Machine Learning: Block 1 Lab 1 (Group K6)

Yiran Wang, Tore Andersson and Shashi Nagarajan Iyer

11/11/2020

Statement of Contribution:

All assignments were completed independently by each team member and approaches/results were discussed within the group. Reports for assignments 1, 2 and 3 were produced primarily by Yiran, Shashi and Tore respectively.

1. Assignment 1 - Handwritten digit recognition with K-NN

1.1 Import data and divide it into training, validation and test sets

The first task is to import digit data and divide it into three sets for training, testing and validation respectively with 50%, 25% and 25% of the data. The code is as below:

```
digits <- read.csv("optdigits.csv", header = F)
colnames(digits)[ncol(digits)] <- "number"
digits$number <- as.factor(digits$number)
n <- dim(digits)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- digits[id,]
id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1, floor(n*0.25))
validation <- digits[id2,]
id3 <- setdiff(id1,id2)
test <- digits[id3,]
```

1.2 Train KNN model and estimate the result

This task consists of training a knn model with 30 nearest neighbors and estimate the result with confusion metrics and misclassification errors for both the training and test data. This should be done, using the kknn function, with the following code:

```
knn_train <- kknn(number ~., train, train, k = 30, kernel = "rectangular")
knn_test <- kknn(number ~., train, test, k = 30, kernel = "rectangular")
pred_train<- fitted(knn_train)
confusion_train <- table(train$number, pred_train)
pred_test <- fitted(knn_test)
confusion_test <- table(test$number, pred_test)
```

The confusion matrix for train data is:

```
print(confusion_train)
```

```
##    pred_train
##       0   1   2   3   4   5   6   7   8   9
##   0 202   0   0   0   0   0   0   0   0   0
##   1   0 179  11   0   0   0   0   1   1   3
##   2   0   1 190   0   0   0   0   1   0   0
##   3   0   0   0 185   0   1   0   1   0   1
##   4   1   3   0   0 159   0   0   7   1   4
##   5   0   0   0   1   0 171   0   1   0   8
##   6   0   2   0   0   0   0 190   0   0   0
##   7   0   3   0   0   0   0   0 178   1   0
##   8   0  10   0   2   0   0   2   0 188   2
##   9   1   3   0   5   2   0   0   3   3 183
```

The confusion matrix for test data is:

```
print(confusion_test)
```

```
##    pred_test
##       0   1   2   3   4   5   6   7   8   9
##   0  77   0   0   0   1   0   0   0   0   0
##   1   0  81   2   0   0   0   0   0   0   3
##   2   0   0  98   0   0   0   0   0   3   0
##   3   0   0   0 107   0   2   0   0   1   1
##   4   0   0   0   0  94   0   2   6   2   5
##   5   0   1   1   0   0  93   2   1   0   5
##   6   0   0   0   0   0   0  90   0   0   0
##   7   0   0   0   1   0   0   0 111   0   0
##   8   0   7   0   1   0   0   0   0  70   0
##   9   0   1   1   1   0   0   0   1   0  85
```

With these two matrix, we can simply have the accuracy rate based on correct numbers over total numbers for each digits:

```
print(diag(confusion_train)/rowSums(confusion_train))
```

```
##         0         1         2         3         4         5         6         7
## 1.0000000 0.9179487 0.9895833 0.9840426 0.9085714 0.9447514 0.9895833 0.9780220
##         8         9
## 0.9215686 0.9150000
```

```
print(diag(confusion_test)/rowSums(confusion_test))
```

```
##         0         1         2         3         4         5         6         7
## 0.9871795 0.9418605 0.9702970 0.9639640 0.8623853 0.9029126 1.0000000 0.9910714
##         8         9
## 0.8974359 0.9550562
```

For train data, the quality of predicting different digits is with a large range between 91% and 100%. The prediction for digit 0 is 100%, and following with 6, 2, 3 and 7 with 97% to 99% rate. While the lower ones are for digits 4, 9 and 1 with about 91% correct rate. This could indicate that even for train data itself, the model does not fit quite well.

For test data, the accuracy rates for prediction on digits do not have a huge difference than for train data. Some of them are higher than the train data prediction rate such as for 1, 6, 7 and 9. The highest one is 100% for digit 6. And other ones are below the train ones with a lowest rate for digit 4 being 86%. As we can see here, the rates for digits comparing with them for train data do not have an obvious decreasing trending, which means the model generalizes well.

We can also have the mis-classification rate calculated as below:

```
missclass <- function(X,X1) {
  n = length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
mismatch_rate_train <- missclass(train$number, pred_train)
mismatch_rate_test <- missclass(test$number, pred_test)
cat("The mismatch rate for train data is:", mismatch_rate_train,
    "\nThe mismatch rate for test data is:", mismatch_rate_test)
```

