Lab1 Group C01

David Nyberg (davny376) Shahin Salehi Marzi Jarani (shasa714) Yunhee Kim (yunki172)

Lab 1

Assignment 1

1) Read data into R using read.xlsx
2) Using logistic regression for spam classification we can see that if we choose 0.5 as our limit for the classifier we get fairly bad results in our confusion matrix.

```
##
## train_pred.5   0   1
##            0 875 156
##            1  56 283
```

```
## [1] 0.1547445
```

```
##
## test_pred.5   0   1
##           0 865 162
##           1  86 257
```

```
## [1] 0.1810219
```

3) When we change our classifier to 0.8 or above then we then start to get data that give us a higher missclassification rate shown below. This is because the new threshold makes the model predict something as not spam easier compared to a lower threshold.

```
##
## train_pred.8   0   1
##            0 898 216
##            1  33 223
```

```
## [1] 0.1817518
```

```
##
## test_pred.8   0   1
##           0 892 229
##           1  59 190
```

```
## [1] 0.210219
```

4a) Missclassification rates of kknn with k = 30.

```
## [1] 0.1671533
```

```
## [1] 0.3131387
```

4b) Missclassification rates of kknn with k = 1.

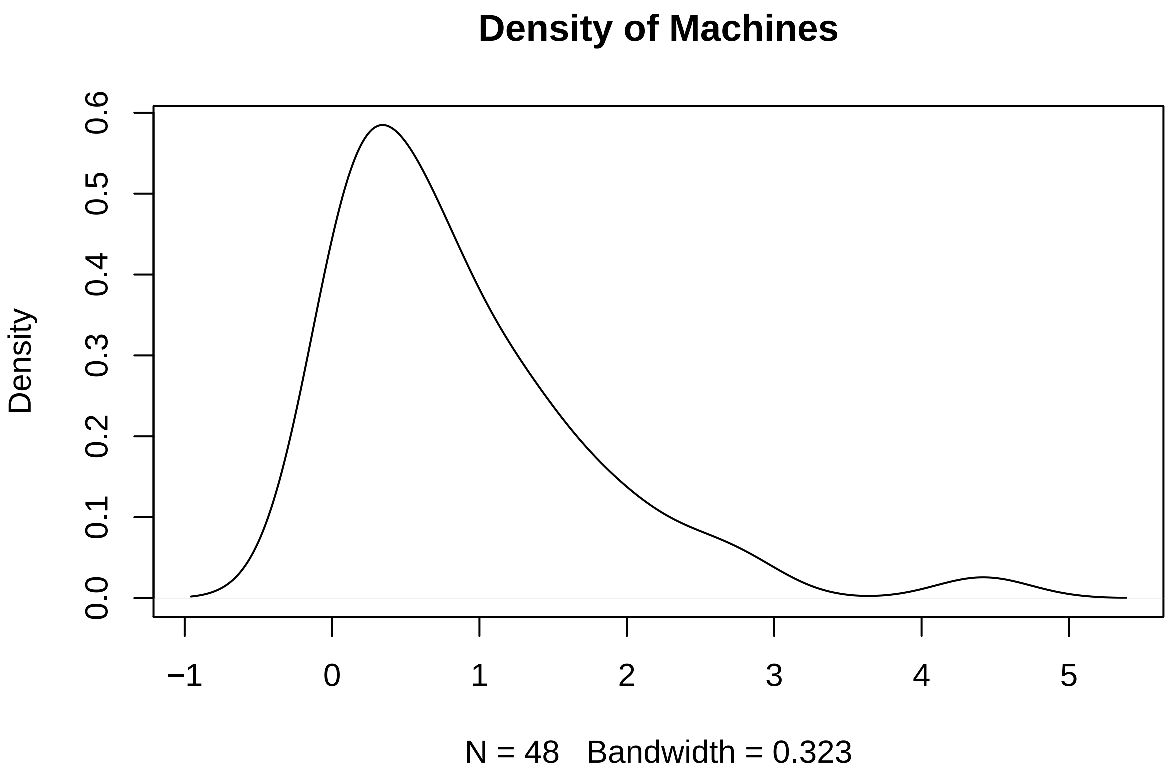
```
## [1] 0
```

```
## [1] 0.3591241
```

Based on these results you can see that fitting knn with a neighborhood of 1 overfits the data with an error of zero. It makes every single data point a neighborhood with itself and will have zero missclassifications. But testing this on unseen data with have a bad performance of 36% missclassification because it is highly biased to its training data and fails to when generalizing to new data.

