Lab 1 Assignment 1

Gustaf Söderholm, gusso811

November 22, 2019

1 Spam classification with nearest neighbors

The data file spambase.xlsx contains information about the frequency of various words, characters etc for a total of 2740 e-mails. Furthermore, these e-mails have been manually classified as spams (spam = 1) or regular e-mails (spam = 0).

Librarys used:

```
library(kknn)
library(readxl)
```

1.1 Import the data into R and divide it into training and test sets (50%/50%)

Also changed *Spam* from numeric to factor to make it easier to follow.

```
spambase <- read_excel("liu/TDDE01/spambase.xlsx")
spambase[49] <- replace(spambase[49], spambase[49]==0, "FALSE")
spambase[49] <- replace(spambase[49], spambase[49]==1, "TRUE")
spambase$Spam <- as.factor(spambase$Spam)
n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=spambase[id,]
test=spambase[-id,]
cat(paste0("Number of test samples: \t",nrow(test),"\n"))
cat(paste0("Number of train samples: \t",nrow(train),"\n"))
```

```
Number of test samples:  1370
Number of train samples:  1370
```

1.2 Use logistic regression to classify the training and test data

Using classification principle:

$$\hat{Y} = 1 \quad \text{if} \quad p(Y = 1|x) > 0.5, \quad \text{else} \quad \hat{Y} = 0 \tag{1}$$

Using *glm()* to create a model from the training data:

```r
log_reg <- glm(Spam ~ ., data=train, family="binomial")
summary(log_reg)
```

```
Call:
glm(formula = Spam ~ ., family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5205  -0.4402  -0.0005   0.6584   3.6196

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.508e+00  2.011e-01   7.499 6.44e-14 ***
Word1       -7.192e-01  5.015e-01  -1.434 0.151520
...
Word48      -4.390e+00  1.473e+00  -2.980 0.002878 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1696.82  on 1369  degrees of freedom
Residual deviance:  928.54  on 1321  degrees of freedom
AIC: 1026.5

Number of Fisher Scoring iterations: 23
```

Report the confusion matrices and the misclassification rates for training and test data

Using *predict()* to make a prediciton on the training and test data.

```r
predicted_train <- predict(log_reg, newdata=train, type="response
    ")
predicted_test <- predict(log_reg, newdata=test, type="response")
```

Creating a confusion matrix and calculating misclassification rate.

```r
conf_matrix_train <- table(train$Spam, predicted_train>0.5)
miss_rate_train <- (conf_matrix_train[2] + conf_matrix_train[3])/
    dim(train)[1]
print(conf_matrix_train)

conf_matrix_test <- table(test$Spam, predicted_test>0.5)
miss_rate_test <- (conf_matrix_test[2] + conf_matrix_test[3])/dim
    (test)[1]
print( conf_matrix_test)
```

Training data:

```
       FALSE TRUE
  FALSE   803  142
  TRUE     81  344
```

```
Misclassification rate: 0.162773722627737
```

Test data:

```
       FALSE TRUE
  FALSE   791  146
  TRUE     97  336
```

```
  Misclassification rate: 0.177372262773723
```

Analyse the obtained results

As we can see the misclassification rates are around 16-18%. It is not great but it is low. As the rates are close to eachother we can determan that the model is not overfitted and works similar for both training and test data.

1.3 Use logistic regression to classify the training and test data

Using classification principle:

$$\hat{Y} = 1 \quad \text{if} \quad p(Y = 1|x) > 0.8, \quad \text{else} \quad \hat{Y} = 0 \tag{2}$$

Same model and prediction as in 1.2

Creating new confusion tables for our new classification principle.

```
conf_matrix_train <- table(train$Spam, predicted_train>0.8)
miss_rate_train <- (conf_matrix_train[2] + conf_matrix_train[3])/
   dim(train)[1]
print(conf_matrix_train)

conf_matrix_test <- table(test$Spam, predicted_test>0.8)
miss_rate_test <- (conf_matrix_test[2] + conf_matrix_test[3])/dim
   (test)[1]
print( conf_matrix_test)
```

Training data:

```
       FALSE TRUE
  FALSE   941    4
  TRUE    335   90
```

```
  Misclassification rate: 0.247445255474453
```

Test data:

```
       FALSE TRUE
  FALSE   926   11
  TRUE    367   66

  Misclassification rate: 0.275912408759124
```

Compare the results with step 1.2. What effect did the new rule have? Compared to the results in 1.2 the rates have gone up, but the difference between training and test data is similar.

1.4 Use standard classifier kknn() with K=30

Using *kknn()* to create a model that classifies depending on the 30 closest neighbors.

```
kknn_30 <-kknn(Spam ~ ., train=train, test=test, k=30)
```

Report the the misclassification rate for the training and test data
Using *fitted()* to extract the fitted values in the kknn sets of data.

```
fit_kknn_30 <- fitted(kknn_30)
```

Creating a confusion matrix and calculating misclassification rate.

```
conf_matrix_kknn_train <- table(train$Spam, fit_kknn_30)
miss_rate_kknn_train <- (conf_matrix_kknn_train[2] + conf_matrix_
   kknn_train[3])/dim(train)[1]
print(conf_matrix_kknn_train)
cat(paste0("Misclassification rate: ",miss_rate_kknn_train,"\n"))


conf_matrix_kknn_test <- table(test$Spam, fit_kknn_30)
miss_rate_kknn_test <- (conf_matrix_kknn_test[2] + conf_matrix_
   kknn_test[3])/dim(test)[1]
print( conf_matrix_kknn_test)
cat(paste0("Misclassification rate: ",miss_rate_kknn_test,"\n"))
```

Traning data:

```
       FALSE TRUE
  FALSE   807  138
  TRUE     98  327

Misclassification rate: 0.172262773722628
```

Test data:

```
       FALSE TRUE
  FALSE   672  265
  TRUE    187  246

Misclassification rate: 0.32992700729927
```

Compare the results with step 1.2

Compared with the model in 1.2 this model is way worse when it comes to test data. It may be the high k-value that causes this.