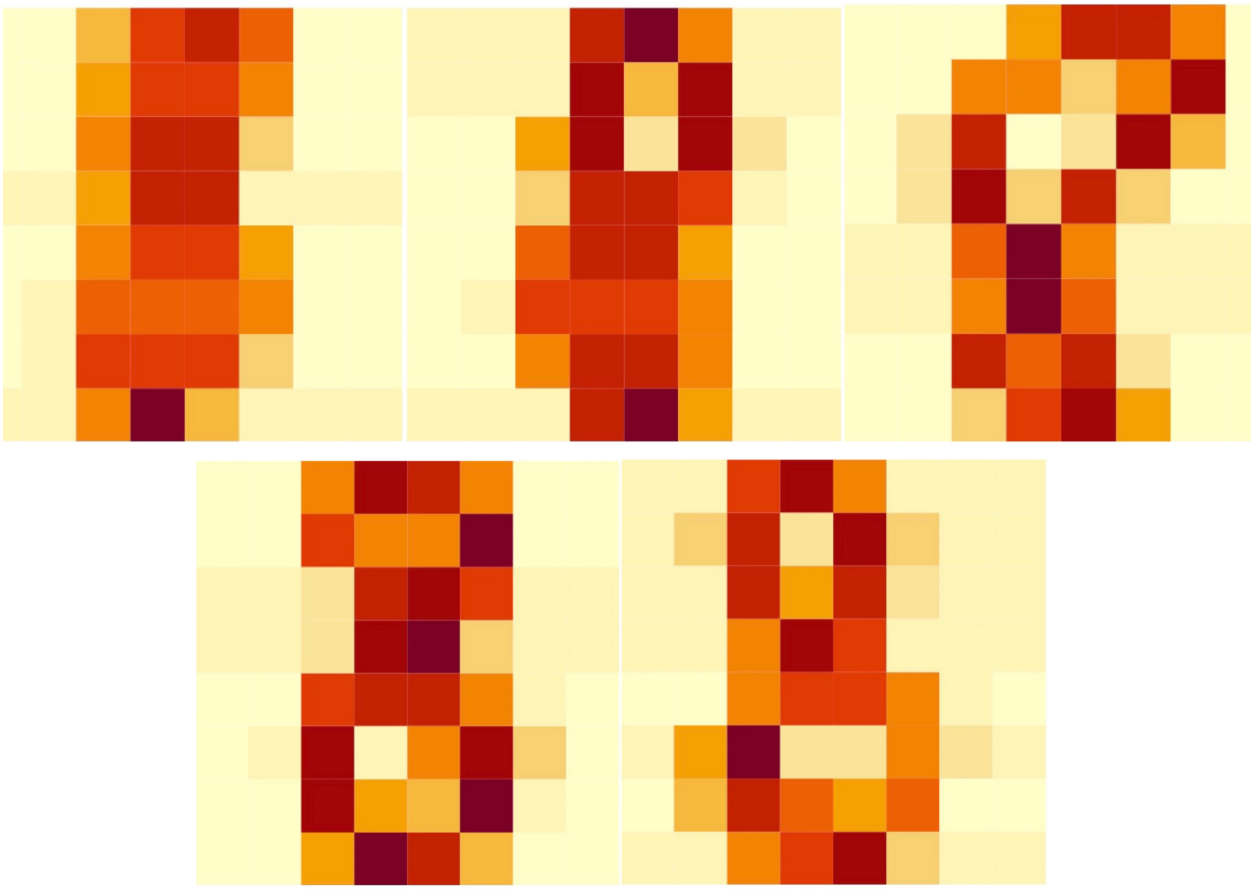
```
## The mismatch rate for train data is: 0.04500262
## The mismatch rate for test data is: 0.05329154
```

For the overall mismatch rate 0.045 for train data and 0.053 for test data, we can see that the rates for both of them are not low enough and the difference between them is not high, this can mean that the model is a little under fitting.

1.3 Identify and plot cases with low and high correct rates

This task is to find cases that are hard to classify and easy to classify and plot them with heatmap plots. This can be done with code as follows:

```
train_eight <- train[which(train$number==8),]
train_eight$prob <- knn_train$prob[which(train$number==8), "8"]
sortedResult <- sort(train_eight[, "prob"], index.return = T)
#sortedResult$x[1:3] #0.1000000 0.1333333 0.1666667
hard_ids <- sortedResult$ix[1:3] #50  43 136
#tail(sortedResult$x, n=c(2)) #1 1
easy_ids <- tail(sortedResult$ix, n=c(2)) #179 183
plotHeatmap <- function(index){
  heatmap(matrix(as.numeric(train_eight[index, 1:64]), 8, 8, byrow = T), Colv = NA, Rowv = NA)
}
# for (i in c(hard_ids, easy_ids)) {
#   plotHeatmap(i)
# }
```