Assignment 2

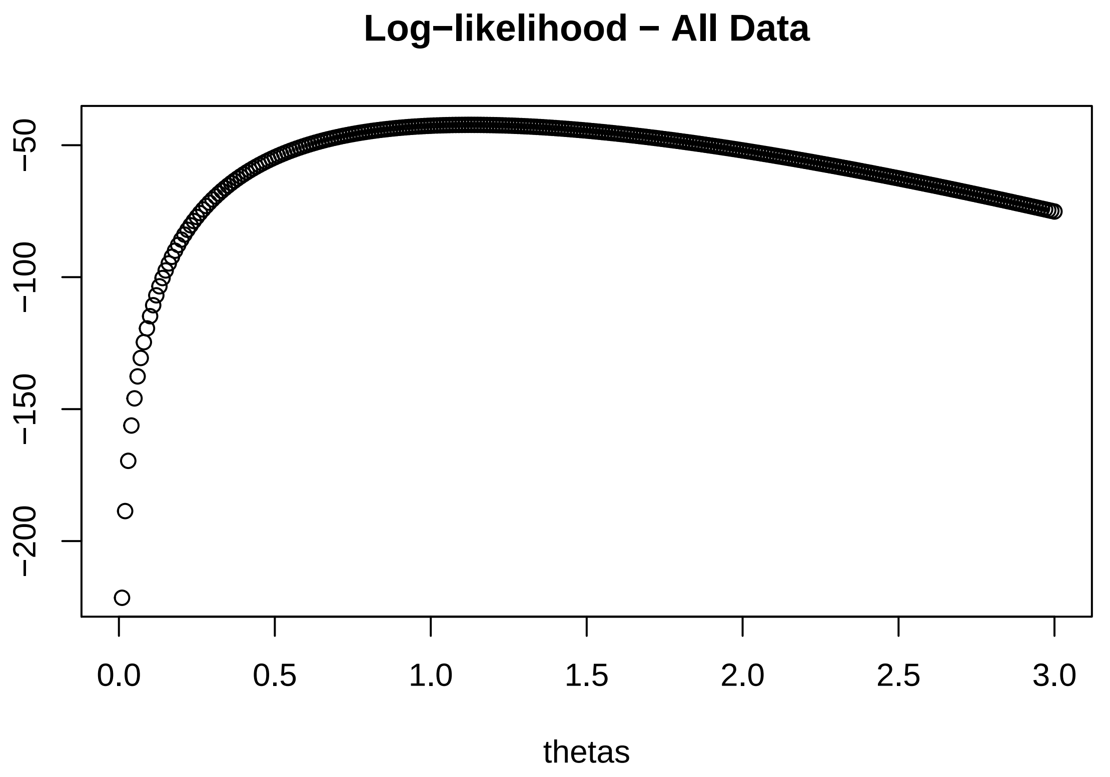
1) Read machines into R, plotting the density of the data we can see an exponential type of distrubution as we can see we have the majority of our data in the 0 - 1 range.



2) We also know this is an exponential distribution based off the given probability density function. The function log_likehood returns the log likelihood based off of a simplified formula found at https://www.statlect.com/fundamentals-of-statistics/exponential-distribution-maximum-likelihood.

```
log_likehood <- function(data, theta) {
  return( length(data) * log(theta) - theta * sum(data) )
}

seq1 <- seq(0, 3, by=.01)
plot(seq1, log_likehood(machines$Length, seq1), main = "Log-likelihood - All Data", ylab = "", xlab="th
```