1.5 Use standard classifier kknn() with K=1

Calling *kknn()* to create a model that classifies depending on the closest neighbor.

```
kknn_1 <-kknn(Spam ~ ., train=train, test=test, k=1)
```

Report the the misclassification rate for the training and test data

Using *fitted()* to extract the fitted values in the kknn sets of data.

```
fit_kknn_1 <- fitted(kknn_1)
```

Creating a confusion matrix and calculating misclassification rate.

```
conf_matrix_kknn_train <- table(train$Spam, fit_kknn_1)
miss_rate_kknn_train <- (conf_matrix_kknn_train[2] + conf_matrix_
   kknn_train[3])/dim(train)[1]
print(conf_matrix_kknn_train)
cat(paste0("Misclassification rate: ",miss_rate_kknn_train,"\n"))


conf_matrix_kknn_test <- table(test$Spam, fit_kknn_1)
miss_rate_kknn_test <- (conf_matrix_kknn_test[2] + conf_matrix_
   kknn_test[3])/dim(test)[1]
print( conf_matrix_kknn_test)
cat(paste0("Misclassification rate: ",miss_rate_kknn_test,"\n"))
```

Training data:

```
       FALSE TRUE
  FALSE   945    0
  TRUE      0  425

Misclassification rate: 0
```

Test data:

```
       FALSE TRUE
  FALSE   640  297
  TRUE    177  256

Misclassification rate: 0.345985401459854
```

Compare the results with step 1.4. What effect does the decrease of K lead to and why?

As we see here, the high k-value was not what caused the high rates in 1.4, as the test data has similar results. The training data has a misclassification rate of zero do to the we only look at one neighbor. That neighbor would be the same as the sample we are trying to classify.

It seems like k-nearest neighbor is not a good fit for this type of case. Futher investigations on a proper k-value can be made. A rule of thumb is to take the square root of the total number of observations, in this case closer to 52.

Lab 1 Assignment 2

Antymos Dag (antda166)

11/23/2019

The data file machines.xlsx contains information about the lifetime of certain machines, and the company is interested to know more about the underlying process in order to determine the warranty time. The variable is Length: shows lifetime of a machine.

1. Import the data to R.

```
data = read.csv("machines.csv", header = TRUE)
```

2. What is the distribution type of x?

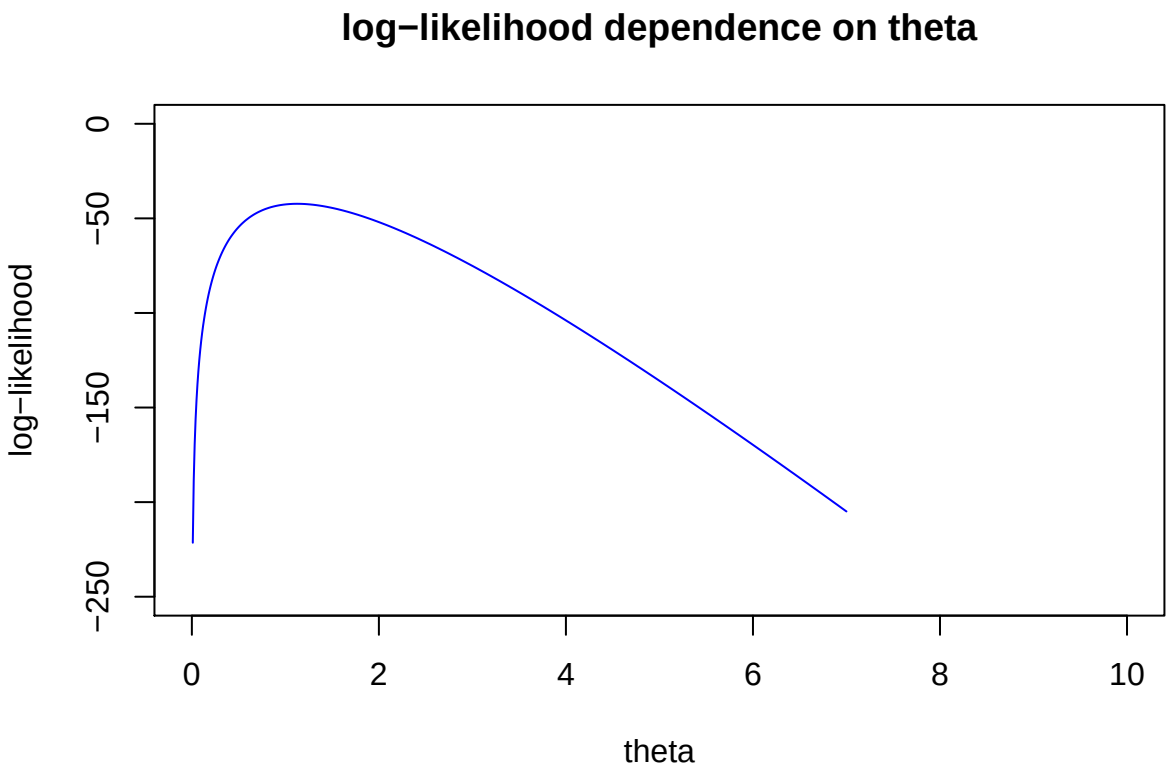
The given probability model $p(x|\theta) = \theta e^{-\theta x}$ is equivalent to the PDF of the exponential distribution. The distribution type is exponential.

