      caption = "Training data confusion matrix. X is Predicted and Y is Gold.")


cat(paste(c("Missclassification rate is", missclass(train_spam$Spam, train_fitted$Spam50)),
          collapse = " "))


test_fitted$Spam50 = ifelse(test_fitted$Prob > 0.5, 1, 0)
table_test_50 = table(Gold=test_spam$Spam, Pred=test_fitted$Spam50)
#print(table_test_50)
kable(table_test_50,
      caption = "Test data confusion matrix. X is Predicted and Y is Gold.")
cat(paste(c("Missclassification rate is", missclass(test_spam$Spam, test_fitted$Spam50)),
          collapse = " "))
```

```r
train_fitted$Spam80 = ifelse(train_fitted$Prob > 0.8, 1, 0)
test_fitted$Spam80 = ifelse(test_fitted$Prob > 0.8, 1, 0)

# Train > 80
table_train_80 = table(Gold=train_spam$Spam, Pred=train_fitted$Spam80)
kable(table_train_80,
      caption = "Training data confusion matrix. X is Predicted and Y is Gold.")
cat(paste(c("Missclassification rate is", missclass(train_spam$Spam, train_fitted$Spam80)),
          collapse = " "))


# Test > 80
table_test_80 = table(Gold=test_spam$Spam, Pred=test_fitted$Spam80)
kable(table_test_80,
      caption = "Test data confusion matrix. X is Predicted and Y is Gold.")
cat(paste(c("Missclassification rate is", missclass(test_spam$Spam, test_fitted$Spam80)),
          collapse = " "))


knn_30_train = kknn(Spam~., train = train_data, test = train_data, k = 30)
knn_30_train_fitted = ifelse(knn_30_train$fitted.values > 0.5, 1, 0)
table_knn_30_train_50 = table(Gold=train_data$Spam, Pred=knn_30_train_fitted)
kable(table_knn_30_train_50,
      caption = "Training data confusion matrix. X is Predicted and Y is Gold.")
cat(paste(c("Missclassification rate is", missclass(train_spam$Spam, knn_30_train_fitted)),
          collapse = " "))


knn_30_test = kknn(Spam~., train = train_data, test = test_data, k = 30)
knn_30_test_fitted = ifelse(knn_30_test$fitted.values > 0.5, 1, 0)
table_knn_30_test_50 = table(Gold=test_data$Spam, Pred=knn_30_test_fitted)
kable(table_knn_30_test_50,
      caption = "Test data confusion matrix. X is Predicted and Y is Gold.")
cat(paste(c("Missclassification rate is", missclass(test_spam$Spam, knn_30_test_fitted)),
          collapse = " "))


knn_1_train = kknn(Spam~., train = train_data, test = train_data, k = 1)
knn_1_train_fitted = ifelse(knn_1_train$fitted.values > 0.5, 1, 0)
table_knn_1_train_50 = table(Gold=train_data$Spam, Pred=knn_1_train_fitted)
kable(table_knn_1_train_50,
      caption = "Training data confusion matrix. X is Predicted and Y is Gold.")
cat(paste(c("Missclassification rate is", missclass(train_spam$Spam, knn_1_train_fitted)),
          collapse = " "))


knn_1_test = kknn(Spam~., train = train_data, test = test_data, k = 1)
knn_1_test_fitted = ifelse(knn_1_test$fitted.values > 0.5, 1, 0)
table_knn_1_test_50 = table(Gold=test_data$Spam, Pred=knn_1_test_fitted)
kable(table_knn_1_test_50,
      caption = "Test data confusion matrix. X is Predicted and Y is Gold.")
cat(paste(c("Missclassification rate is", missclass(test_spam$Spam, knn_1_test_fitted)),
```

```
          collapse = " "))

if (!exists("machines")) {
  machines = read.xls("~/Documents/ml-tdde01/lab1/machines.xlsx")
}

# We assume the probability model p(x|0) = 0*e^(-0*x).
# This is an exponential distribution.

likelihood = function(x, theta) {
  v = theta * exp(- theta * x)
  return(v)
}

exp_log_likelihood = function(x, theta) {
  helper = function(xi) {
    return(log(likelihood(xi, theta)))
  }
  log_likelihood = sum(sapply(x, helper))
  return(log_likelihood)
}

thetas = seq(0.01, 5, by = 0.01)
#
# Using all observations.
#
log_likelihoods_all = sapply(thetas,
                             function(theta) exp_log_likelihood(machines$Length, theta))
max_theta_index_all = match(max(log_likelihoods_all), log_likelihoods_all)
max_theta_all = thetas[max_theta_index_all]

# Plots the thetas against the machines$Length
plot(thetas, log_likelihoods_all, type = "l", col = "darkred", lwd = 2,
     xlim = c(min(thetas), max(thetas)), ylim = c(-250, 0),
     xlab = "Theta", ylab = "log-likelihood")
points(max_theta_all, max(log_likelihoods_all), pch = 15, col = "darkred")

# Prints maximum likelihood value of theta for all obs.
cat(paste(c("Maximum likelihood value of theta with all obs. is", max_theta_all),
          collapse = " "))
#
# Using the first 6 observations
#
log_likelihoods_6 = sapply(thetas,
                           function(theta) exp_log_likelihood(machines$Length[1:6], theta))
max_theta_index_6 = match(max(log_likelihoods_6), log_likelihoods_6)
max_theta_6 = thetas[max_theta_index_6]

# Plots the thetas against the machines$Length.
plot(thetas, log_likelihoods_6, type = "l", col = "darkgreen", lwd = 2,
      xlim = c(min(thetas), max(thetas)), ylim = c(-250, 0),
      xlab = "Theta", ylab = "log-likelihood")
points(max_theta_6, max(log_likelihoods_6), pch = 15, col = "darkgreen")
```

