Lab report in Machine Learing

Laboration 1

TDDE01

Oskar Andersson
oskan803@student.liu.se
Daniel Engelson
danen038@student.liu.se

Department of Statistics and Machine Learning
Institution of Datascience (IDA)
Linköpings Universitet
24-11-2019

Contents

1. Introduction

The purpose of this laboration is to practise classification with nearest neighbour models, study inference with bayesian models, and finally practise linear regression and regularization.

Assignment 1 and 2 in this report is written by Oskar Andersson.
Assignment 4 is written by Daniel Engelson.

2. Assignments

The assignments provided are in order, to find the assignments please refer to the course page for TDDE01. All code are collected in the end of this report. In the code snippets throughout this report let "train" denote results from models fitted on training-data and "test" denote results from models fitted on the test-data.

Assignment 1

Step 1

We start off by dividing the dataset spambase into a train and test part (50/50).

Step 2

Using logistic regression and classifying by the principle:
$\hat{Y} = 1,\ if\ P(Y = 1|X) > 0.5,\ otherwise\ \hat{Y} = 0,$
we get the following confusion matrixes:

```
> # Confusion matrix test
> table(test_truth, test_yh)
           test_yh
test_truth   0   1
         0 808 143
         1  92 327
>
> # Conf matrix train
> table(train_truth, train_yh)
            train_yh
train_truth   0   1
          0 804 127
          1  93 346
```

and misclassification rates:

```
> # Mislassification rate test
> missclass(test_truth, test_yh)
[1] 0.1715328
>
> # Misclassification rate train
```

```
> missclass(train_truth, train_yh)
[1] 0.1605839
```

We can see that the classification on the training data gives a smaller misclassification rate i.e. smaller amount of wrong predictions, which also shows in the confusion matrix compared to the classification on the test data. This makes sense because our model is fitted on the training data and thus the model should result in better predictions when using the training data.

Step 3

In the next excercise we increase the threshold to 0.8, thus the principle:
$\hat{Y} = 1,\ if\ P(Y = 1|X) > 0.8,\ otherwise\ \hat{Y} = 0.$
Resulting in the following confusion matrixes and misclassification rates:

```
> # Test
> test_yh_08 = generate_yh(test_probs, 0.8)
> table(test_truth, test_yh_08)
           test_yh_08
test_truth   0   1
         0 931  20
         1 314 105
> missclass(test_truth, test_yh_08)
[1] 0.2437956
>
> # Train
> train_yh_08 = generate_yh(train_probs, 0.8)
> table(train_truth, train_yh_08)
            train_yh_08
train_truth   0   1
          0 921  10
          1 333 106
> missclass(train_truth, train_yh_08)
[1] 0.250365
```

Changing the threshold gives a more strict condition to classify as spam and we classify as not spam more often, therefore we get more wrong predictions and the misclassification ratios are larger.

Step 4

Now using the standard classifier for k-nearest neighbours $kknn()$ with $K = 30$ to create a model, we get:

```
> # Test missclassification rate
> missclass(test_truth, yh2)
[1] 0.3131387
>
> # Train missclassification rate
> missclass(train_truth, yh3)
```

```
[1] 0.1671533
>
```

We see that the prediction on the test data is worse than in step 2, but the train predictions results in the same misclassification rate.

Step 5

Repeating step 4 with $K = 1$ instead results in the following:

```
> # Test missclassification rate
> missclass(test_truth, yh4)
[1] 0.3591241
>
> # Train missclassification rate
> missclass(train_truth, yh5)
[1] 0
>
```

The prediction on the test data is much worse and the prediction on the training data is perfect, an indication of a very overfitted model. The misclassification rate on the training data is zero because with $K = 1$ for any selected point in the space we just go to the closest point and that will be our correct value.

Assignment 2

Step 1

We import the dataset machines.xlsx into variable machines.