Write a function that computes the log-likelihood $log\ p(x|\theta)$ for a given θ and a given data vector.

```
loglikelihood_exponential = function(data_vec, theta)
{
  likelihood_vec = 1:length(data_vec)

  for (i in 1:length(theta)) {
    likelihood_vec[i] = log(prod(theta[i] * exp(-theta[i] * data_vec)))
  }

  return (likelihood_vec)
}
```

Plot the curve showing the dependence of log-likelihood on θ where the entire data is used for fitting.

```
theta_vec = seq(from = 0, to = 7, by = 0.01)
loglik = loglikelihood_exponential(data$Length, theta_vec)
plot(theta_vec, loglik, type = 'l', col="blue", main = "log-likelihood dependence on theta"
     , xlim = c(0,10), ylim = c(-250, 0), xlab = "theta", ylab = "log-likelihood")
```



What is the maximum likelihood value of θ according to the plot?

The maximum likelihood value of θ is -42.2948 which is given by $\theta = 1.13$.

```
max_theta = theta_vec[as.numeric(which.max(loglik))]
print(max_theta)
```

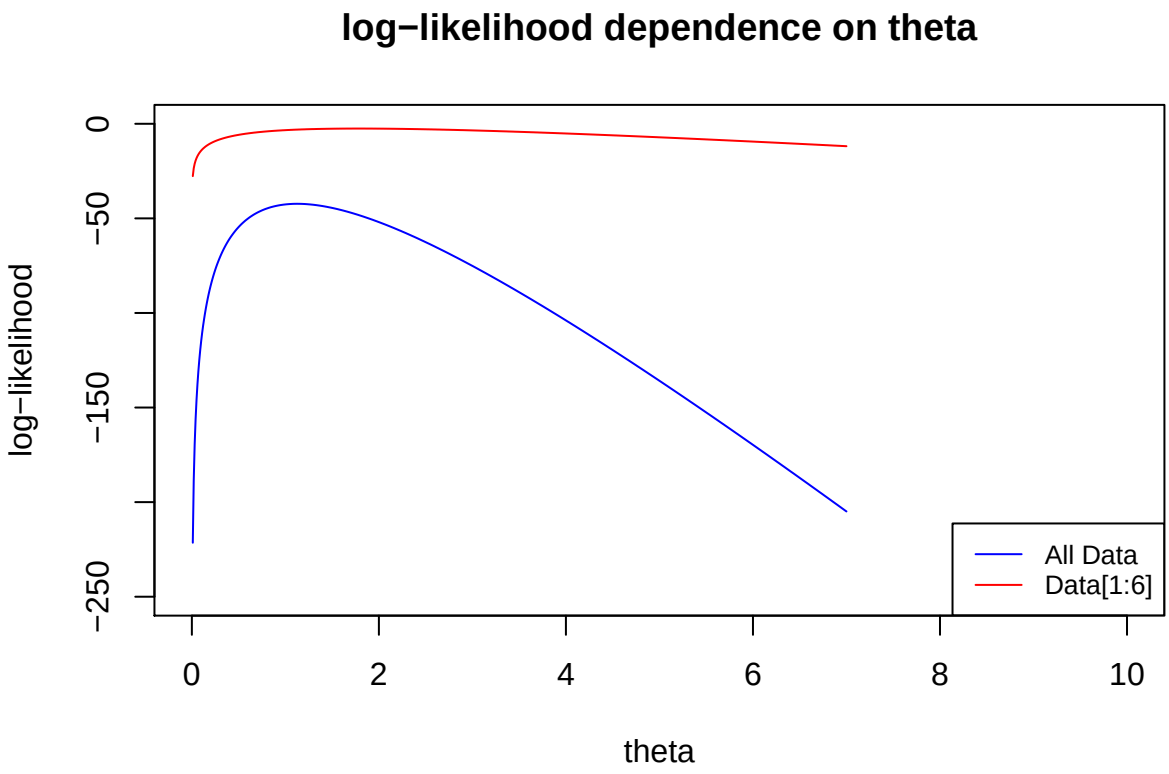
```
## [1] 1.13
```

```
print(max(loglik))
```

```
## [1] -42.2948
```

3. Repeat step 2 but use only 6 first observations from the data, and put the two log-likelihood curves (from step 2 and 3) in the same plot.

```
loglik_subset = loglikelihood_exponential(data$Length[1:6], theta_vec)
plot(theta_vec, loglik, type = 'l', col="blue", main = "log-likelihood dependence on theta"
     , xlim = c(0,10), ylim = c(-250, 0), xlab = "theta", ylab = "log-likelihood")
points(theta_vec, loglik_subset, type='l', col = "red")
legend("bottomright", legend=c("All Data", "Data[1:6]"), col=c("blue", "red"), lty=1:1, cex=0.8)
```



What can you say about reliability of the maximum likelihood solution in each case?

As the size of of the validation data increases the reliability also increases. With the data subset there is a greater chance of noise negatively affecting the model in this regard. This could be the reason that the log-likelihood for the complete dataset decreases exponentially while the corresponding one for the subset of data does not.

