Lab1 - TDDE01

Anton Gefvert

11/18/2018

Assignment 1

1.2

Confusion matrix for training set:

Table 1: Confusion matrix for training set using threshold 0.5

	Actual 0	Actual 1
0	803	142
1	81	344

Confusion matrix for test set:

Table 2: Confusion matrix for test set using threshold 0.5

	Actual 0	Actual 1
0	791	146
1	97	336

As we can see in Table 1 and Table 2, the results using the formula

$$\hat{Y} = \text{if } p(Y = 1|X) > 0.5, \text{ otherwise } \hat{Y} = 0$$

gives the following statisticts:

- Train
  - 142 false positives
  - 81 false negatives
  - total we have 223 out of 1370 wrongful classification
  - missclassification: $223/1370 = 0.163 = 16.3\%$
- Test
  - 146 false positives
  - 97 false negatives
  - total we have 243 out of 1370 wrongful classification
  - missclassification: $243/1370 = 0.177 = 17.7\%$

1.3

Confusion matrix for training set:

Table 3: Confusion matrix for train set using threshold 0.9

	Actual 0	Actual 1
0	944	1
1	419	6

Confusion matrix for test set:

Table 4: Confusion matrix for test set using threshold 0.9

	Actual 0	Actual 1
0	936	1
1	427	6

What we see from Table 3 and Table 4, is that when we increase the constraint for marking an email as spam, we get way less false positives (1 instead of 50 in test set), but the false negatives are increased.

Missclassifications: * Train: 420/1370 = 0.306 = 30.6% * Test: 434/1370 = 0.312 = 31.2%

As we can see, the missclassification is increased, but you could argue that in this case, marking a non-spam email as spam, is way worse than letting through spam. It's easier for the recipient to filter away less spam, than to check through the spam mail to fins non-spam emails among them. In this case though, there are barely any true positives, so this "spam-filter" will pretty much do nothing.

1.4

Confusion matrix for train set

Table 5: Confusion matrix for train set using knn with k = 30

	Actual 0	Actual 1
0	807	138
1	98	327

Confusion matrix for test set

Table 6: Confusion matrix for test set using knn with k = 30

	Actual 0	Actual 1
0	672	265
1	187	246

As we can see from Table 5 and Table 6 we get the following missclassification rates:

- Train: 236/1370 = 0.172 = 17.2%

- Test: 452/1370 = 0.329 = 32.9%

What we see here is that missclassification rate for train set is way lower than in exercise 2, which is reasonable since those will be the nodes that can be neighbors. We see that the missclassification rate for the test set has gone up compared to exercise 2. In both cases, the false positive rate has gone up, which is problematic for this spam classification.

Task 5 - knn, k = 1

Confusion matrix for train set

Table 7: Confusion matrix for train set using knn with k = 1

	Actual 0	Actual 1
0	945	0
1	0	425

Confusion matrix for test set

Table 8: Confusion matrix for test set using knn with k = 1

	Actual 0	Actual 1
0	640	297
1	177	256

As we can see from Table 7 and Table 8 we get the following missclassification rates: * Train: 0/1370 = 0% * Test: 474/1370 = 0.346 = 34.6%

As can be seen in Table 7, using k = 1, makes this model perfect. This is simply because we are using the model to classify the same dataset that is datapoints, meaning the closest neighbor will be the node itself and since there is only one neighbor, we will perfectly match each node.

The test set gets slightly worse than in exercise 4.