Step 2

The distribution of x is exponential. We create a loglikelihood model $log(p(x|\theta))$ from the probability model: $\mathrm{p(x}|\theta) = \theta e^{-\theta x},\ for\ x = Length.$
Plotting the loglikelihood model on θ shows:

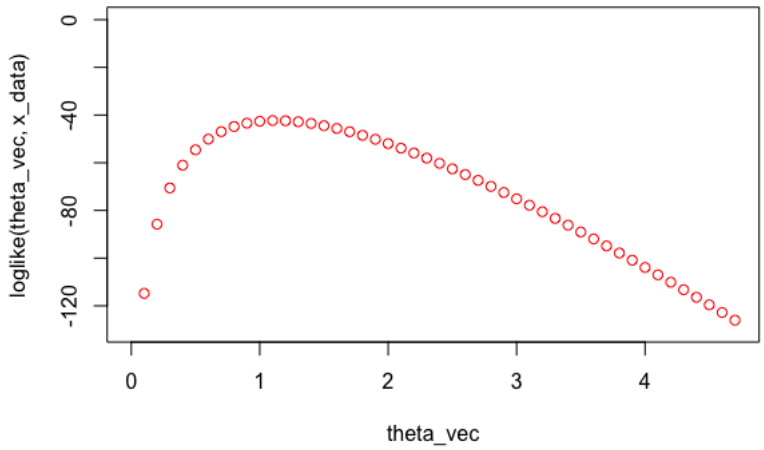

Figure 2.1: Loglikelihood model on θ

Which we can draw the conclusion that the maximum loglikelihood value is approximately -42.3 at $\theta = 1.13$.

Step 3

Creating a new model from the first six values in machines and plotting that model against θ and the model in step in the same plot:

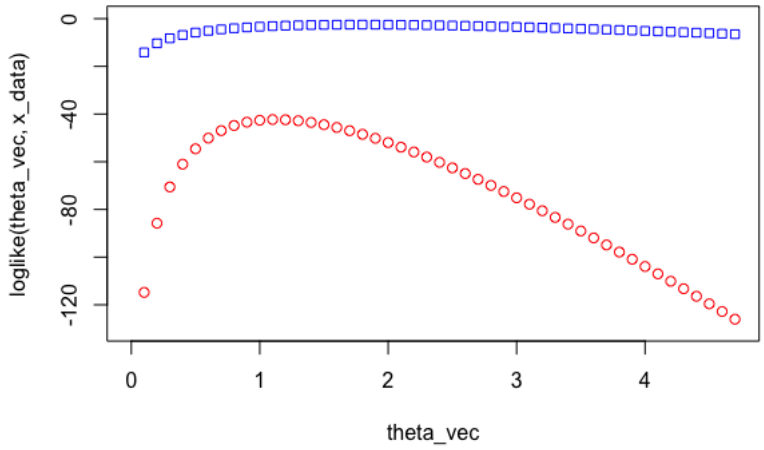

Figure 2.2: Loglikelihood on first six data (blue) and whole dataset (red)

The reliability depends on the amount of data, thus our conclusion is that the new model is less reliable than the model in step 2.

Step 4

Assuming the bayesian model:
$\mathrm{p}(\mathrm{x}|\theta) = \theta e^{-\theta x}$,
with prior:
$\mathrm{p}(\theta) = \lambda e^{-\lambda x},\ \lambda = 10$
we compute:
$\mathrm{l}(\theta) = log(p(x|\theta)p(\theta))$.
By plotting $l(\theta)$ and the model from step 2 we get:

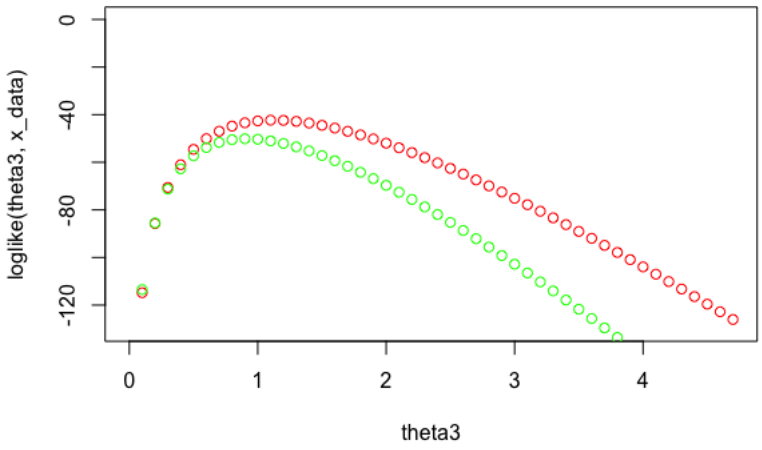

Figure 2.3: Bayesian model (green) compared to Loglikelihood from step 2 (red)

The function $l(\theta)$ is proportional to the posterior probability which is a measurement of uncertainty for our model. Optimum at $\theta = 0.9$ which is a little bit lower than previous results.