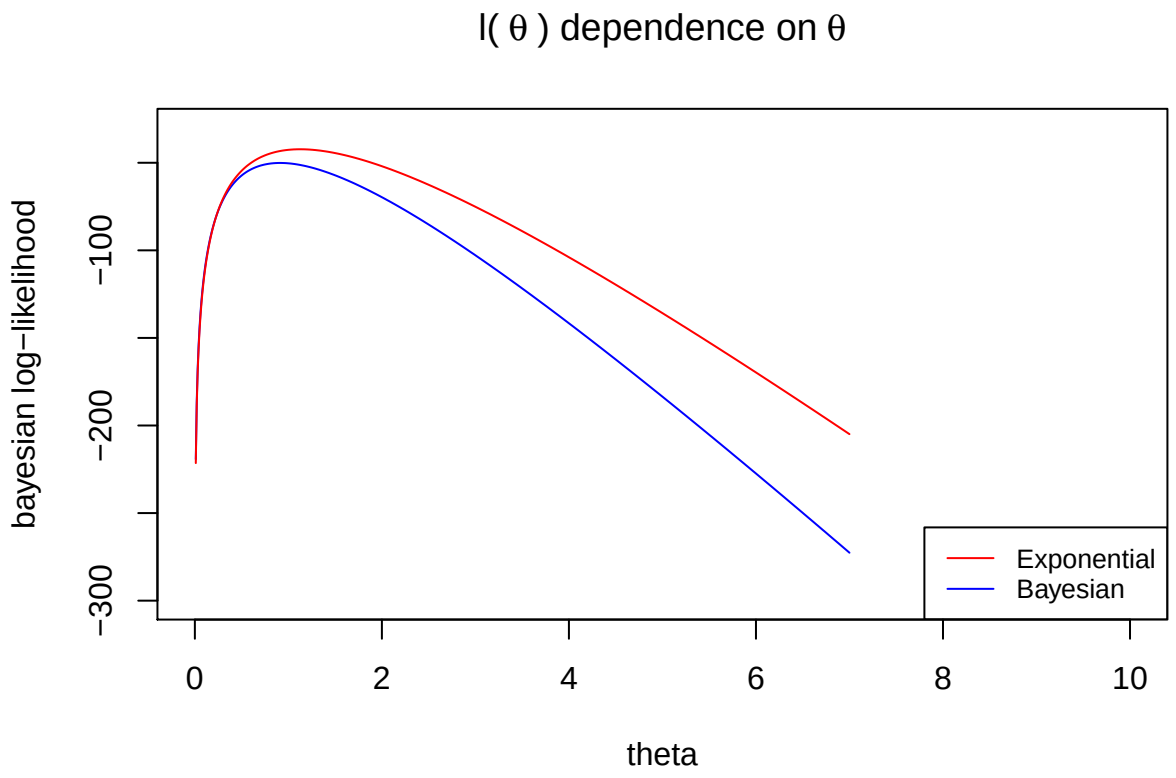
4. Write a function computing $l(\theta) = log(p(x|\theta)p(\theta))$. What kind of measure is actually computed by this function?

```
loglikelihood_bayesian <- function(data_vec, theta, lambda)
{
  likelihood_vec = numeric(length(theta))

  for (i in 1:length(theta)) {
    posterior_prob = prod(theta[i] * exp(-theta[i] * data_vec))
    prior = lambda * exp(-theta[i] * lambda)
    likelihood_vec[i] = log(posterior_prob*prior)
  }

  return(likelihood_vec)
}
```

The function above computes the bayesian log-likelihood using prior thetas.

Plot the curve showing the dependence of $l(\theta)$ on θ computed using the entire data and overlay it with a plot from step 2. Find an optimal θ and compare your result with the previous findings.

```
bayes_loglik = loglikelihood_bayesian(data$Length, theta_vec, 10)
plot(theta_vec, bayes_loglik, type = 'l', col="blue", main = "l("~theta~") dependence on" ~ theta
     , xlim = c(0,10), ylim = c(-300, -30), xlab = "theta", ylab = "bayesian log-likelihood")
points(theta_vec, loglik, type = 'l', col="red")
legend("bottomright", legend=c("Exponential", "Bayesian"), col=c("red", "blue"), lty=1:1, cex=0.8)
```




```
max_theta = theta_vec[as.numeric(which.max(bayes_loglik))]
print(max_theta)
```

```
## [1] 0.91
```

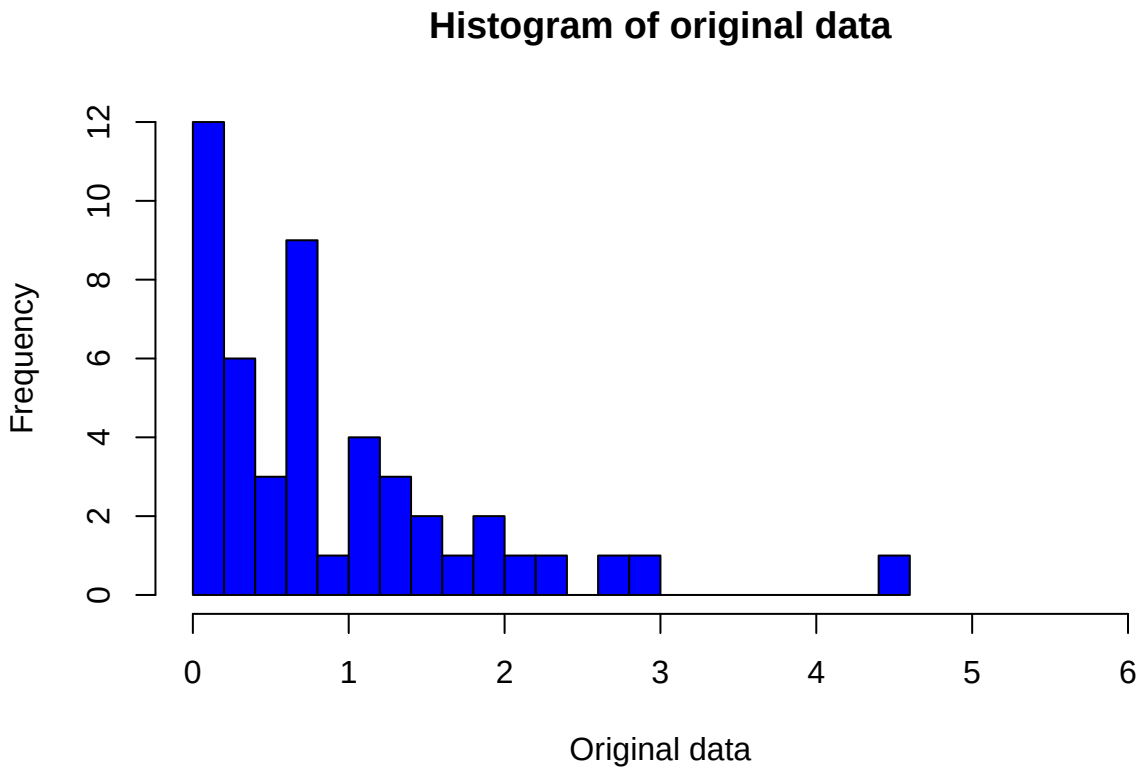
```
print(max(bayes_loglik))
```

```
## [1] -50.10905
```