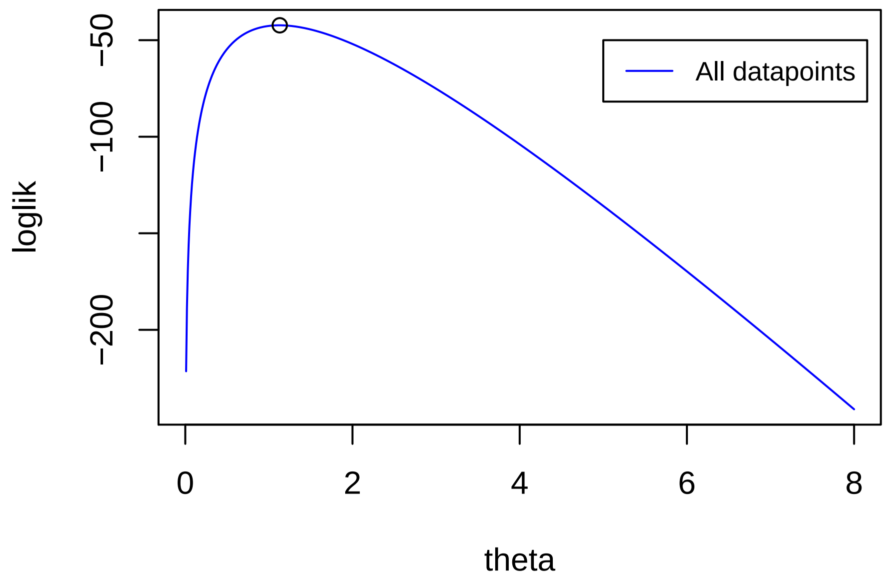


Figure 1: log likelyhood for task 2

Assignment 2

2.2

The distribution type of $p(x|\theta)$ is a exponential distribution, a distribution that models the time between events in a poisson distribution.

As seen in Figure 1, the maximum loglikelyhood is given for $\theta = 1.13$ and gives the value $loglik \approx -42.3$

2.3

As seen in Figure 2, the maximum log likelyhood with just the first six variables from x (red line), is given for $\theta = 1.79$ and gives the value $loglik \approx -2.52$.

What we can see from these figures is that the loglikelyhood has a way pointier graf in the case where all values of x is used, meaning it's more reliable (which is reasonable, since we have more datapoints). Whereas in the case with only the first sic values of x, the graph is much more flat top and with more values close to the maximum loglikelyhood.

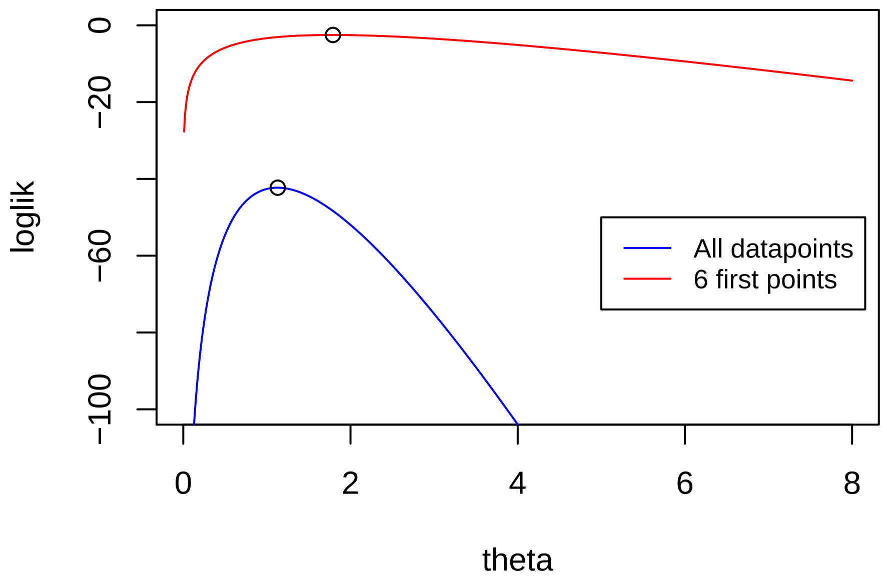


Figure 2: log likelyhood for task 3

2.4

$$L(\theta) \propto p(\theta|x) \propto p(x|\theta)p(\theta) \propto \theta \left(\prod_{i=1}^{n} e^{-\theta x_i} \right) 10e^{-10\theta}$$

Gives us

$$l(\theta) \propto ln \left[\left(\prod_{i=1}^{n} e^{-\theta x_i} \right) 10e^{-10\theta} \right]$$

The new loglikelihood based on a prior is maximized at $\theta = 1.79$ and gives $loglik \approx -2.51$, if we compare this to the result given when not using a prior, we get about the same loglikelihood and a similar θ, albeit a bit smaller. If we look at the graph in Figure 3, we see that the one using a prior has a smaller peak, this shows it has a higher certainty (given the conditions) of what θ is.