Step 5

Generating 50 new observations from randomly generated values with the created observation generator and then plot the generated observations:

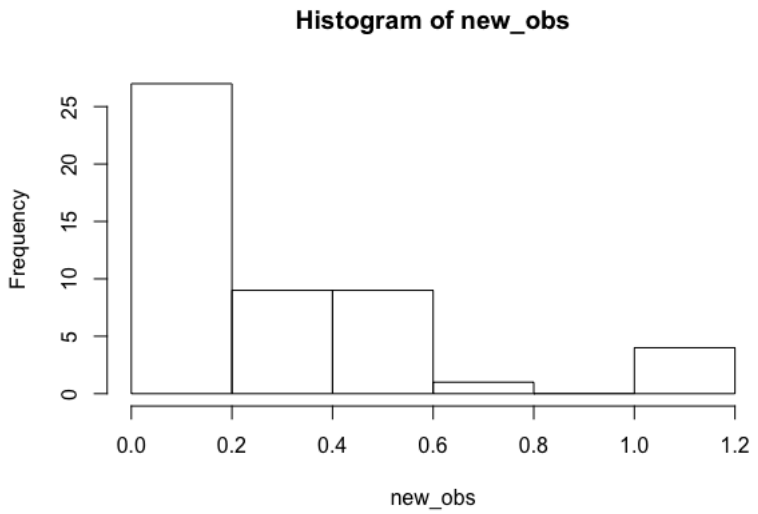

Figure 2.4: Histograms from generated observations from random values.

we compare with the real observations:

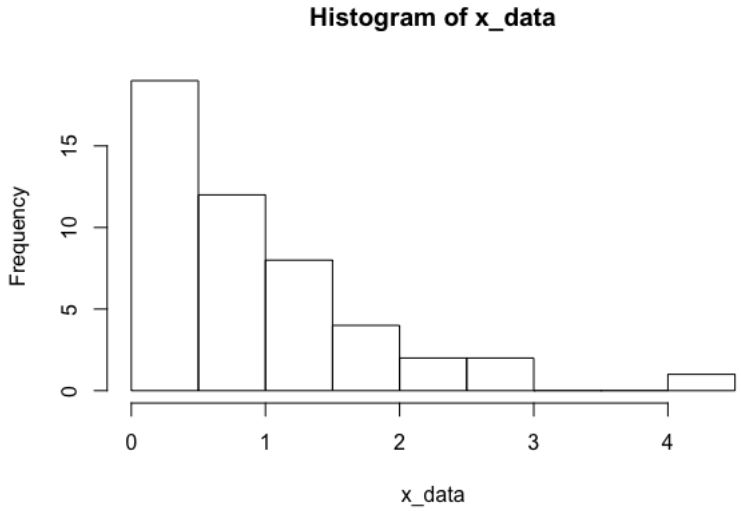

Figure 2.5: Histograms from real observations.

Since the histograms 2.5 and 2.4 are similar for the generated observations and the real observations we conclude that the estimation of θ is fairly good.

Assignment 4

Step 1

Moisture against Protein was plotted, see figure 2.6.