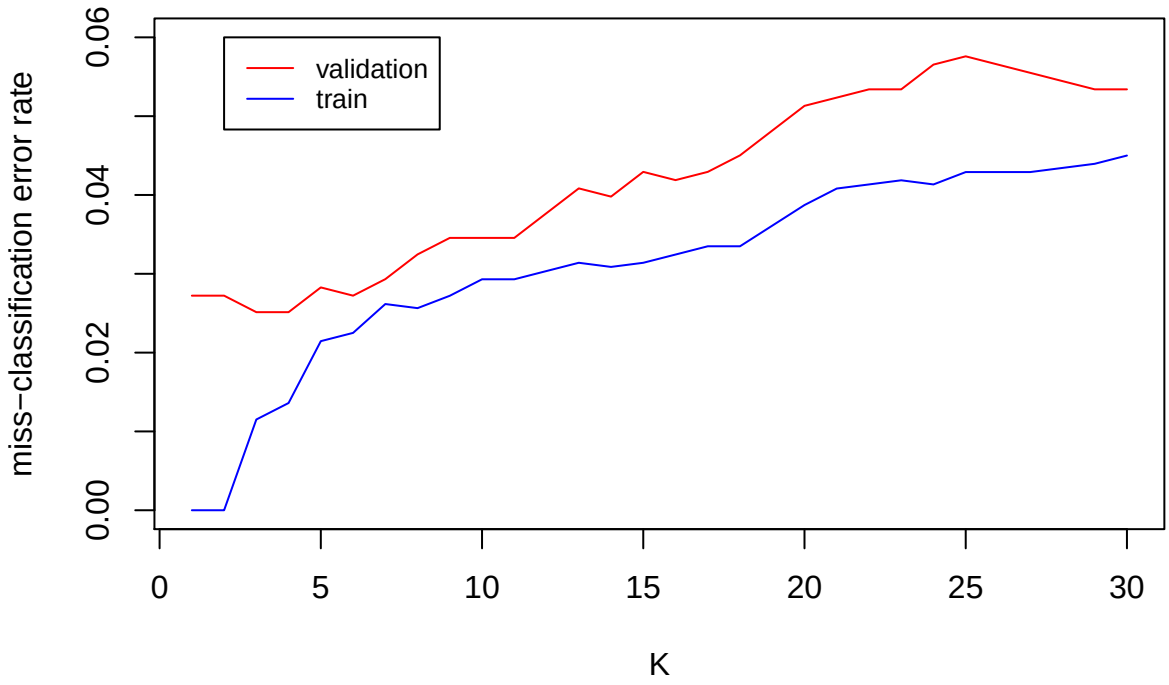


The cases that are hard to see are chosen with lowest correct rates being 0.1000000, 0.1333333 and 0.1666667. While for the easy to identify ones, the rates are both 100%. As the heatmap shows, the cases that has lowest accuracy rates(the top three showing on the image) are hard to recognized as digits 8 when visualizing them as well. All of them are either too blur or the shape is not clear enough to identify as digit 8. While the 100% accurate cases(the bottom two on the image) are very easy to see as 8.

1.4 Use different knn models to fit the train data and analyze with miss-classification rate

This is about fitting the train data with K=1..30 knn models and plot the mis-classification rates for train and validation data. Then observe the plot in different perspective. The code shows as following:

```
e_t <- 0
e_v <- 0
for (k in 1:30) {
  kt <- kknn(number ~., train, train, k = k, kernel = "rectangular")
  e_t[k] <- missclass(train$number, fitted(kt))
  kv <- kknn(number ~., train, validation, k = k, kernel = "rectangular")
  e_v[k] <- missclass(validation$number, fitted(kv))
}
plot(1:30, e_v, ylim = c(0,0.06), type = "l", col = "red", xlab = "K", ylab = "miss-classification erro
lines(1:30, e_t, ylim = c(0,0.06), col = "blue")
legend(2, 0.06, legend=c("validation", "train"),
       col=c("red", "blue"), lty=1:1, cex=0.8)
```



As k increases, the complexity of the model for both data sets also increases overall. As we can see in the plot, the error rate for the validation data decreases first to 0.025 when K is 3 and then goes up quickly as K increases with a final slight drop till 0.05. While the error rate for the train data starts with 0 when k is 0, and then goes up gradually as K goes up.

The optimal K is a little difficult to detect right away since the differences for different K values are very small, especially when K is about 3 to 7. But I would say maybe 3 is the optimal one based on only the rate we plot here. As at that point, the value for validation rate is the lowest and for train data is also relatively low.

From the perspective of bias-variance trade-off, low bias is not always perfect as it can lead to a higher variance(when error of train data is 0, the error of validation data is not the lowest). When bias is a little bit higher, the variance can be a bit lower as it can generalize new data better. But after certain point, when bias gets too high, the error rate for both train and validation data will be high. In this case, it would be not a suitable model as it may be under-fitting.

With the optimal K value selected based on above analysis, the model for test data can be shown with following code:

```
k_test_optik <- kknn(number ~., train, test, k = 3, kernel = "rectangular")
cat("Test mis-classification rate when k=3 is:", missclass(test$number, fitted(k_test_optik)))
```

```
## Test mis-classification rate when k=3 is: 0.02403344
```

```
cat("Train mis-classification rate when k=3 is:", e_t[3])
```

```
## Train mis-classification rate when k=3 is: 0.0115123
```

```
cat("Validation mis-classification rate when k=3 is:", e_v[3])
```

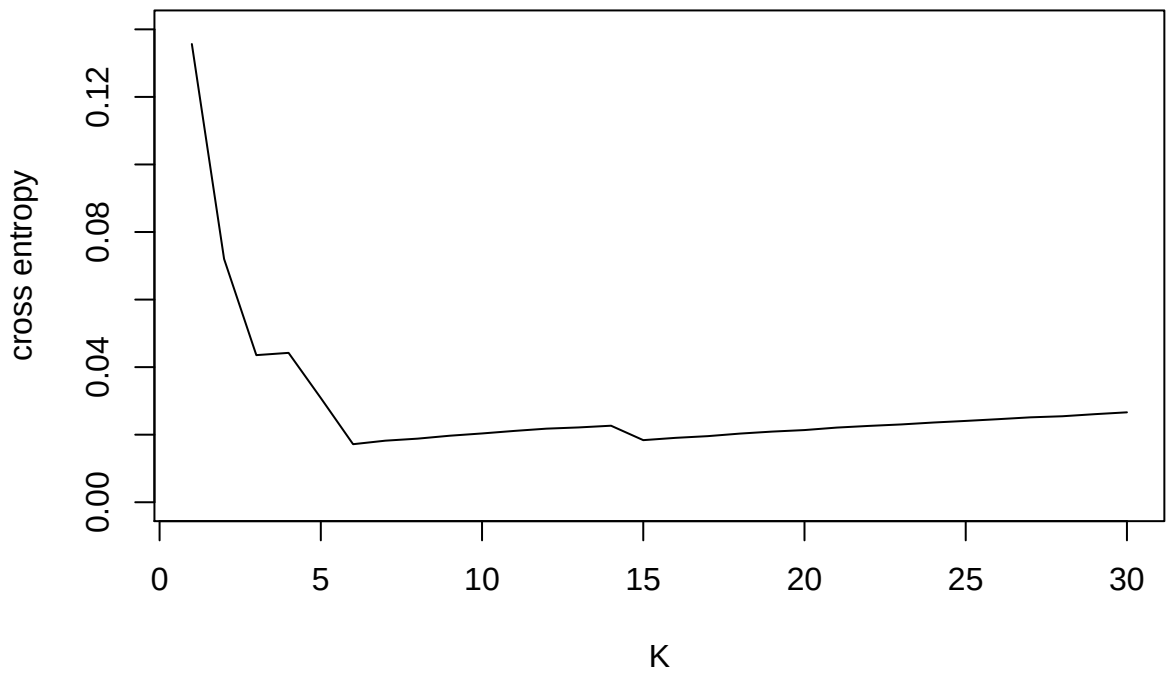
```
## Validation mis-classification rate when k=3 is: 0.02513089
```