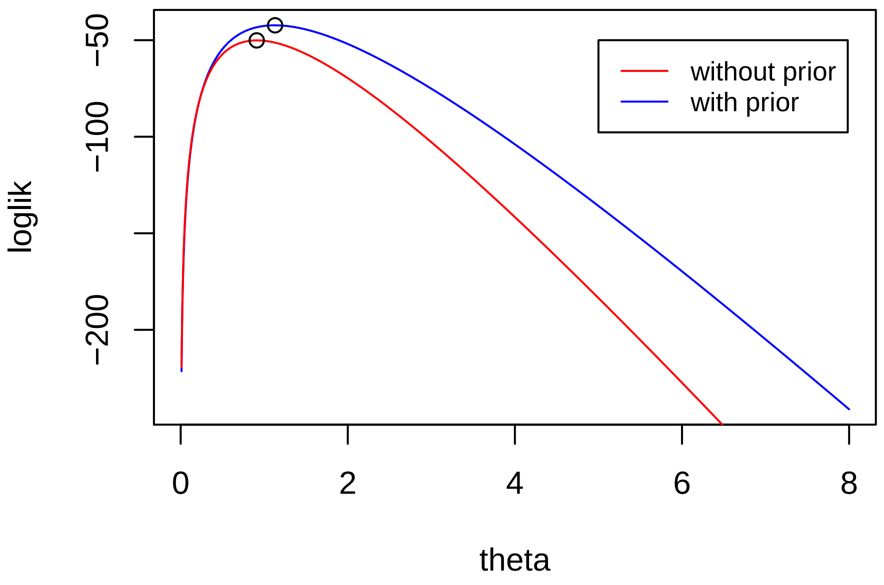
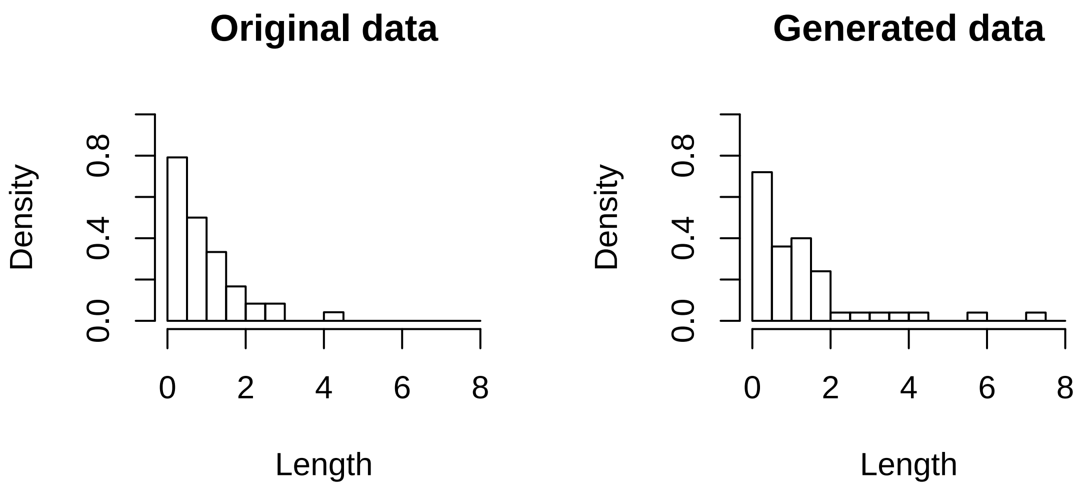


Figure 3: log likelyhood for task 4

2.5



As seen in the above figures(couldn't for the life of me get captions to work with two histograms), we see that the distributions are similiar.

They are not identical though, the value of the original data follows an exponential curve much more clearly, in the generated data.