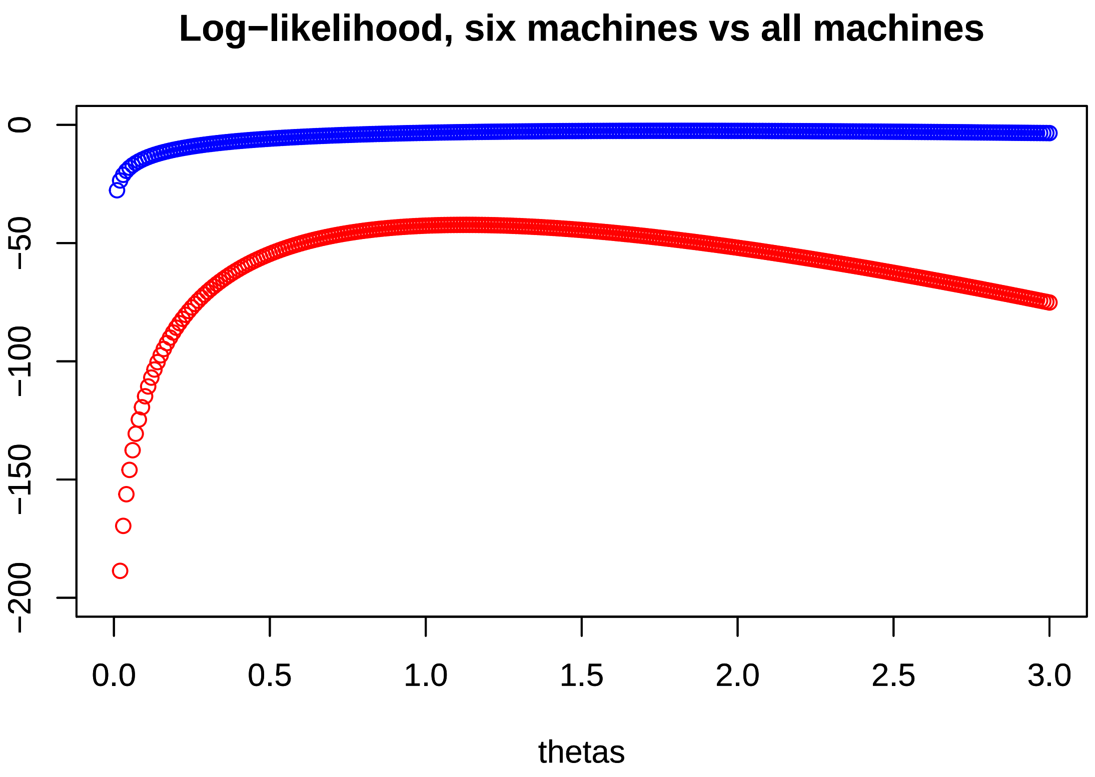


From this plot we can see that the maximum from our sequence is right around one, but it's hard to tell exactly the value from the plot. Using a max function we can see the exact value of lambda which is around 1.12.

```r
#returns the max likelihood estimate for theta
theta_max_est <- function(data) {
  return( length(data) / sum(data))
}
theta_max_est(machines$Length)
```

```
## [1] 1.126217
```

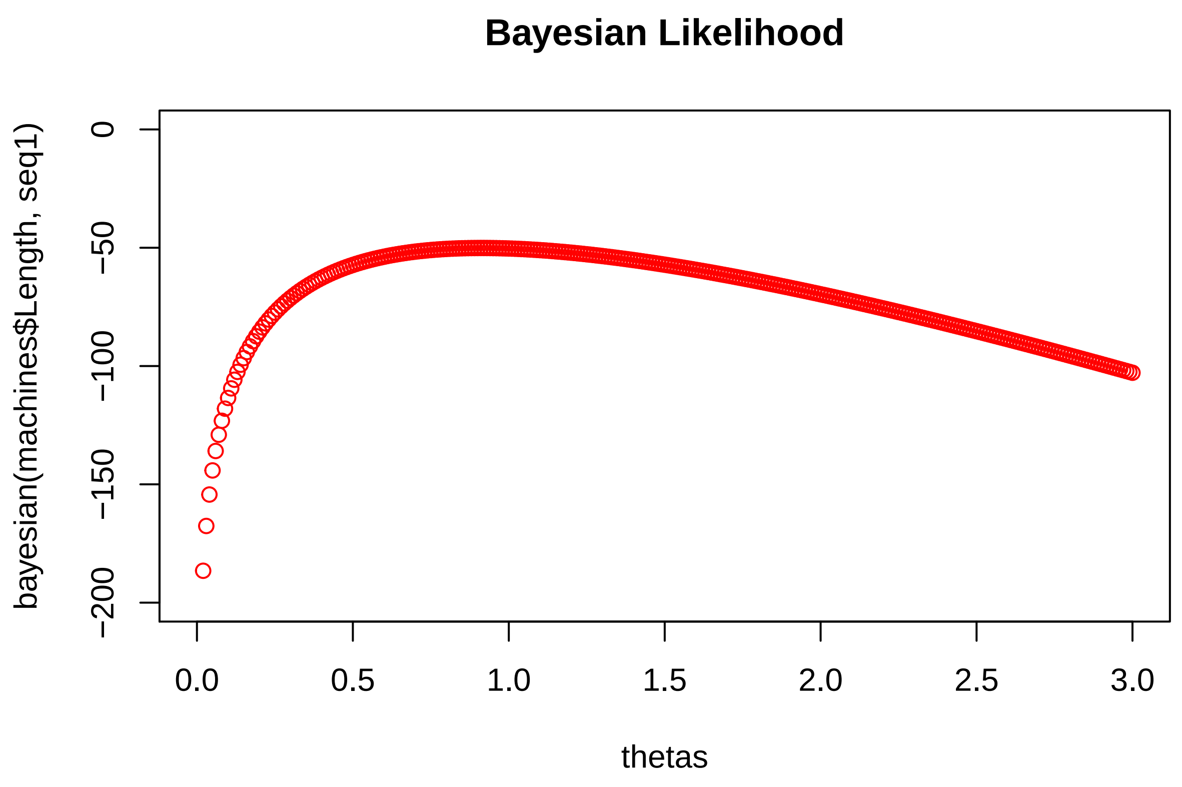
3) Using only the first six machines in our data set we are obviously using a lot less data, but we still get a similar curve even from only six data points. From the plot and running our max function we get a value around 1.78. This number is higher than previous results. We can probably assume that the theta we get from using all the data is more accurate as using only the first 6 machines misses many data points.



```
## [1] 1.785681
```