The test error being 2.40% with K=3 is about the same as the error rate of validation data and a little higher (about 1.25%) higher than train data. I would say the quality of the model with K=3 is acceptable because with a similar relatively high fitting rate to both validation and test data(not overfitting) and well fitted for train data(not underfitting).

1.5 Use different knn models to fit the train data and analyze with empirical risk

This task is also about fitting train data with different k values. But estimate the result with empirical risk, in this case, cross entropy value. This is done with the following code:

```
r_v <- 0
for (k in 1:30) {
  kv <- kknn(number ~., train, validation, k = k, kernel = "rectangular")
  s <- 0
  for (i in 0:9) {
    s <- s + sum(log2(kv$prob[which(validation$number==i), toString(i)] + 1e-15)/nrow(kv$prob))
  }
  r_v[k] <- -s/10
}
plot(1:30, r_v, ylim = c(0,0.14), type = "l", xlab = "K", ylab = "cross entropy")
```



The optimized K value is 6, because at that point, the risk value is the lowest with a value being 0.0172. The reason cross entropy is a better approach of estimating optimal model here is because it uses the probabilities of each class for every observation. While the mis-classification rate only count the final result class of each observation. Thus, cross-entropy can detect which model is better even though their mis-classification rates are the same.

Question 2

Bayesian ridge regression model for the data (without the intercept parameter) is:

$$\mathbf{y}|\mathbb{X}, w_0, \mathbf{w} \sim N(w_0 + \mathbb{X}\mathbf{w}, \sigma^2\mathbb{I}),$$

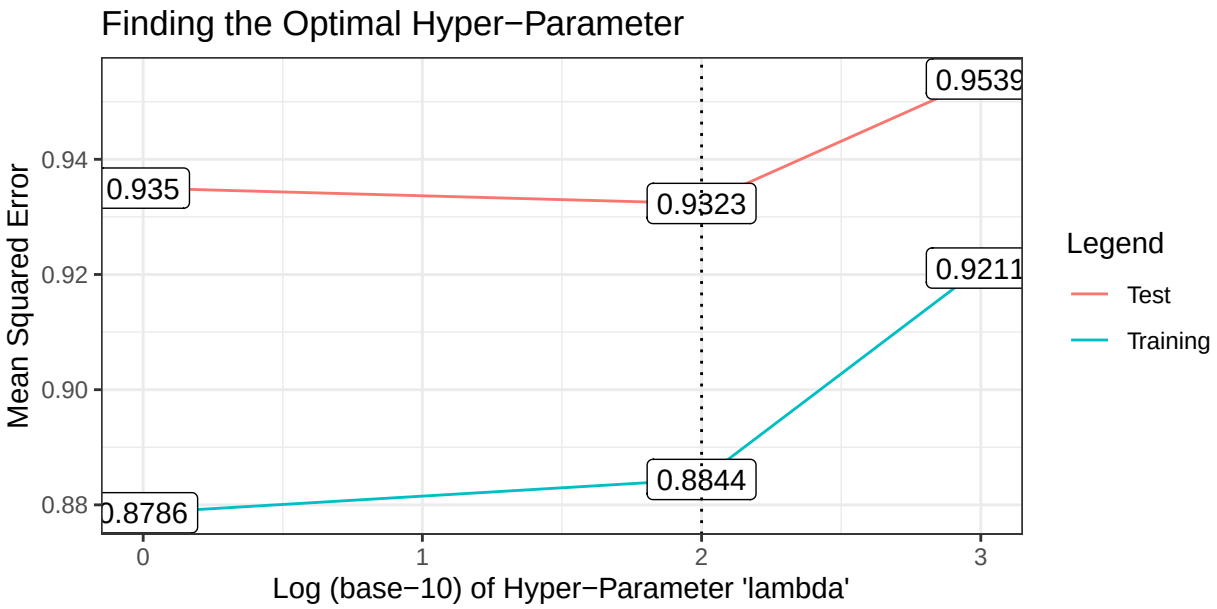
where:

- $\mathbf{y}$ is the vector of Parkinson's disease symptom scores, "motor UPDRS", the dependent variable
- $\mathbb{X}$ is the design matrix containing the observed values of all the independent variables corresponding to $\mathbf{y}$ (voice characteristics in this case)
- $\boldsymbol{w}$ is the vector of bayesian ridge regression parameters
- σ is a scalar such that $\sigma^2 \cdot \mathbb{I}$ is the covariance matrix of the linear regression error terms