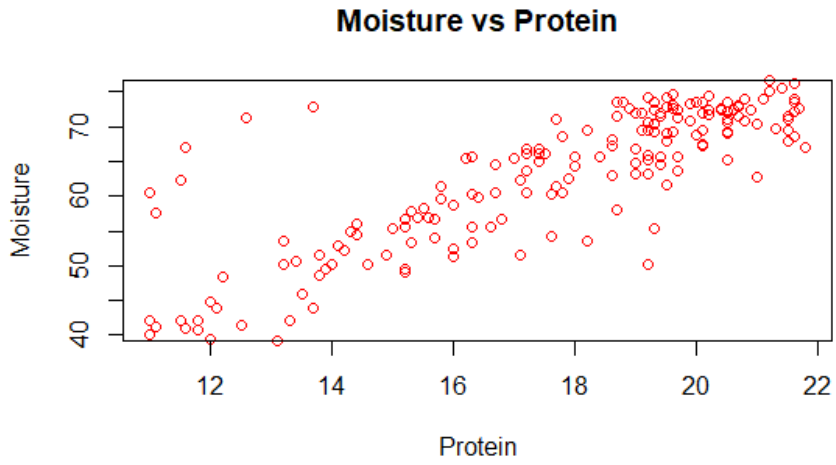

Figure 2.6: Plot of Moisture data vs Protein data

I think the data can be descriped pretty well by a linear model since most of the values are close to a diagonal line.

Step 2

The probabilistic model for M_i can be described as

$$Y \sim N(\sum_{n=0}^{i} w_i x^i, \sigma^2), \tag{2.1}$$

where Y is Moisture and x is protein data.

MSE criterion is a way to compare models with different parameters and degrees by looking at the mean square error. Since this is the case now, it is appropriate to use MSE criterion when fitting the model.

Step 3

After deviding the data into training and validation sets and fit them to polynomial functions of different degrees, MSE was used. The result is presented in figure 2.7.

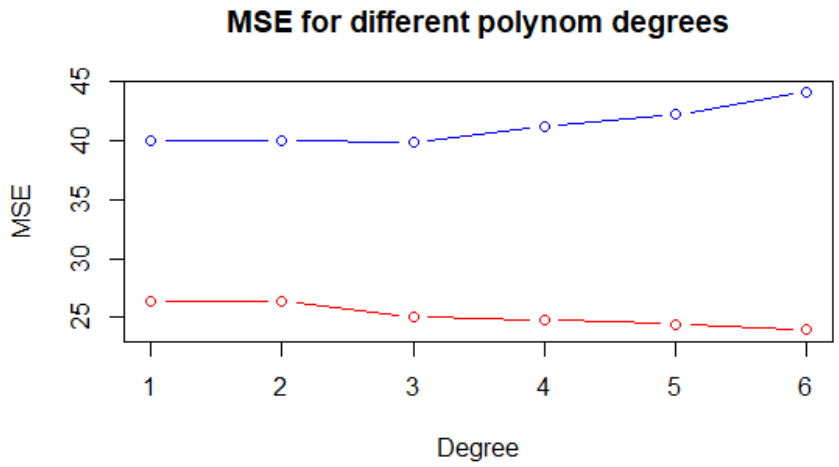

Figure 2.7: MSE for training data in red. MSE for validation data in blue

According to training data $i = 6$ is the best model. Accortding to test data $i = 3$ is the best. The reason why MSE for train data gets lower for high-degree models and MSE for test data gets higher is due to overfitting. Simple models has high bias but low varience, while advanced models has low bias and high varience. It is a trade off which model to use.

Step 4

When using stepAIC for variable selection, 63 variables were selected.

Step 5

In this task a Ridge regression model was fitted. As lambda increases the coefficiants go towards zero, see Figure 2.8.

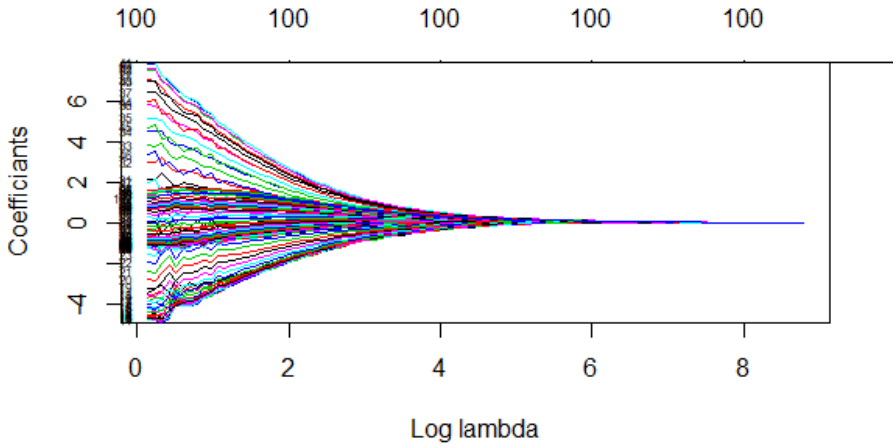

Figure 2.8: Ridge regression

Step 6

When fitting the data using LASSO the plot in figure 2.9 were given.