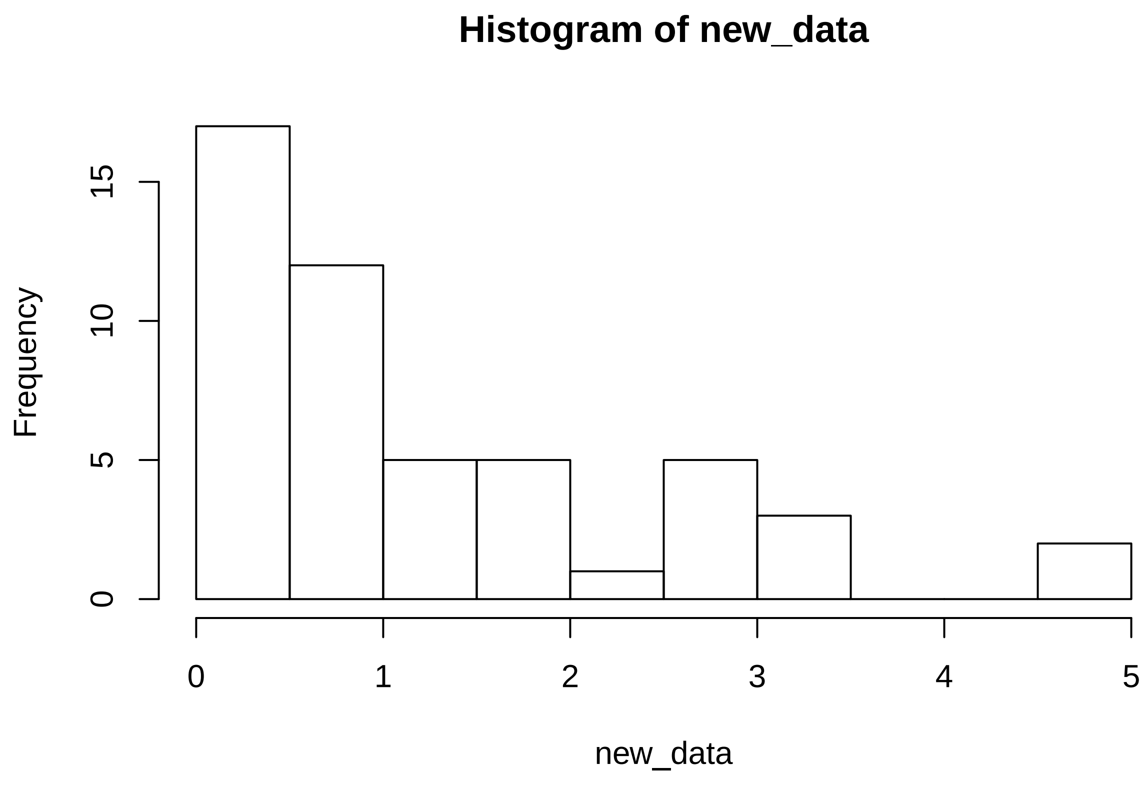
Red curve is the whole data set, blue is the first six machines.

4) Assume a bayesian model and using the probabalistic model given to us, we can calculate the likelihood as well with the given formulas. Here we can see a similar curve to the previous results, and from the plot we can see our maximum is around 1. Calculating manually the maximum likelihood we get a value of 0.91.



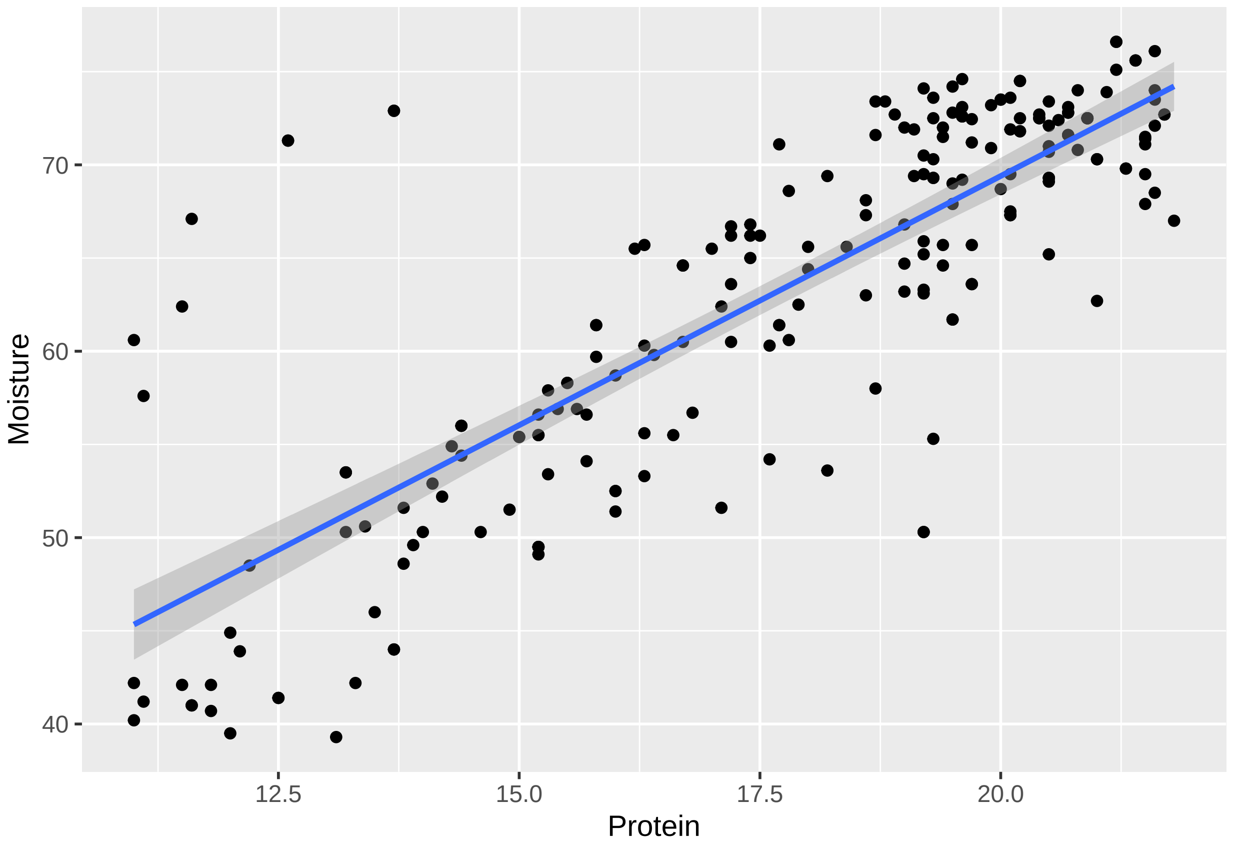
5) Using our optimal theta 0.91 that we observed in question 4, we can now generate 50 new data points based off this probability and distrubution. Using rexp we generate these new data points shown in the histogram, which we can see are also exponentially distributed.

```
new_data <- rexp(50, .91)
hist(new_data)
```