The Bayesian prior for $\boldsymbol{w}$ is:

$$\mathbf{w} \sim N(0, \frac{\sigma^2}{\lambda}\mathbb{I})$$

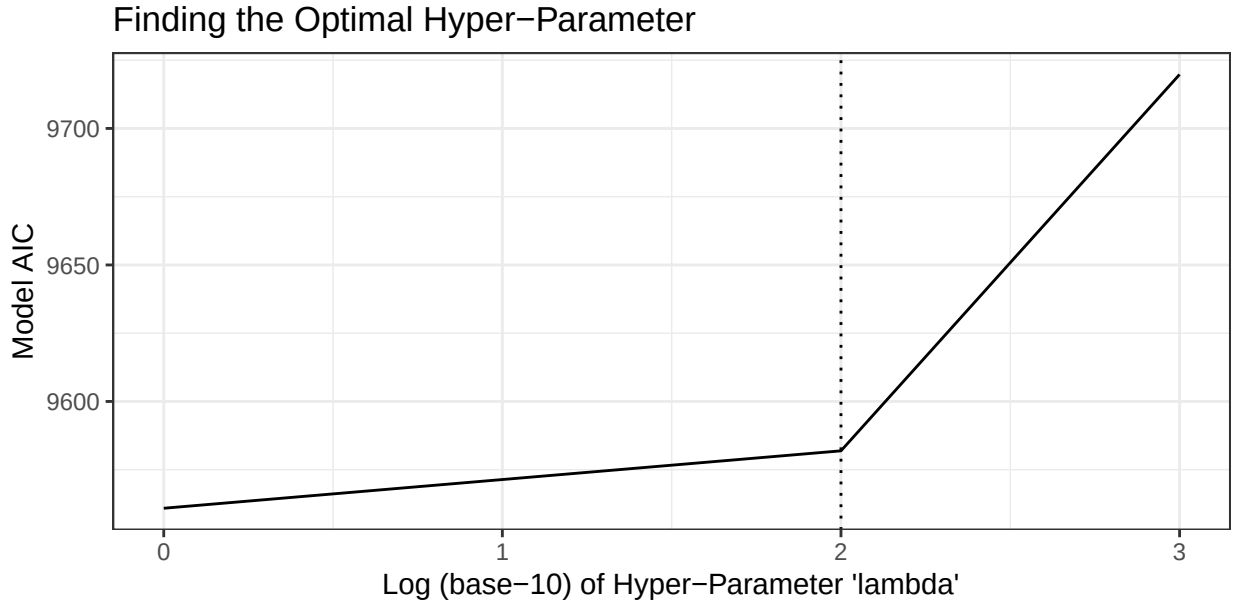
Fitting the above mentioned ridge regression model on the training dataset, with $\lambda = 1, 100 and 1,000$, we see the training and test MSE values as shown in the plot below.



We can see that the optimal value of λ within the set $\{1, 100, 1000\}$ is 100. At $\lambda = 1$, there is overfitting, evidenced by low training error and large test error; at $\lambda = 1000$, on the other hand, there is underfitting, evidenced by high training and test error.

MSE a more appropriate measure here than other empirical risk functions such as MAE, because in regression problems, unlike measures like MAE, MSE can also serve as an estimator of the variance of the error terms. Of course empirical risk functions such as mis-classification rate or cross-entropy do not apply in regression problems such as this, making MSE more appropriate than such measures.

In terms of AIC as well, the optimal value of λ within the set $\{1, 100, 1000\}$ is 100. One theoretical advantage of using AIC is that unlike the holdout method, it can be used for model selection in unsupervised learning models, where losses, unlike MSE, will not be a function of 'labels' and where one cannot ascertain under/overfitting based on test MSEs.



Question 3

The linear regression model for the data is:

$$\mathbf{y} \sim \mathcal{N}(\mathbb{X}\boldsymbol{\beta}, \sigma^2\mathbb{I}),$$

where:

- $\mathbf{y}$ is the vector "Fat" (the dependent variable)
- $\mathbb{X}$ is the design matrix containing the observed values of all the independent variables corresponding to $\mathbf{y}$ (absorbance characteristics, in thtis case), prefixed with a column of 1s for the intercept
- $\boldsymbol{\beta}$ is the vector true linear regression parameters
- σ is a scalar such that $\sigma^2 \cdot \mathbb{I}$ is the covariance matrix of the linear regression error terms

Fitting the above mentioned model on the given dataset we see low training error and high test error; errors were measured using the mean-squared error loss function.

```
## Training MSE is: 0.007108636
```

```
## Test MSE is:  635.6713
```

These statistics indicate overfitting. On average training predictions are very close to actual training labels, but test predictions are relatively very different from test labels. As a result we cannot use satisfactorily use this model to predict fat content in any new meat sample.