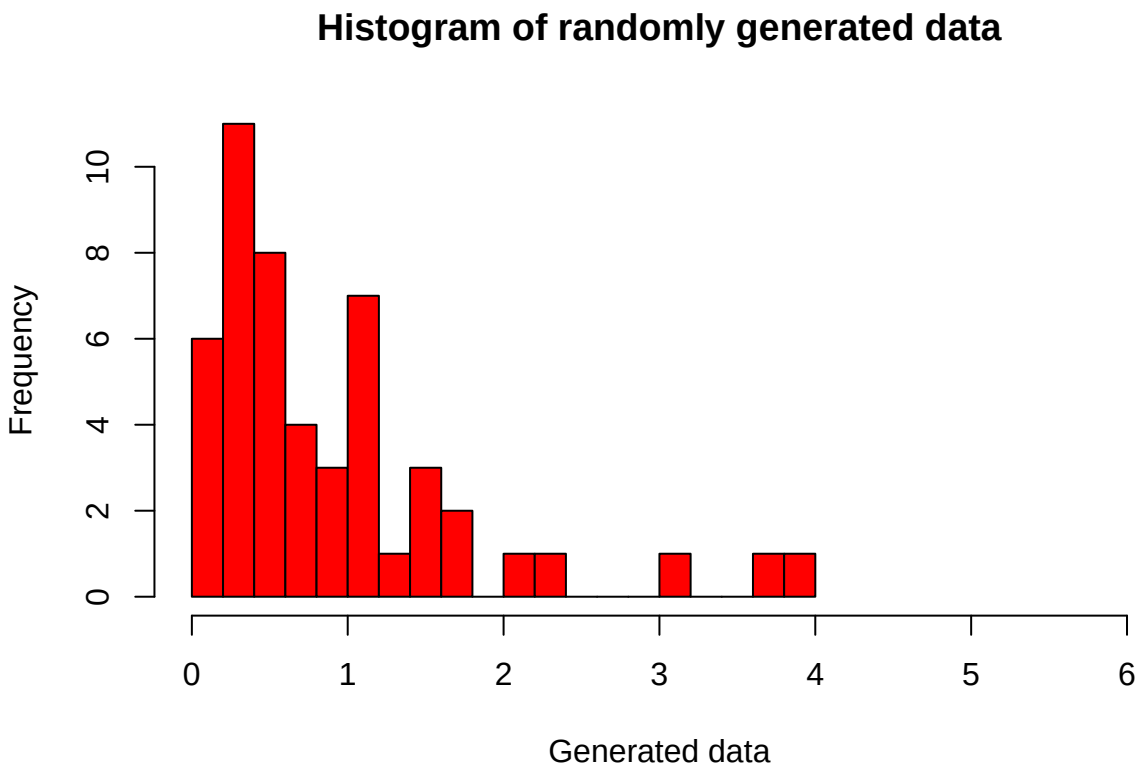
The optimal θ in this case is 0.91 and it gives us a log-likelihood of -50.10905. Since the priors are exponentially distributed and the mean is low, the log-likelihood now peaks and then decreases quicker compared to previous steps. This explains why part of the curve seems to have been displaced.

5. Generate 50 new observations. Create the histograms of the original and the new data and make conclusions.

```
new_data = rexp(50, rate=max_theta)
hist(as.matrix(data), col = "blue", main="Histogram of original data", xlab = "Original data",
     xlim = c(0,6), breaks = 20)
```




```
hist(new_data, col = "red", main="Histogram of randomly generated data", xlab = "Generated data",
     xlim = c(0,6), breaks = 20)
```



The data that was randomly generated from our model has a similar distribution to the original data. This indicates that our estimation of θ is reasonably close to its actual value.

Lab 1 Assignment 4

marpa961

Linear regression and regularization

The Excel file *tecator.xlsx* contains the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry.