Assignment 4

1) Plotting Moisture vs Protein we can see the spread of the data. We can see it is well described by a linear model. Because we only have two features it is also clearly explained by a linear model, if we had higher dimensionality a linear model would not fit well.



2) Assuming the model we will use is linear regression, just adding degrees of polynomial this is valid model as it will simply add curvature to our 'linear' model. It is appropriate to use mean square error when evaluating these models because it is an easy calculation to do and they will give one value that can be compared to each other. Therefore we can find an optimal model with the correct degree polynomial by taking the lowest error.

```
m1<-lm(Moisture ~ Protein, data = train)
p1.train<-predict.lm(m1, train)
p1.test<-predict.lm(m1, test)
e1.train<-mse(p1.train, train$Moisture)
e1.test<-mse(p1.test, test$Moisture)


m2<-lm(Moisture ~ Protein + I(Protein^2), data = train)
p2.train<-predict(m2, train, type = "response")
p2.test<-predict(m2, test, type = "response")
e2.train<-mse(p2.train, train$Moisture)
e2.test<-mse(p2.test, test$Moisture)

m3<-lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3), data = train)
p3.train<-predict(m3, train, type = "response")
p3.test<-predict(m3, test, type = "response")
```

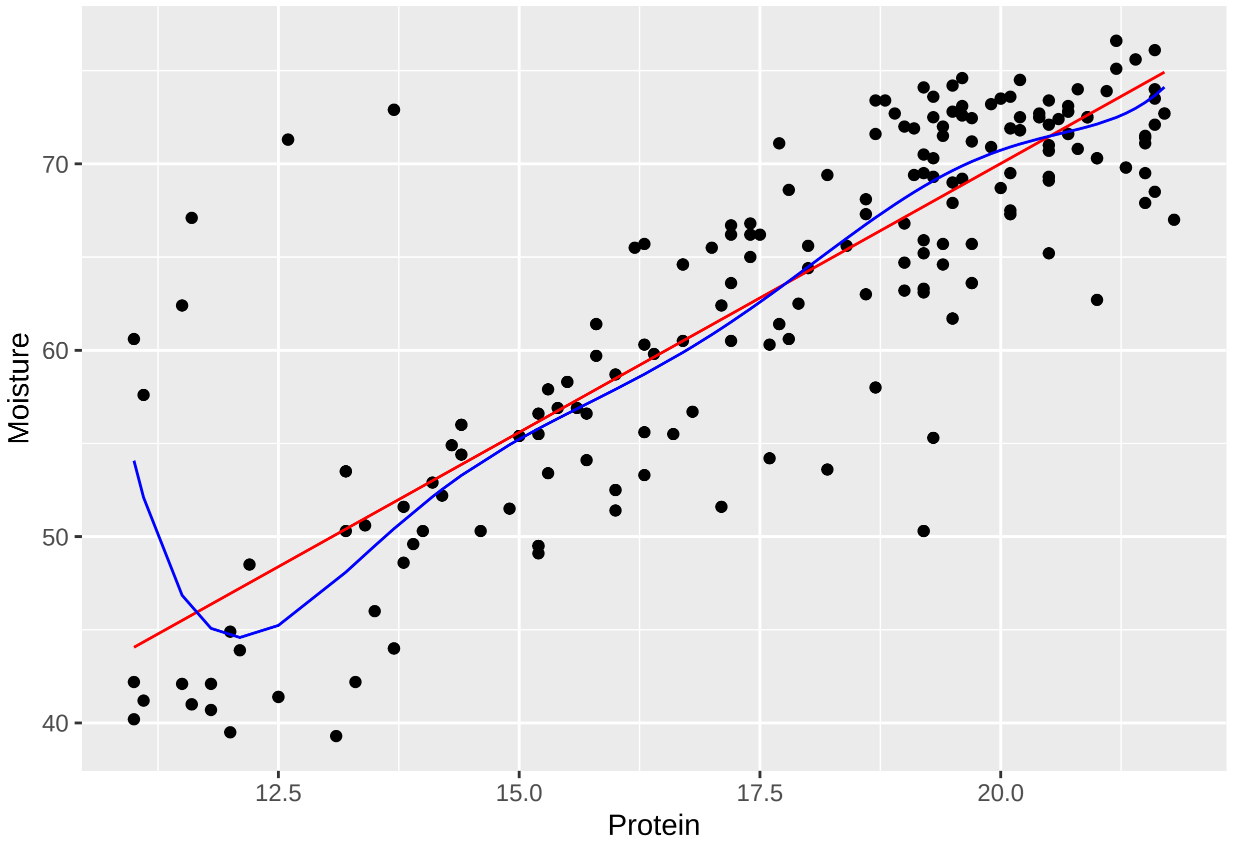
```r
e3.train<-mse(p3.train, train$Moisture)
e3.test<-mse(p3.test, test$Moisture)

m4<-lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3) + I(Protein^4), data = train)
p4.train<-predict(m4, train, type = "response")
p4.test<-predict(m4, test, type = "response")
e4.train<-mse(p4.train, train$Moisture)
e4.test<-mse(p4.test, test$Moisture)

m5<-lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3) + I(Protein^4) + I(Protein^5), data = train)
p5.train<-predict(m5, train, type = "response")
p5.test<-predict(m5, test, type = "response")
e5.train<-mse(p5.train, train$Moisture)
e5.test<-mse(p5.test, test$Moisture)

m6<-lm(Moisture ~ Protein + I(Protein^2) + I(Protein^3) + I(Protein^4) + I(Protein^5) + I(Protein^6), da
p6.train<-predict(m6, train, type = "response")
p6.test<-predict(m6, test, type = "response")
e6.train<-mse(p6.train, train$Moisture)
e6.test<-mse(p6.test, test$Moisture)

ggplot(data, aes(x = Protein, y = Moisture)) + geom_point() +
  geom_line(data = train, aes(x = Protein, y = p1.train), col = 'red') +
  geom_line(data = train, aes(x = Protein, y = p6.train), col = 'blue')
```