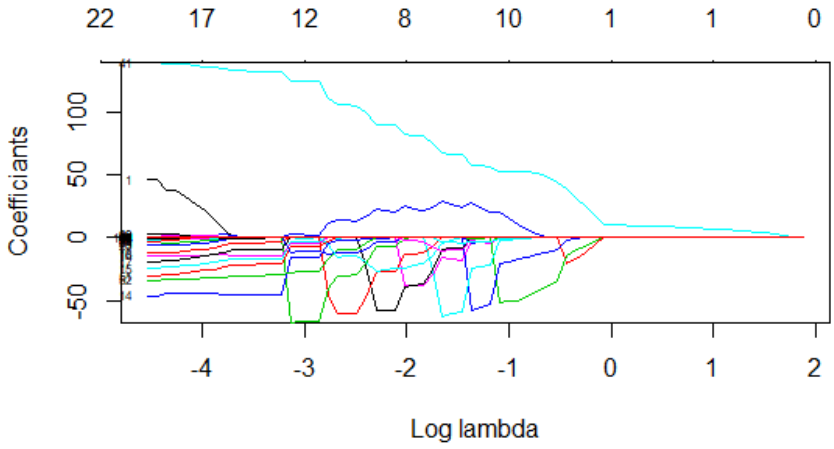

Figure 2.9: LASSO regression

The difference betwen Ridge regression and LASSO is that LASSO penalize different coefficiants for different lambdas. Unlike the Ridge regression, the coefficiants in the LASSO model can be zero for different lambdas.

Step 7

By using cross-validation and ploting the CV score against lamba, it can be shown that higher lambda values gives higher CV score, see figure 2.10. It can also be shown that the optimal LASSO model would use all 100 variables.

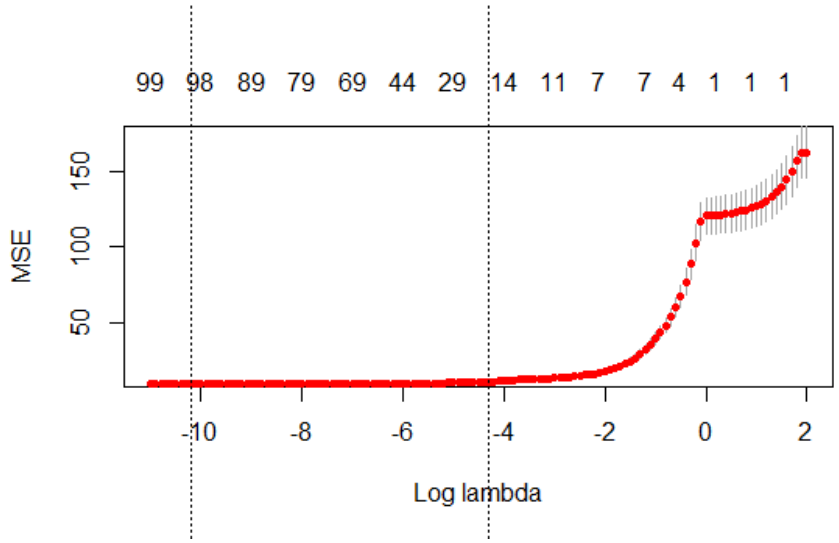

Figure 2.10: Variable selsction

Step 8

The otimal LASSO uses all varibles, while stepAIC only uses 63.