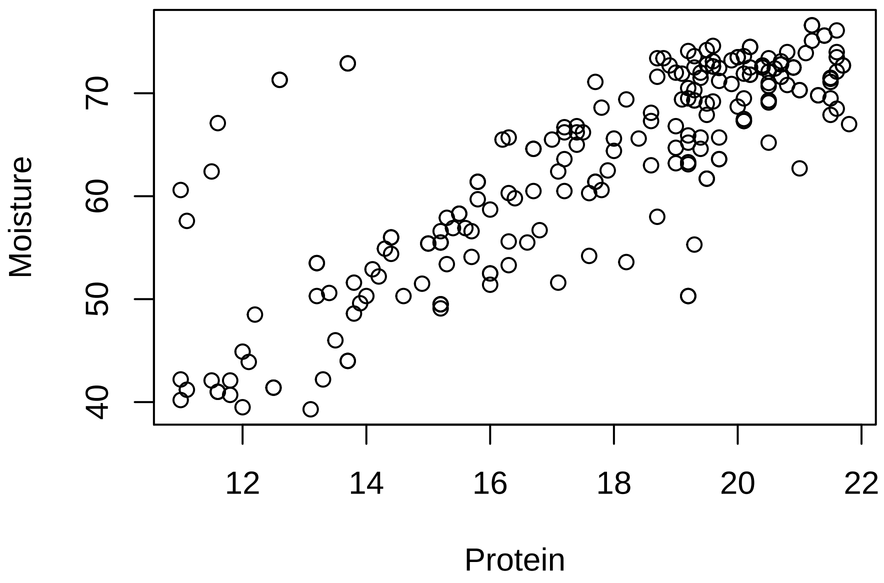


Figure 4: Moisture vs protein

Assignment 4

4.1

Looking at Figure 4, 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!

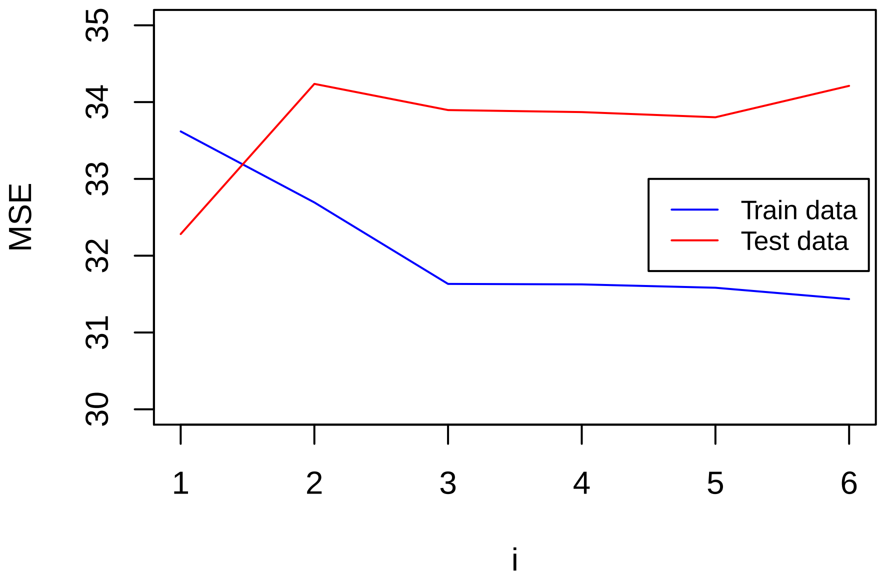


Figure 5: MSE for diffrent i in M_i

4.3

As seen if Figure 5 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 6, we see that when lambda gets larger, the coeffiecents converge towards zero.

4.6

When we compare LASSO in Figure 7 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).