```r
# Prints maximum likelihood value of theta for 6 obs.
cat(paste(c("Maximum likelihood value of theta with 6 obs. is", max_theta_6),
          collapse = " "))

prior = function(lambda, theta) {
  val = lambda * exp(-lambda * theta)
  return(val)
}


log_posterior = function(x, theta) {
  val = exp_log_likelihood(x, theta)  + length(x) * log(prior(10, theta))
  return(val)
}


log_posterior_all = sapply(thetas, function(theta) log_posterior(machines$Length, theta))
max_theta_index_post_all = match(max(log_posterior_all), log_posterior_all)
max_theta_post_all = thetas[max_theta_index_post_all]

# Plots the thetas against the machines$Length
plot(thetas, log_likelihoods_all,
     type = "l",
     col = "darkred",
     lwd = 2,
     xlim = c(min(thetas), 3),
     ylim = c(-250, 0),
     xlab = "Theta",
     ylab = "log-likelihood")
points(max_theta_all, max(log_likelihoods_all), pch = 15, col = "darkred")
points(thetas, log_posterior_all, type = "l", col = "darkblue", lwd = 2)
points(max_theta_post_all, max(log_posterior_all), pch = 15, col = "darkblue")

# Prints the maximum theta value from the log posterior.
cat(paste(c("Optimal value of theta with all obs. using posterior is", max_theta_post_all),
          collapse = " "))

plot(thetas, sapply(thetas, function(theta) prior(10, theta)),
     type = "l", col = "orange", lwd = 2, xlab = "Prior", ylab = "p(theta)")

#
# Generate 50 new observations.
#
#min_v = min(sapply(machines$Length, function(l) likelihood(l, max_theta_all)))
#max_v = max(sapply(machines$Length, function(l) likelihood(l, max_theta_all)))

# We can draw random uniform values in the range(min_v, max_v)
# and plug get x = (ln(theta) - ln(r)) / theta
# which is simply the solution to r = theta * e ^ (-theta * x).

#r_nums = runif(50, min = min_v, max = max_v)
#new_obs = sapply(r_nums, function(r) (log(max_theta_all) - log(r)) / max_theta_all)
```

```r
new_obs = rexp(50, max_theta_all)
# Plots the generated observations.
hist(new_obs, main = "", xlab = "Length")


# Plots the original observations.
hist(machines$Length, main = "", xlab = "Length")


if (!exists("tecator")) {
  tecator = read.xls("~/Documents/ml-tdde01/lab1/tecator.xlsx")
}

#
# Part 1
#
#summary(tecator)
plot(tecator$Protein, tecator$Moisture, type = "p", xlab="Protein", ylab="Moisture")


set.seed(12345)
rows = dim(tecator)[1]
train_indexes = sample(1:rows, floor(rows * 0.5))
train = tecator[train_indexes, ]
validation = tecator[-train_indexes, ]

get_mse_poly = function(deg) {
  model = lm(Moisture ~ poly(Protein, deg), data = train)
  train_fit = predict(model)
  validation_fit = predict(model, newdata = validation)

  mse = list()
  mse["Train"] = mean((train_fit - train$Moisture)^2)
  mse["Validation"] = mean((validation_fit - validation$Moisture)^2)
  return(mse)
}

degrees = seq(1, 6)

mses = data.frame(t(data.frame(sapply(degrees, function(deg) get_mse_poly(deg)))))
# Gets min/max MSE for plot limits
min_mse = min(unlist(mses$Validation), unlist(mses$Train))
max_mse = max(unlist(mses$Validation), unlist(mses$Train))
plot(degrees, mses$Train,
     type = "o",
     col = "darkred",
     ylim = c(min_mse - 5, max_mse + 10),
     ylab = "MSE",
     xlab = "i",
     lwd = 2)
points(degrees, mses$Validation, type = "o", col = "darkgreen", lwd = 2)

legend("topright", c("validation", "train"), text.col=c("darkgreen", "darkred"))
```

```r
#
# Part 4
#
exclude = function(columns, df) {
  return(df[, !(colnames(df) %in% columns)])
}

channel_fat = exclude(c("Sample", "Protein", "Moisture"), tecator)
fat_model = glm(Fat~., data = channel_fat)

if (!exists("fat_model_aic")) {
  fat_model_aic = stepAIC(fat_model, trace = FALSE)
}

#summary(fat_model_aic$model)
num_parameters = length(exclude("Fat", fat_model_aic$model))
cat(paste(c("Number of parameters chosen by stepAIC is ", num_parameters),
          collapse = " "))

#
# Part 5
#
# Alpha = 0, ridge
# Alpha = 1, lasso
channels = scale(exclude("Fat", channel_fat))
fats = scale(data.frame(Fat = channel_fat$Fat))
ridge_model = glmnet(as.matrix(channels), fats, alpha = 0, family = "gaussian")
plot(ridge_model, xvar = "lambda", label = TRUE)


#
# Part 6
#

lasso_model = glmnet(as.matrix(channels), fats, alpha = 1, family = "gaussian")

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


#
# Part 7
#

if (!exists("lasso_cv_model")) {
  lasso_cv_model = cv.glmnet(as.matrix(channels), fats,
                             alpha = 1,
                             family = "gaussian",
                             lambda = seq(0, 1, 0.001))
}
```

```r
plot(lasso_cv_model)


num_variables = length(coef(lasso_cv_model, s = "lambda.min")[,1]) - 1
cat(paste(c("Number of coefficients chosen by CV on LASSO model is", num_variables),
          collapse = " "))
```