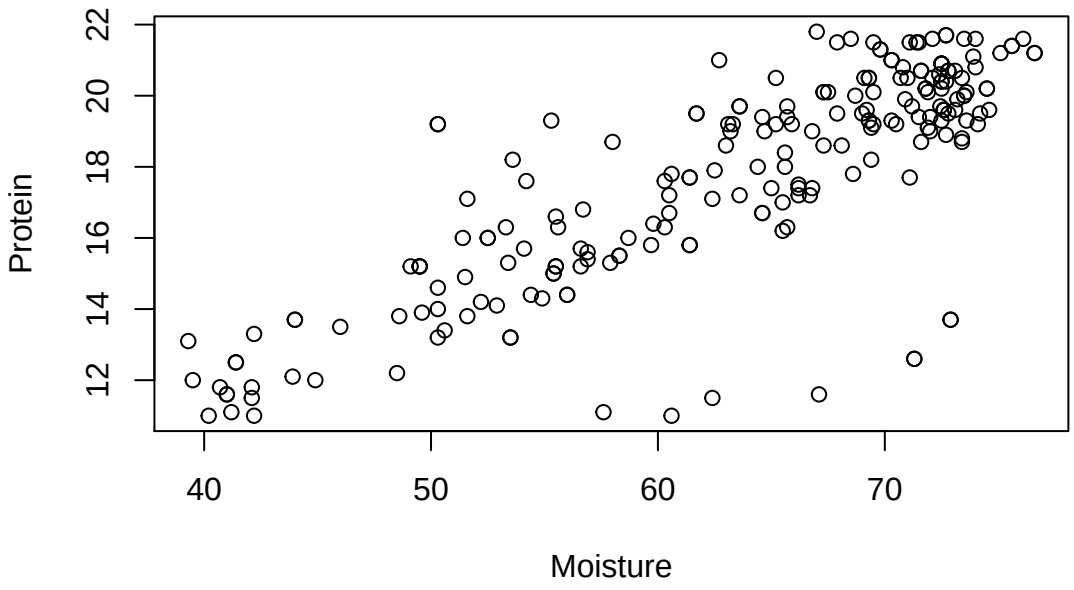
1. Import data to R.

```
library(xlsx)
tecator <- read.xlsx("tecator.xlsx", sheetName = "data")
cat(paste0("Number of samples: \t",nrow(tecator),"\n"))
```

```
## Number of samples:   215
```

Create a plot of Moisture versus Protein.

```
plot(tecator$Moisture,tecator$Protein,xlab="Moisture",ylab="Protein")
```



Do you think that these data are described well by a linear model?

Yes, because it can be easily seen in the graph that there is a linear relation between both variables.

2. Consider model M_i in which Moisture is normally distributed, and the expected Moisture is a polynomial function of Protein including the polynomial terms up to power *i* (i.e M_1 is a linear model, M_2 is a quadratic model...). Report a probabilistic model that describes M_i.

The model can be described by a normal ditribution which mean is defined with a polynomial function.

$$Y \sim N(w_0 + w_1 x + w_2 x^2 + ... + w_n x^n, \sigma^2)$$

, where x represents the feature "Protein" and Y the expected Moisture level.

Why is it appropriate to use MSE criterion when fitting this model to a training data?

Because MSE computes the distance between the predicted values and the real ones, which is a good measure to compare models in a very intuitive way.

3. Divide the data into training and validation sets(50%/50%).

```
n=dim(tecator)[1]
id=sample(1:n, floor(n*0.5))
train=tecator[id,]
test=tecator[-id,]
cat(paste0("Number of training samples: \t",nrow(train),
         "\nNumber of test samples: \t",nrow(test),"\n"))
```

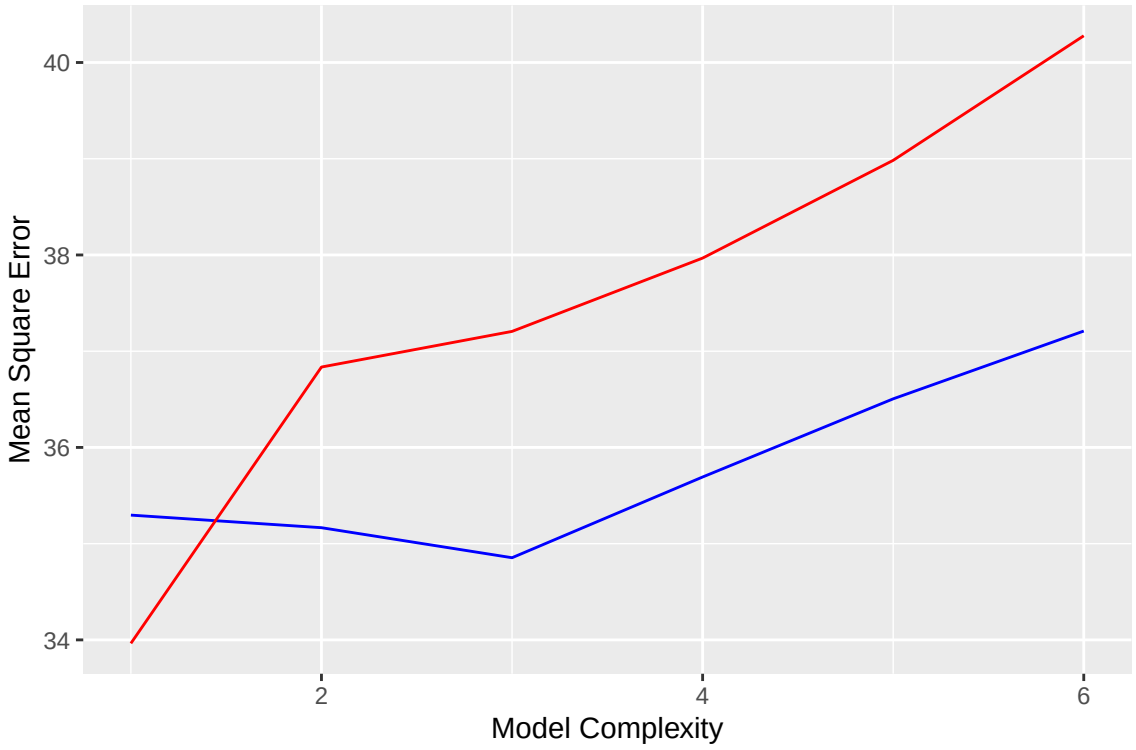
```
## Number of training samples:  107
## Number of test samples:  108
```