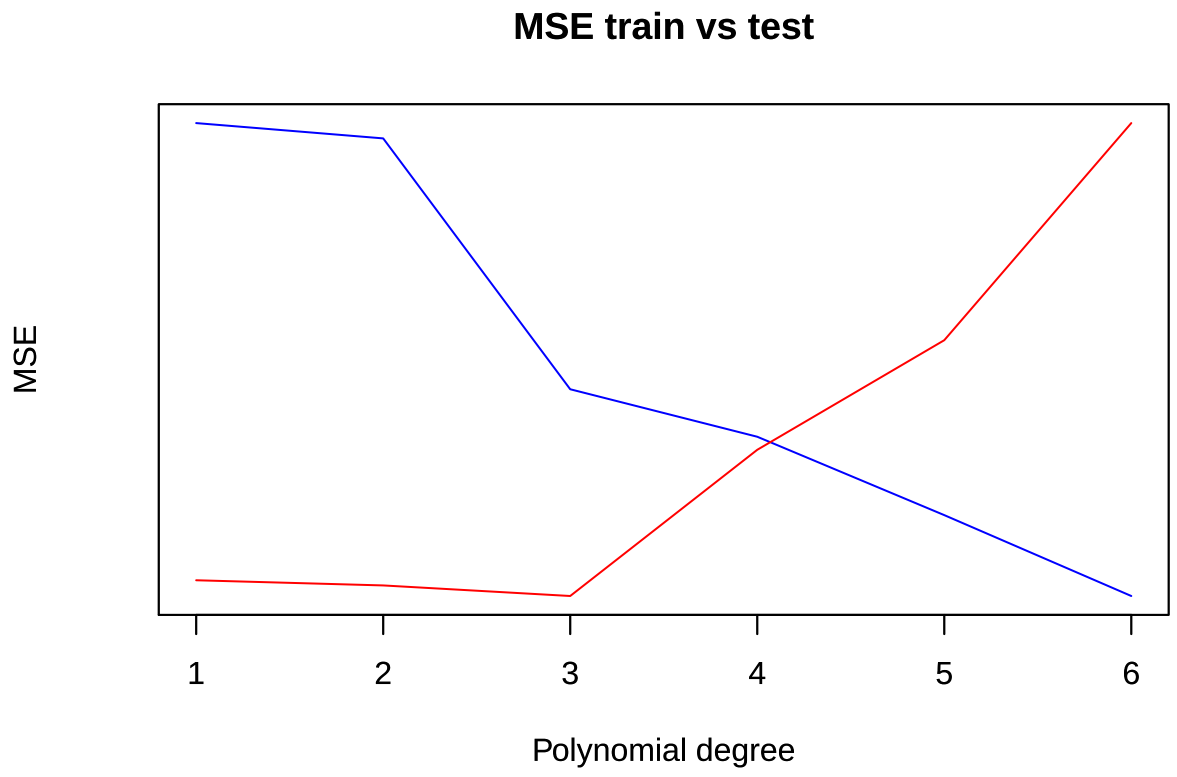


The plot above shows graphically what our linear model looks like compared with our degree six regression.

3) Plotting our MSE vs the degree of the polynomial we can see how we start to overfit the training data because our error goes down. This results in our testing data error to begin to increase as we overfit more. We see our optimal model is linear because the error is at minimum. With our linear model we have a high bias but a low variance. Where as when we increase our complexity we get low bias but higher variance.