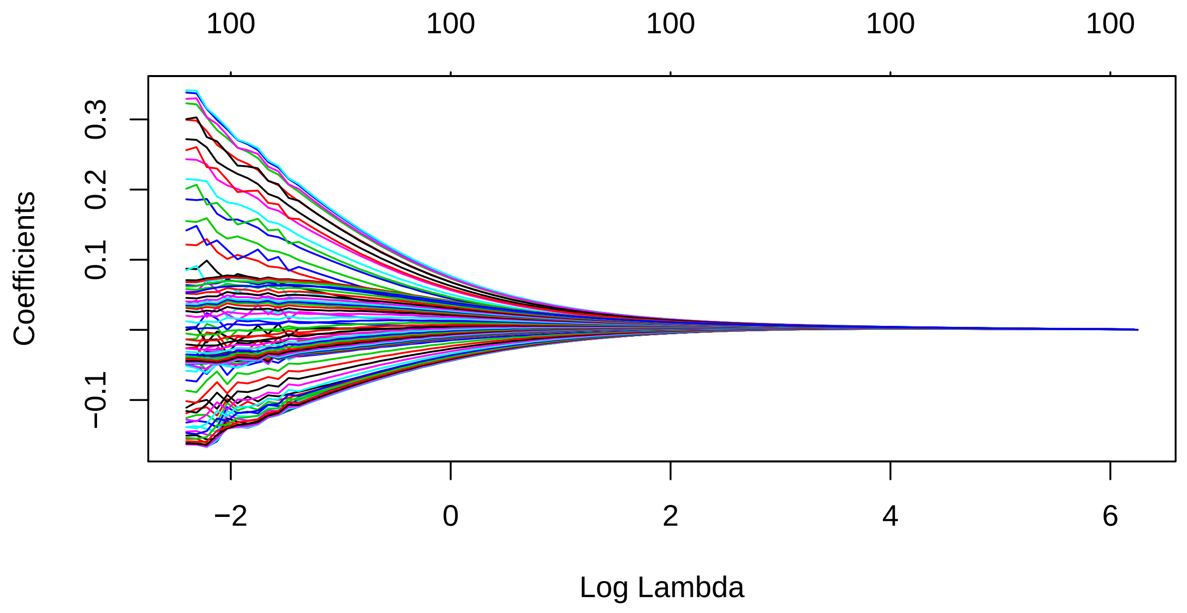


Figure 6: ridge coefficients over lambda

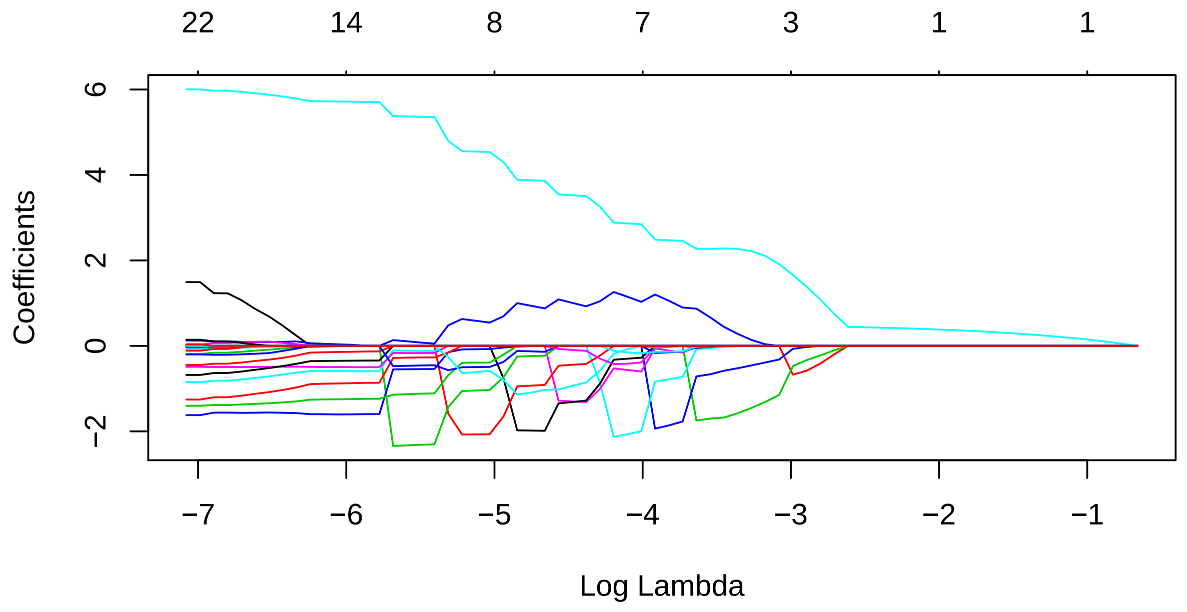


Figure 7: lasso coefficients over lambda

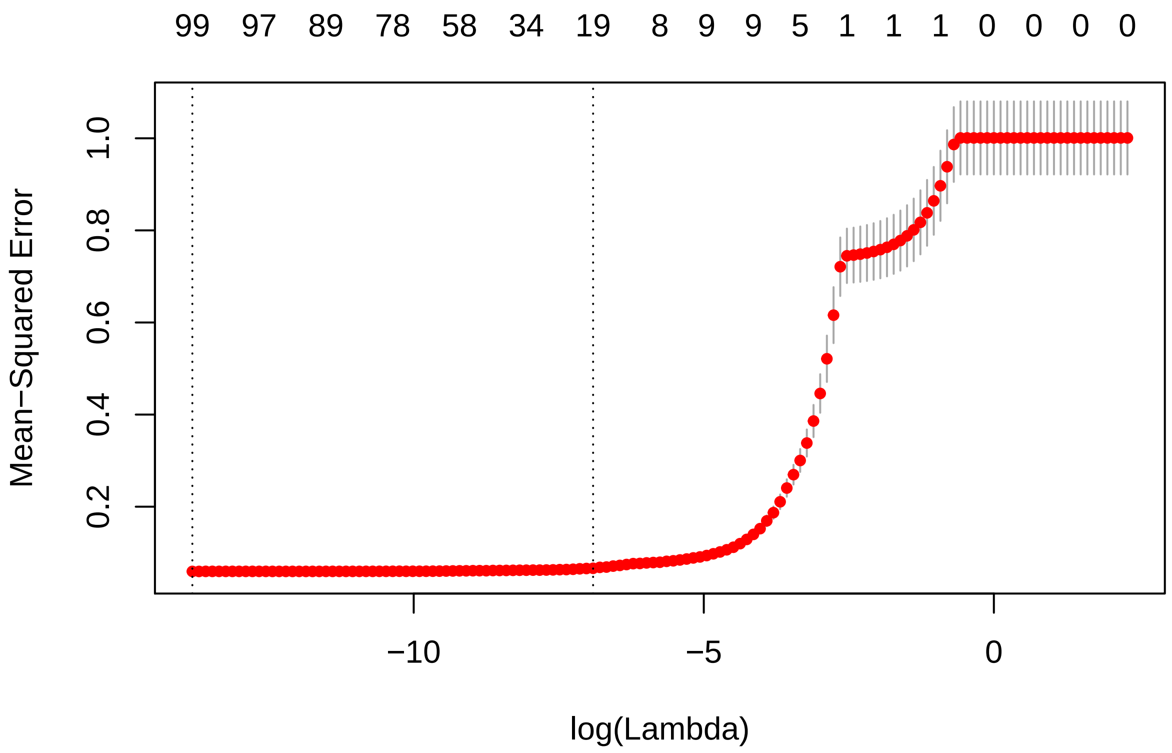


Figure 8: Cross-validation of LASSO

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 8, 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

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

#1.1
data1 <- read_excel("spambase.xlsx")

n=dim(data1)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train1 = data1[id,]
test1 = data1[-id,]

#1.2
trained = glm(Spam ~ ., data = train1, family=binomial())
pred_train = predict(trained,  newdata = train1, type="response")
pred_test = predict(trained,  newdata = test1, type="response")

#Confusion matrix training
y_hat_tr1 = as.numeric(pred_train > 0.5)
kable(table(t(train1[,49]), y_hat_tr1),
      caption = "Confusion matrix for training set using threshold 0.5",
      col.names = c("Actual 0", "Actual 1"))

#Confusion matrix test
y_hat_te1 = as.numeric(pred_test > 0.5)
kable(table(t(test1[,49]), y_hat_te1),
      caption = "Confusion matrix for test set using threshold 0.5",
      col.names = c("Actual 0", "Actual 1"))
#1.3 training
y_hat_tr2 = as.numeric(pred_train > 0.9)
kable(table(t(train1[,49]),y_hat_tr2),
      caption = "Confusion matrix for train set using threshold 0.9",
      col.names = c("Actual 0", "Actual 1"))

#test
y_hat_te2 = as.numeric(pred_test > 0.9)
kable(table(t(test1[,49]), y_hat_te2),
      caption = "Confusion matrix for test set using threshold 0.9",
      col.names = c("Actual 0", "Actual 1"))