Fit models $M_i, i = 1...6$. For each model, record the training and the validation MSE.

```
my_mse <- function(predicted,actual){
    mean((predicted-actual)^2)
}

mse_df <- data.frame(model=1:6, train_error=numeric(6), test_error=numeric(6))
for(i in 1:6){
    lm_model <- lm(formula=Moisture~poly(Protein,i), data=train)

    predict_train <- predict.lm(lm_model, train, interval="confidence")
    mse_df$train_error[i] <- my_mse(predict_train, train$Moisture)

    predict_test <- predict.lm(lm_model, test, interval="confidence")
    mse_df$test_error[i] <- my_mse(predict_test, test$Moisture)
}
```

Present a plot showing how training and validation MSE depend on *i*.

```
library(ggplot2)
ggplot(mse_df, aes(model)) +
    geom_line(aes(y=train_error), colour="blue") +
    geom_line(aes(y=test_error), colour="red") +
    ylab("Mean Square Error") +
    xlab("Model Complexity")
```



Which model is best according to the plot?

According to the plot, the linear model (M_1) is clearly the one that better fits the data. The reason is that as the model complexity increases, the prediction error increases as well. In other words, the least complex model is the one with the lowest prediction error.

How do the MSE values change and why?

The MSE values decrease with the polynomial degree for the training data. However, on test data the error increases when the model uses higher order polynomial functions. This means that the model is overfitted when it uses high degree polynomials.

Interpret this picture in terms of bias-variance tradeoff.

The picture shows low bias and high variance, since the separation between both curves is considerably large.

Use the entire data set in the following exercises.

4. Perform variable selection of a linear model in which Fat is response and Channel1-Channel100 are predictors by using stepAIC.

```
library(MASS)
fit <- lm(formula=Fat~. -Sample -Protein -Moisture , data=tecator)
step <- stepAIC(fit, direction="both", trace=0)
```

Comment on how many variables were selected.

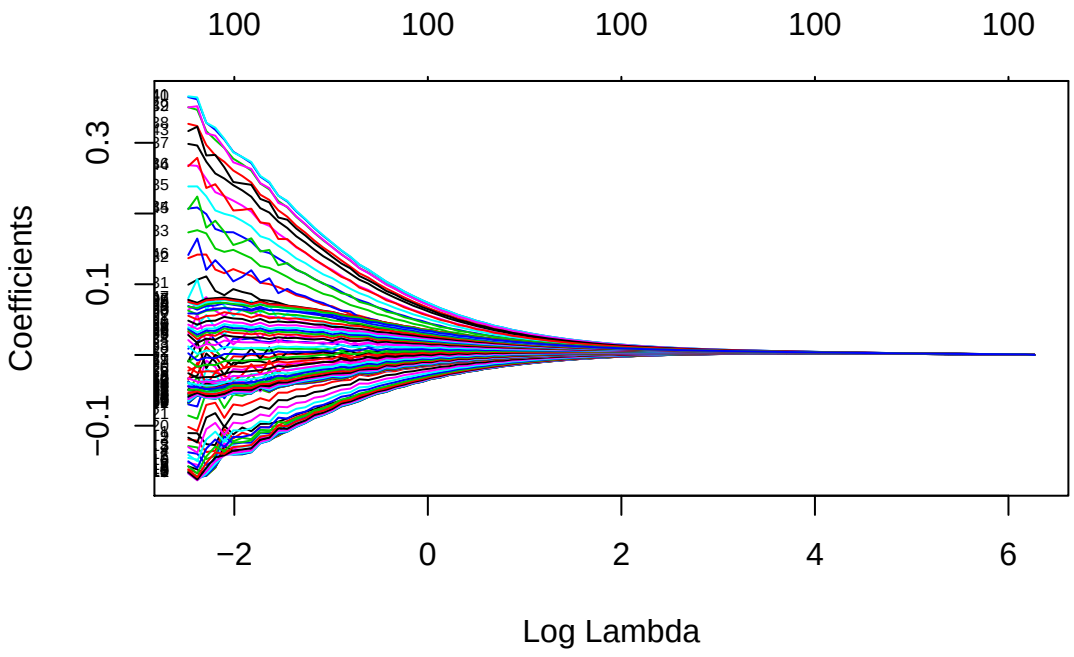
```
print(paste0("Number of selected variables: ",step$rank-1))
```

```
## [1] "Number of selected variables: 63"
```

The number of selected variables can be computed as the number of coefficients in the model minus 1, which corresponds to the intercept.