3. Code

Assignment 1

```
#-------------------------------Assignment 1-----------------------------#
library(readxl)
library(kknn)
spambase = read_excel("spambase.xlsx")

# 1)
# Divide the data into train and test
n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=spambase[id,]
test=spambase[-id,]

##############################################################################
# 2)
# Spam is final column in train data, fit for all except that one.
fitmodel = glm(Spam~., data = train, family = "binomial")

# Returns a vector of probabilities in log scale, type = response gives regular
# Probabilities of wether a mail is spam or not from test data.
test_probs = predict(fitmodel, test[,-49], type = "response")

# Probabilities on trainingdata
train_probs = predict(fitmodel, train[,-49], type = "response")

# Generate a Yh vector from probability vector
generate_yh = function(probabilities, limit)
{
  over = which (probabilities > limit)
  under = which (probabilities < limit)

  probabilities[over] = 1
  probabilities[under] = 0

  return(probabilities)
}
```

```
# Predictions
test_yh = generate_yh(test_probs, 0.5)
train_yh = generate_yh(train_probs, 0.5)

# Truths
test_truth = test[['Spam']]
train_truth = train[['Spam']]

# Confusion matrix test
table(test_truth, test_yh)

# Conf matrix train
table(train_truth, train_yh)

# Missclassification rates, the amount of wrong predictions.
missclass = function(X,X1)
{
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

# Mislassification rate test
missclass(test_truth, test_yh)

# Misclassification rate train
missclass(train_truth, train_yh)

##############################################################################
# 3)

# Test
test_yh_08 = generate_yh(test_probs, 0.8)
table(test_truth, test_yh_08)
missclass(test_truth, test_yh_08)

# Train
train_yh_08 = generate_yh(train_probs, 0.8)
table(train_truth, train_yh_08)
missclass(train_truth, train_yh_08)

# Effect of new rule:
# The condition is more strict, thus more wrong predictions and the ratio is higher.

##############################################################################
# 4)

library(kknn)

# Create k nearest neighbour model from training data. K = 30 as proposed.
```

```
m2 = kknn(formula = Spam~., train, test, k = 30)
prob2 = unlist(m2['fitted.values'], use.names=FALSE)

# Model for train data
m3 = kknn(formula = Spam~., train, train, k = 30)
prob3 = unlist(m3$fitted.values, use.names=FALSE)

# Test data predictions
yh2 = generate_yh(prob2, 0.5)

# Train data predictions
yh3 = generate_yh(prob3, 0.5)

# Truths
test_truth = test[['Spam']]
train_truth = train[['Spam']]

# Test missclassification rate
missclass(test_truth, yh2)

# Train missclassification rate
missclass(train_truth, yh3)

# Test prediction is worse than in task 2), but train is the same.

##########################################################################################
#5)


# Create k nearest neighbour model from training data. K = 1 as proposed.
m4 = kknn(formula = Spam~., train, test, k = 1)
prob4 = unlist(m4['fitted.values'], use.names=FALSE)

# Model for train data
m5 = kknn(formula = Spam~., train, train, k = 1)
prob5 = unlist(m5$fitted.values, use.names=FALSE)

# Test data predictions
yh4 = generate_yh(prob4, 0.5)

# Train data predictions
yh5 = generate_yh(prob5, 0.5)

# Truths
test_truth = test[['Spam']]
train_truth = train[['Spam']]

# Test missclassification rate
missclass(test_truth, yh4)
```

```
# Train missclassification rate
missclass(train_truth, yh5)

# As expected with one nearest neighbour on the training set the error will be zero.
# But as a consequence overfitting and the error from the test data is larger.
# We only select the 1 nearest neighbour which lead to the correct value.
```