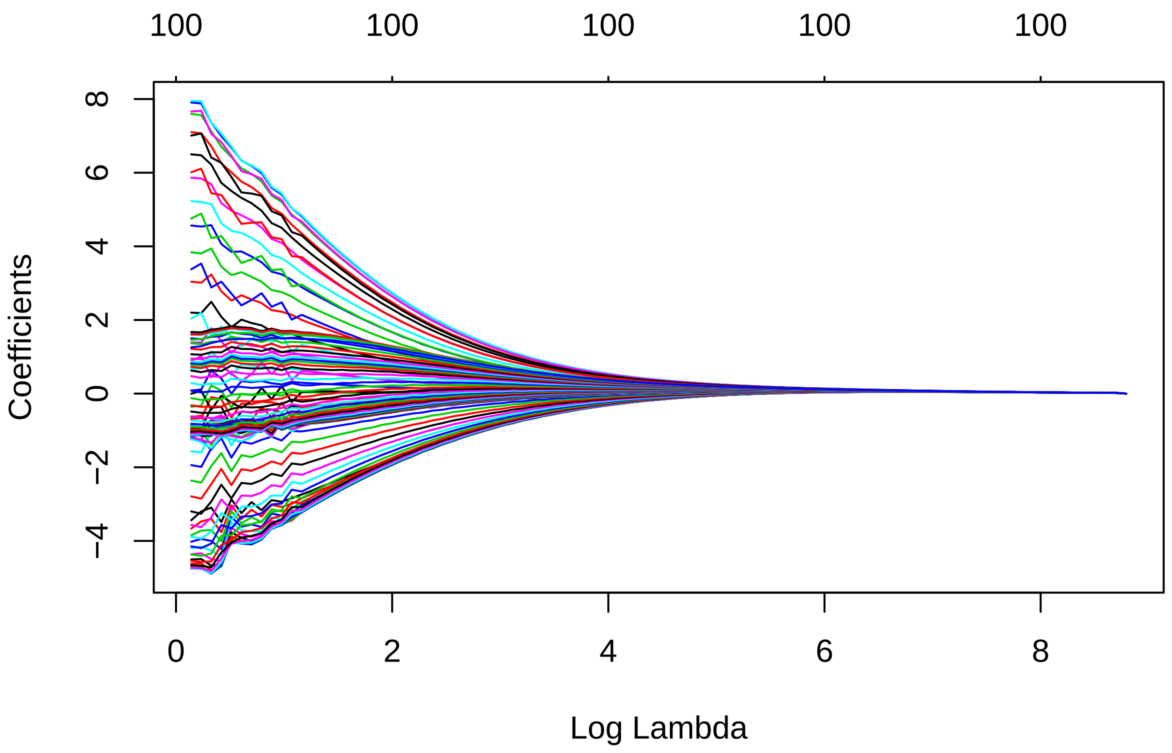
```r
#errors <- function(training, testing){
#  mse_train = numeric(6)
#  mse_test = numeric(6)
#
#  for(i in 1:6){
#    mse_train[i] = (MSE(I(polymodel(training, i, training)), train$Moisture))
#    mse_test[i] = (MSE(I(polymodel(training, i, testing)), test$Moisture))
#  }
#  msedf <- data.frame(train = mse_train, test = mse_test)
#  return(msedf)
#}

#msedf = errors(train,test)
#x=1:6
#ggplot(msedf, aes(x))+ geom_line(aes(y=train, x),  colour = "red") + geom_line(aes(y=test, x,), colour
```