5. Fit a Ridge regression model with the same predictor and response variables. Present a plot showing how model coefficients depend on the log of the penalty factor λ.

```
library(glmnet)
covariates <- as.matrix(scale(train[,2:101])) # ignore columns "Sample", "Protein" and "Moisture"
response <- scale(train$Fat)
ridge_model <- glmnet(x=covariates, y=response, alpha=0, family="gaussian")
plot(ridge_model, xvar="lambda", label=TRUE)
```

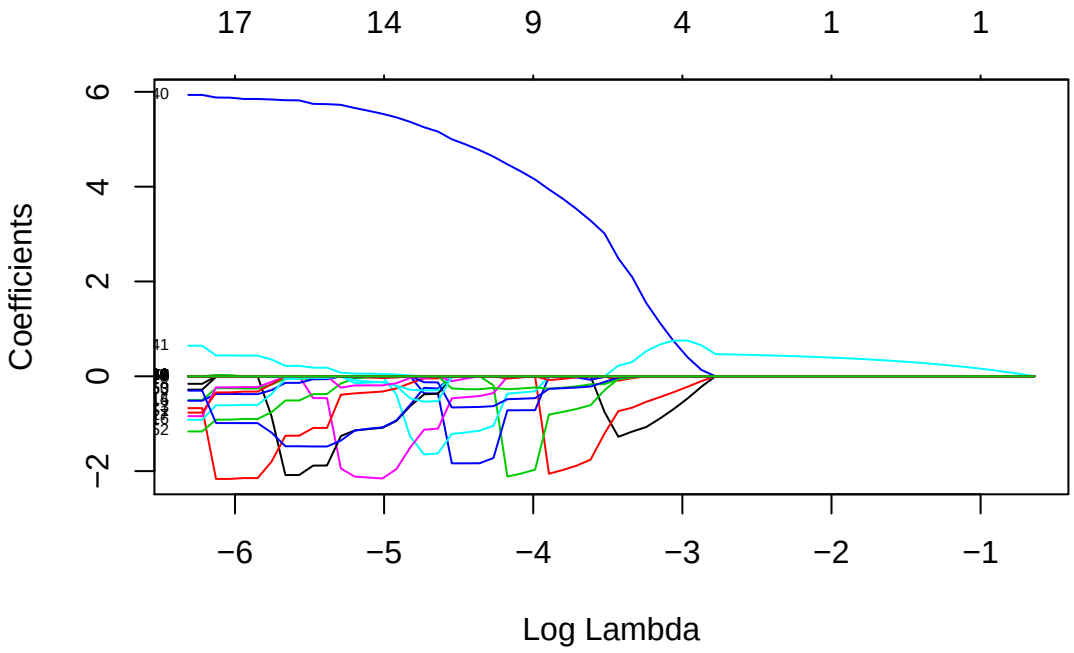


Report how the coefficients change with λ.

The coefficients get closer to zero as lambda grows.

6. Repeat step 5 but fit LASSO instead of the Ridge regression.

```
lasso_model <- glmnet(x=covariates, y=response, alpha=1, family="gaussian")
plot(lasso_model, xvar="lambda", label=TRUE)
```



Compare the plots from steps 5 and 6. Conclusions?

The main difference is that in Ridge all coefficients where converging to zero in the same way, while in LASSO they all have different behaviors as λ evolves. It looks like LASSO will always have some non-zero coefficients regardless of the λ value, whereas in Ridge for high values of λ all coefficients were zero.

7. Use cross-validation to find the optimal LASSO model (make sure that case $\lambda = 0$ is also considered by the procedure).

```
cv_lasso <- cv.glmnet(x=covariates, y=response, alpha=1, lambda=seq(0,1,0.001))
```

Report the optimal λ and how many variables were chosen by the model and make conclusions.

```
cat(paste0("Lambda value of the optimal LASSO model: ", cv_lasso$lambda.min, "\n"))
```

```
## Lambda value of the optimal LASSO model: 0
```

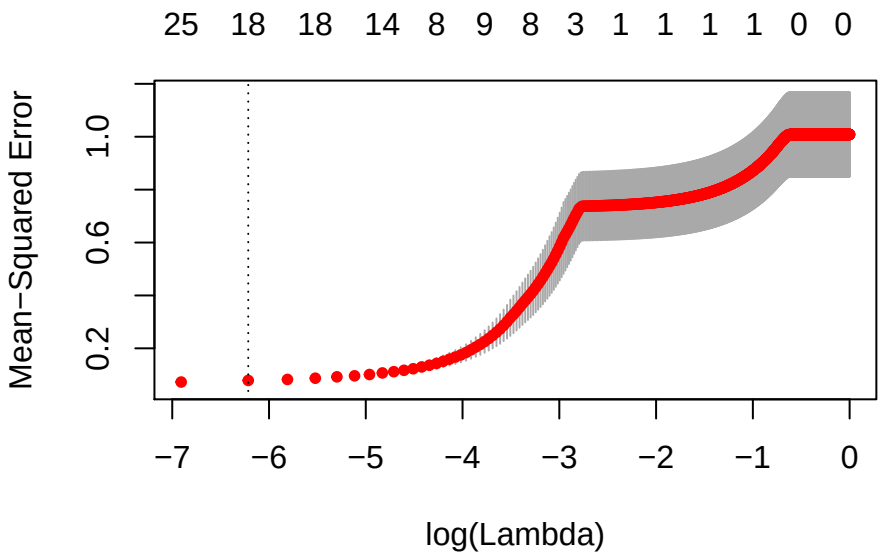
```
optimal_lasso <- glmnet(x=covariates, y=response, alpha=1, lambda=0)
cat(paste0("Number of variables chosen by the optimal LASSO model: ",optimal_lasso$df))
```

```
## Number of variables chosen by the optimal LASSO model: 100
```

Since the penalty factor is zero, all chanells will be active.

Present also a plot showing the dependence of the CV score.

```
plot(cv_lasso)
```



Comment how the CV score changes with λ.

It can be seen in the graph that the MSE is lower for low values of λ

8. Compare the results from steps 4 and 7.

In step 4 the AIC function was selecting only 63 variables, while the optimal LASSO model uses all of them.