Assignment 2

```
#------------------------------Assignment 2------------------------------#
################################################################################
# 1)
library(readxl)
machines = read_excel("machines.xlsx")
################################################################################
# 2)

# x is exponential distributed

# Create p(x|theta)
pthetax = function(theta, x)
{
  return(theta * exp(- theta * x))
}

# Can use sum(log(pthetax(theta, x_vec))) instead of loop below
# Log likelihood function
loglike = function(theta, x_vec)
{
  loglike = 0
  for (xi in x_vec)
  {
    loglike = loglike + log(pthetax(theta, xi))
  }
  return(loglike)
}

# Generate a theta vector to plot against
theta_vec = seq(0, 4.7, 0.1)

# Parse data to run loglike on
x_data = machines$Length

# Plot
plot(theta_vec, loglike(theta_vec, x_data), col="Red", ylim = c(-130, 0))

# Maximum at theta = 1.13, max = -42.30. Using theta_vec = = seq(1.0, 1.2, 0.01)
```

```
#################################################################################################
#3)

# Get the first 6 data points
first_six = x_data[1:6]

# From 2)
plot(theta_vec, loglike(theta_vec, x_data), col = "Red", ylim = c(-130, 0))

# Add to prev plot
points(theta_vec, loglike(theta_vec, first_six), pch = 22, col = "Blue")

#plot(seq(1.2,2.2,0.01), loglike(seq(1.2,2.2,0.01), first_six), pch = 22, col = "Blue")
# Uncomment the plot closest above to see maximum is around theta = 1.8

# The model is more reliable with more data, thus 2) is better than 3).

#################################################################################################
#4)

# Calculate p(theta)
prior = function(theta)
{
  lambda = 10
  return (lambda * exp(- (lambda * theta)))
}

# Calculate l(theta) using pthetax above and prior
l_th = function(x_vec, thetas)
{
  return (loglike(thetas, x_vec) + log(prior(thetas)))
}

# Thetas to plot against
theta3 = seq(0, 4.7, 0.1)

# From 2)
plot(theta3, loglike(theta3, x_data), col="Red", ylim = c(-130, 0))

# Add new plot
points(theta3, l_th(x_data, theta3), pch = 21, col = "Green")

# Maximum for l(theta)
theta3[which.max(l_th(x_data, theta3))]

# Gives max at theta = 0.9

#################################################################################################
# 5)
```

```
# The distribution for the observations
obs_gen = function(x_val)
{
  # From 2)
  theta = 1.13
  return(theta * exp(- theta * x_val))
}

# Generate random values in a vector of size 50
set.seed(12345)
rand_x = 0.001 * sample(0:4000, 50, replace = TRUE)

# Generate new observations according to distribution
new_obs = obs_gen(rand_x)

# Plot generated observations
hist(new_obs)

# Plot real observations
hist(x_data)

# Conclusion: similar histograms indicates that our model and our estimation of theta is
# fairly good.
```

Assignment 4

```
library(glmnet)
library(MASS)

#Exercise 1
data = read.csv("~/TDDE01/tecator.csv")

mois = as.vector(data[,"Moisture"])
prot = as.vector(data[,"Protein"])

plot(prot, mois, ylab = "Moisture", xlab = "Protein", main = "Moisture vs Protein", col = "

#Exercise 2

#Exercise 3

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

mse_train = numeric()
mse_test = numeric()
```

```
for (i in 1:6){
  M = lm(Moisture ~ poly(Protein, i), data = train)
   predictTrain = predict(M, newdata = train)
   predictTest = predict(M, newdata = test)

   mse_train[i] = mean((train$Moisture-predictTrain)^2)
   mse_test[i] = mean((test$Moisture-predictTest)^2)
}

plot (1:6, mse_train, type="b", ylim=c(23,45), ylab = "MSE", xlab = "Degree", main = "MSE fo
points (1:6, mse_test, type ="b" , col = "Blue")

#Exercise 4
Channel = data[,2:101]
fat = as.vector(data[,"Fat"])

fit = lm(fat ~ .,data=data.frame(Channel))
step = stepAIC(fit, direction="both")

#Exercise 5

RidgeM=glmnet(as.matrix(Channel), fat, alpha=0,family="gaussian")
plot(RidgeM, xvar="lambda", label=TRUE, ylab = "Coefficiants", xlab = "Log lambda")

#Exercise 6

LassoM=glmnet(as.matrix(Channel), fat, alpha=1,family="gaussian")
plot(LassoM, xvar="lambda", label=TRUE, ylab = "Coefficiants", xlab = "Log lambda")

#Exercise 7

OptLassoM=cv.glmnet(as.matrix(Channel), fat, alpha=1, family="gaussian", lambda= c(0, exp(se
OptLassoM$lambda.min
plot(OptLassoM, ylab = "MSE", xlab = "Log lambda", )
coef(OptLassoM, s="lambda.min")
```