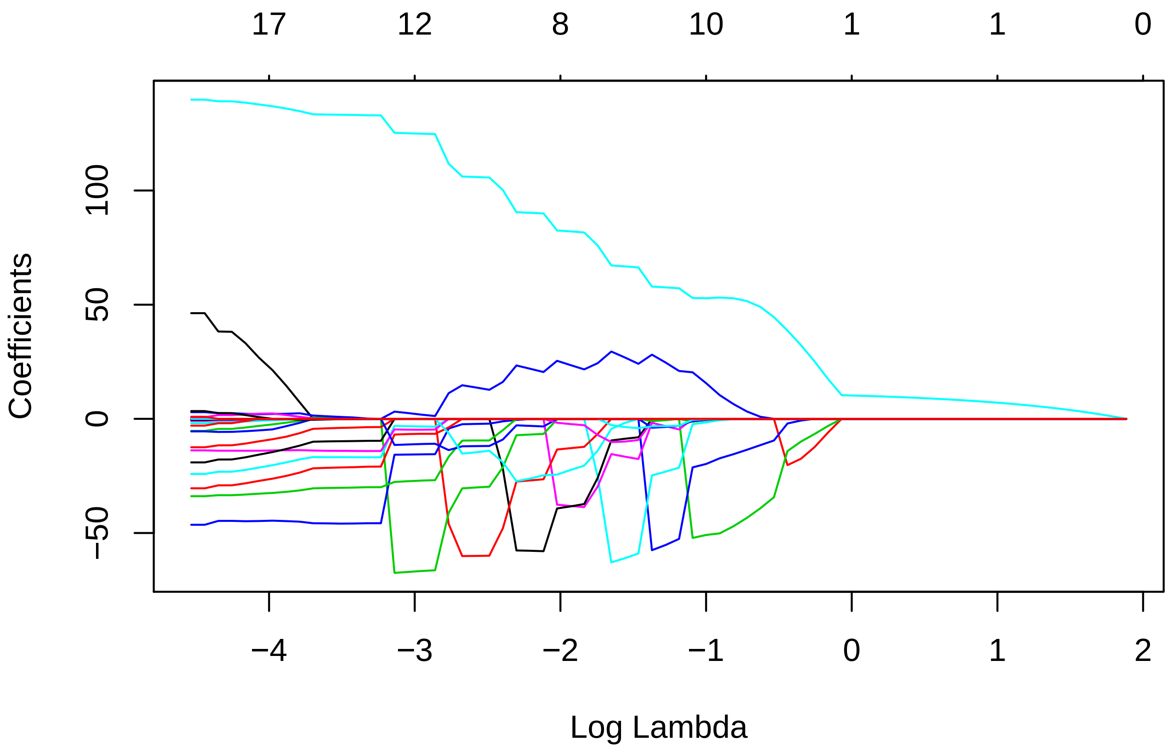


4) Using stepAIC to find the optimal set of input features. When checking the results from the anova table from our result we get a new model chosen with 63 variables selected out of the 100 initial.

5) Using glmnet function to run a ridge regression on our whole dataset we can plot the results showing the coefficients vs lambda. We can see that as lambda increases our coefficients converge to zero.

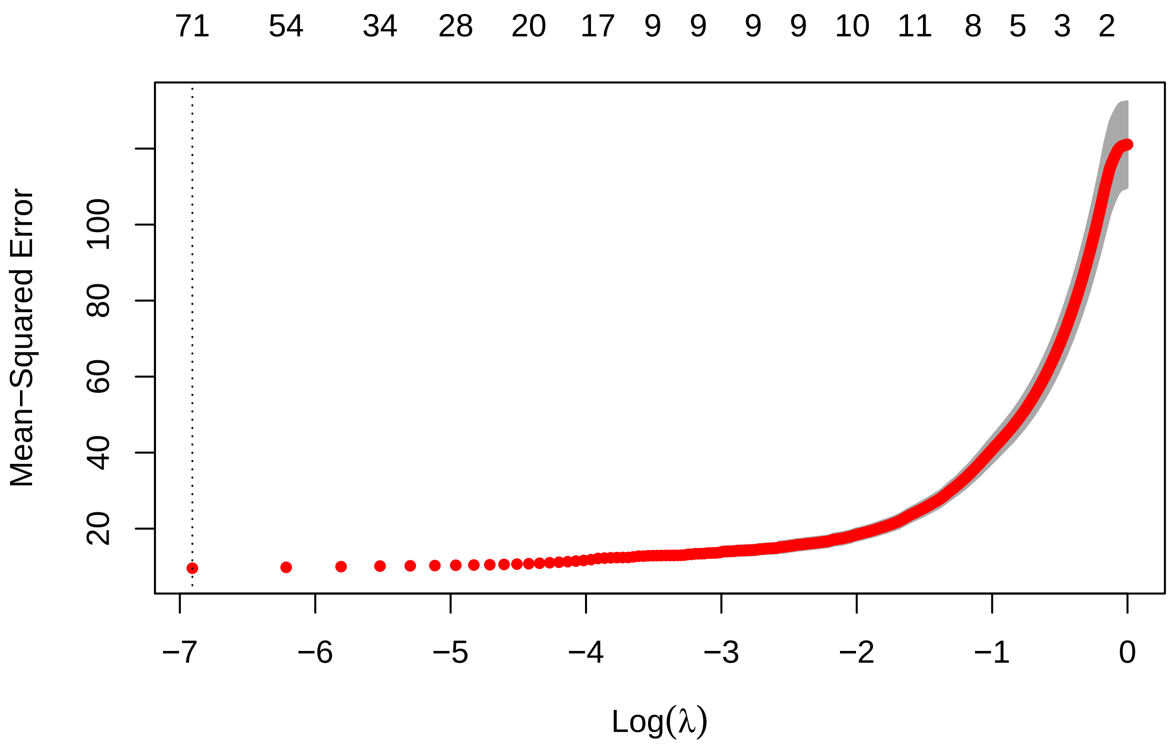


6) Repeating similar steps from question 5 we run our lasso reggresion and plot. Here we can see a clearly different graph from ridge, the coefficients for each variable do still converge to zero but much less smoothly because lasso has the ability to simply remove variables resulting in the 'jumps' in coefficients seen in the graph. Whereas ridge cannot remove them so they slowly reach zero.



7) Using cross validation we found the optimal lasso reggresion. Retrieving our minimum value of lambda we can see that it is zero. This means that according to this cross validation technique for lasso we

should choose a model with all the variables. When we look at our plot of the cross validation we can see that the dashed line actually chose a model with 71 features. The error rate seems to increase exponentially when we reach higher values of lambda. Compared to step 4 we actually removed a lot of variables in the stepAIC model, with this lasso model we are not removing any.


Mean−Squared Error
Log(λ)