#1.4 train
res = kknn(Spam ~ ., train1, train1, k = 30)
y_hat_tr3 = as.numeric(res[["fitted.values"]] > 0.5)
kable(table(t(train1[,49]), y_hat_tr3),
      caption = "Confusion matrix for train set using knn with k = 30",
      col.names = c("Actual 0", "Actual 1"))
```

```r
#test
res = kknn(Spam ~ ., train1, test1, k = 30)
y_hat_te3 = as.numeric(res[["fitted.values"]] > 0.5)
kable(table(t(test1[,49]), y_hat_te3),
      caption = "Confusion matrix for test set using knn with k = 30",
      col.names = c("Actual 0", "Actual 1"))

#1.5 train
res = kknn(Spam ~ ., train1, train1, k = 1)
y_hat_tr3 = as.numeric(res[["fitted.values"]] > 0.5)
kable(table(t(train1[,49]), y_hat_tr3),
      caption = "Confusion matrix for train set using knn with k = 1",
      col.names = c("Actual 0", "Actual 1"))

#test
res = kknn(Spam ~ ., train1, test1, k = 1)
y_hat_te3 = as.numeric(res[["fitted.values"]] > 0.5)
kable(table(t(test1[,49]), y_hat_te3),
      caption = "Confusion matrix for test set using knn with k = 1",
      col.names = c("Actual 0", "Actual 1"))
#2.1
data2 <- read_excel("machines.xlsx")

#2.2
lpx = function(x_seq, theta){
  return(log(prod(theta * exp(-theta * x_seq))))
}

ll1 = function(x_seq, theta_seq){
  return(sapply(theta_seq, function(t) lpx(x_seq, t)))
}

thetas = seq(0, 8, 0.01)
logs1 = ll1(data2, thetas)

plot(thetas, logs1, type="n", xlab = "theta", ylab = "loglik")
lines(thetas, logs1, col="blue")
legend(5, -50, legend=c("All datapoints"),
       col=c("blue"), lty=1:2, cex=0.8)

maxindex = match(max(logs1), logs1)
maxlog = logs1[maxindex]
maxtheta = thetas[maxindex]
points(maxtheta, maxlog)

#2.3
thetas = seq(0, 8, 0.01)
logs1 = ll1(data2, thetas)


logs2 = ll1(data2[0:6,], thetas)
y = c(-100, 0)
plot(thetas, logs2, type="n", xlab = "theta", ylab = "loglik", ylim=y)
```

```r
lines(thetas, logs1, col="blue")
lines(thetas, logs2, col="red")

legend(5, -50, legend=c("All datapoints", "6 first points"),
       col=c("blue", "red"), lty=1, cex=0.8)

maxindex = match(max(logs1), logs1)
maxlog = logs1[maxindex]
maxtheta = thetas[maxindex]
points(maxtheta, maxlog)

maxindex = match(max(logs2), logs2)
maxlog = logs2[maxindex]
maxtheta = thetas[maxindex]
points(maxtheta, maxlog)


#3.4
calc_posterior = function(x_seq, theta){
  return(log(prod(theta * exp(-theta * x_seq)) * 10 * exp(-10 * theta)))
}

posteriors = sapply(thetas, function(t) calc_posterior(data2, t))

plot(thetas, logs1, type="n", xlab = "theta", ylab = "loglik")
lines(thetas, logs1, col="blue")
lines(thetas, posteriors, col="red")
legend(5, -50, legend=c("without prior", "with prior"),
       col=c("red", "blue"), lty=1, cex=0.8)

maxindex = match(max(logs1), logs1)
maxlog = logs1[maxindex]
maxtheta = thetas[maxindex]
points(maxtheta, maxlog)

maxindex = match(max(posteriors), posteriors)
maxlog = posteriors[maxindex]
maxtheta = thetas[maxindex]
points(maxtheta, maxlog)

#3.5
set.seed(12345)
random_x = rexp(50, maxtheta)
data_numeric = as.numeric(as.character(data2$Length))
hist(data_numeric, probability=TRUE,breaks=seq(0,8,0.5),
     ylim=c(0,1), xlab="Length", main="Original data")
hist(random_x, probability=TRUE,breaks=seq(0,8,0.5),
     ylim=c(0,1), xlab="Length", main="Generated data")
#4.1
data3 <- read_excel("tecator.xlsx")

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

```r
#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)
```