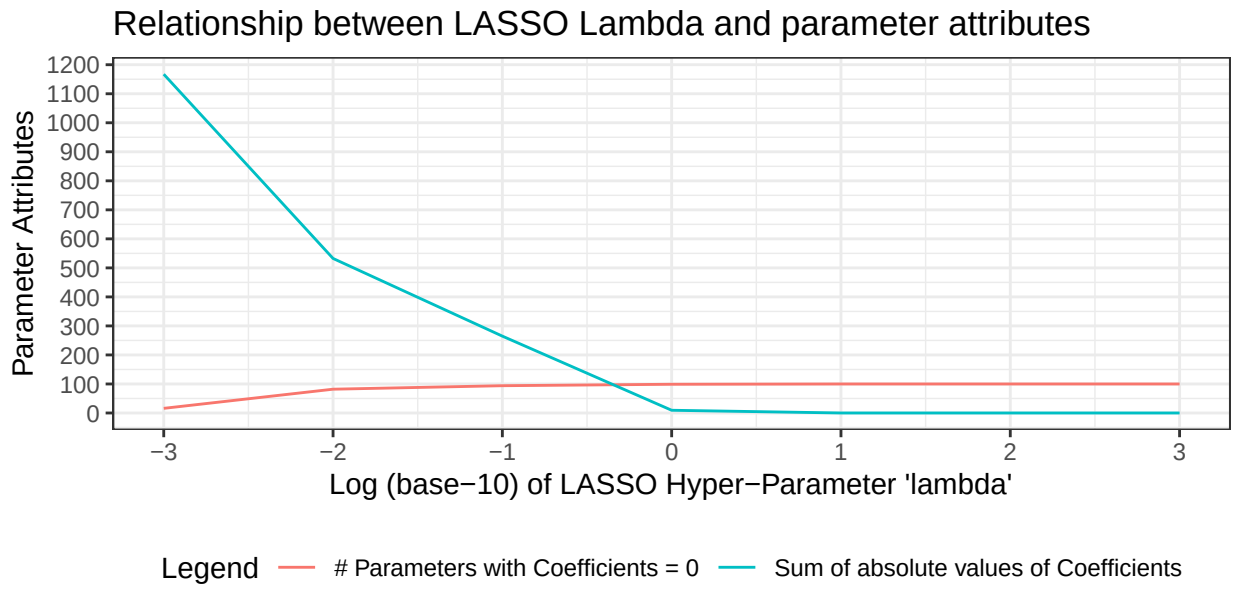
If we turn to the LASSO regression method, however, we would be selected $\boldsymbol{\beta}$ such that the following loss function is minimised:

$$\sum_{i=1}^{N} (y_i - \mathbb{X}_i\boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_i|$$

where, additionally:

- N is the total number of observations in the dataset
- i is an index the observations in the dataset, and runs from 1, 2, 3, ... N
- λ is the LASSO hyper-parameter

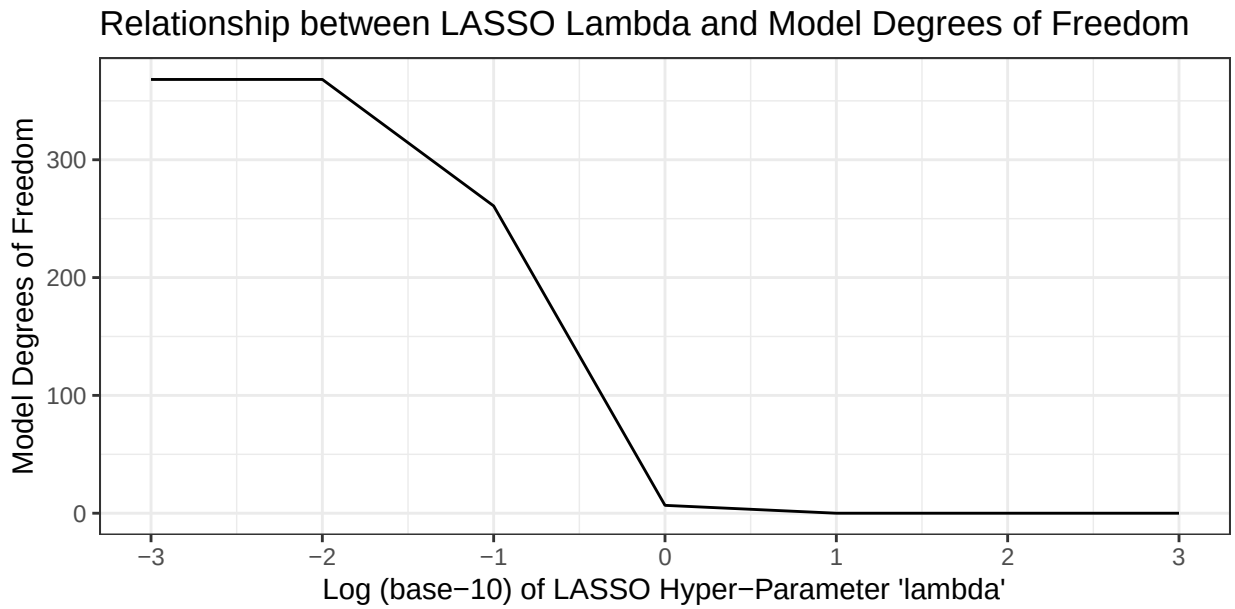
With $\lambda \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, LASSO regression estimates parameters with attributes as shown in the plot below:



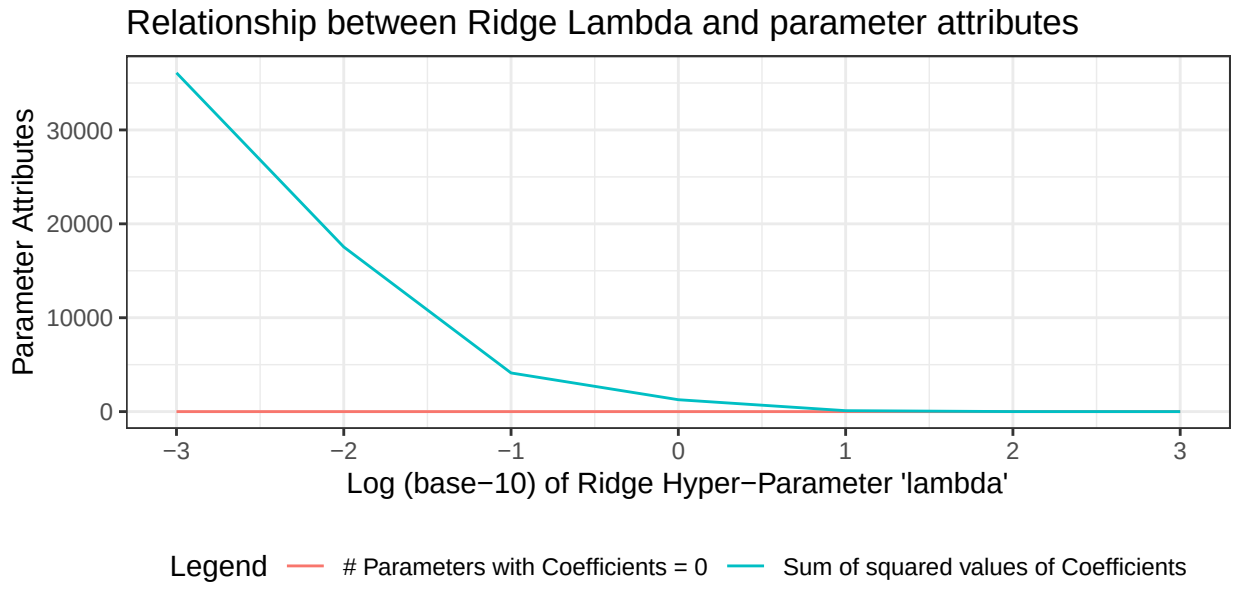
What we see above, as expected in theory, is that increases in λ result in increased penalty on the absolute values of parameter estimates. When $\lambda >= 1$, LASSO reduces to estimates of all 100 parameters to zero and brings down the value of the sum of absolute values of all parameters estimates to zero.

The most appropriate value for lambda in a case where there are only 3 independent variables can, as is common, be estimated through cross-validation or holdout sample methods, but one should not expect the estimates to be very large.

Plotting model degrees of freedom versus λ on log (base 10) scale, we can see the former decrease as λ increases; with very large values of lambda (such as $\lambda >= 100$ in this case), degrees of freedom will drop to zero, the lowest possible value for this measure.

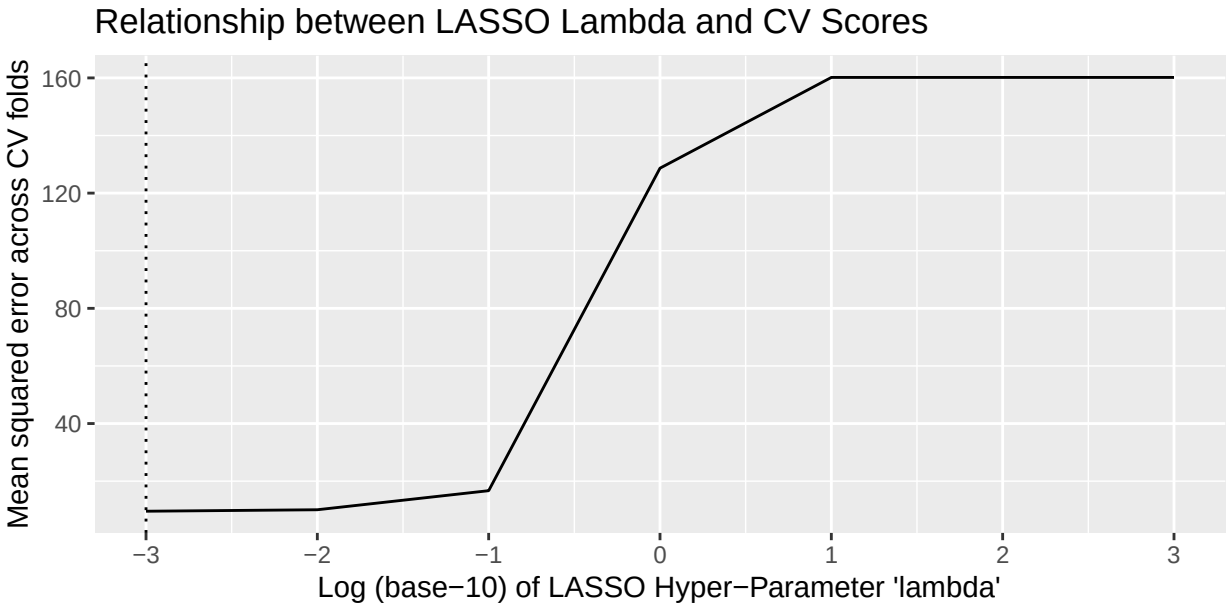


With $\lambda \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, ridge regression, on the other hand, estimates parameters with attributes as shown in the plot below:

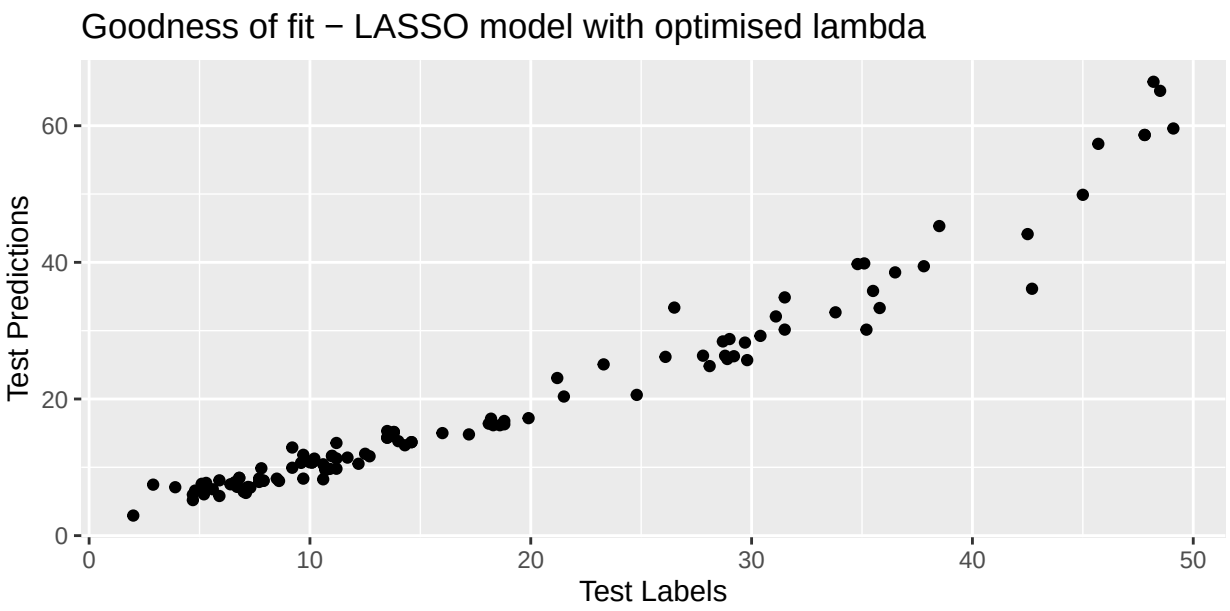


Upon comparison, it becomes clear that ridge regression, unlike LASSO, does not bring down any parameter estimates to zero.

Upon running 3-fold cross-validation on the training dataset, we can see that CV score increases as λ increases.

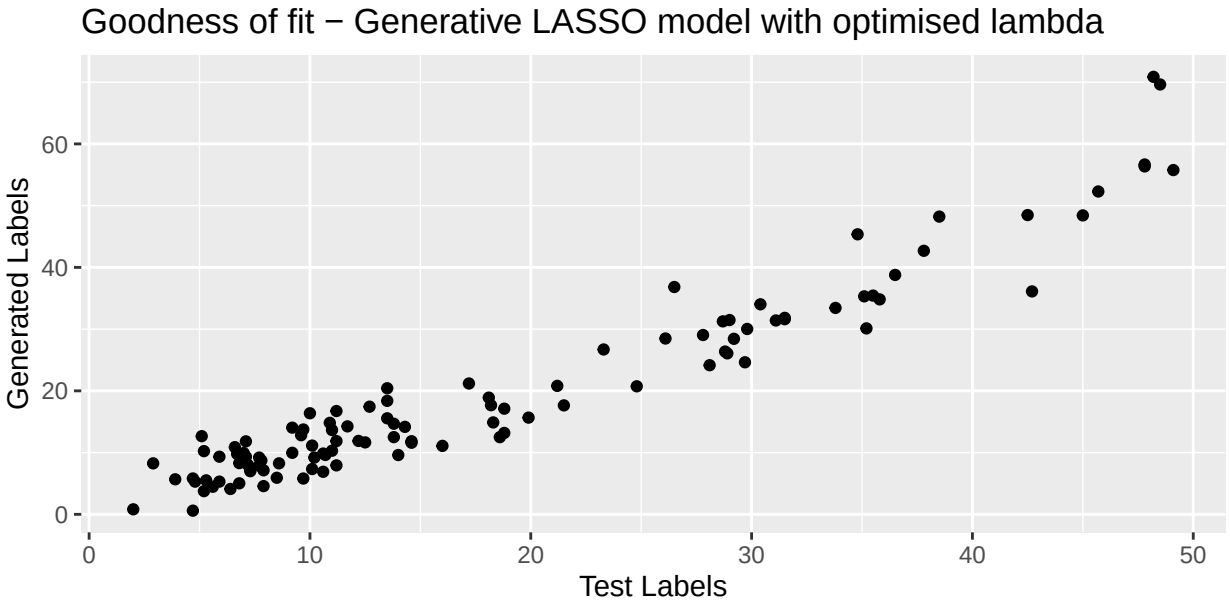



```
## Optimal Lambda is: 0.001

## Number of variables chosen in the model: 84

## The p-value for the test of null hypothesis that optimal lambda works similar to log(lambda) = -2 vs

## We, therefore, cannot conclude confidently that the optimal lambda works significantly better than lo
```



The LASSO regression model with optimised $\lambda = 10^-3$ produces good test predictions which are concentrated around the $y = x$ line in the plot of Predictions vs. Actual Labels.

The same, however, is not a good generative model, because of the assumption that the training dataset is deterministic. This can also be seen in the plot below, where points generated from the $\mathcal{N}(\mathbb{X}\hat{\beta}_{LASSO}, \hat{\sigma}^2_{MLE})$ are quite spread out around the $y = x$ line (relative to the spread seen in the non-generative, deterministic model).



Part done by Tore so far

So 3.6 and 3.7, are not finished need to do the hypothisis test, and was abit unsure as what to plot thought it to be abit unclear for 3.6.

3.1

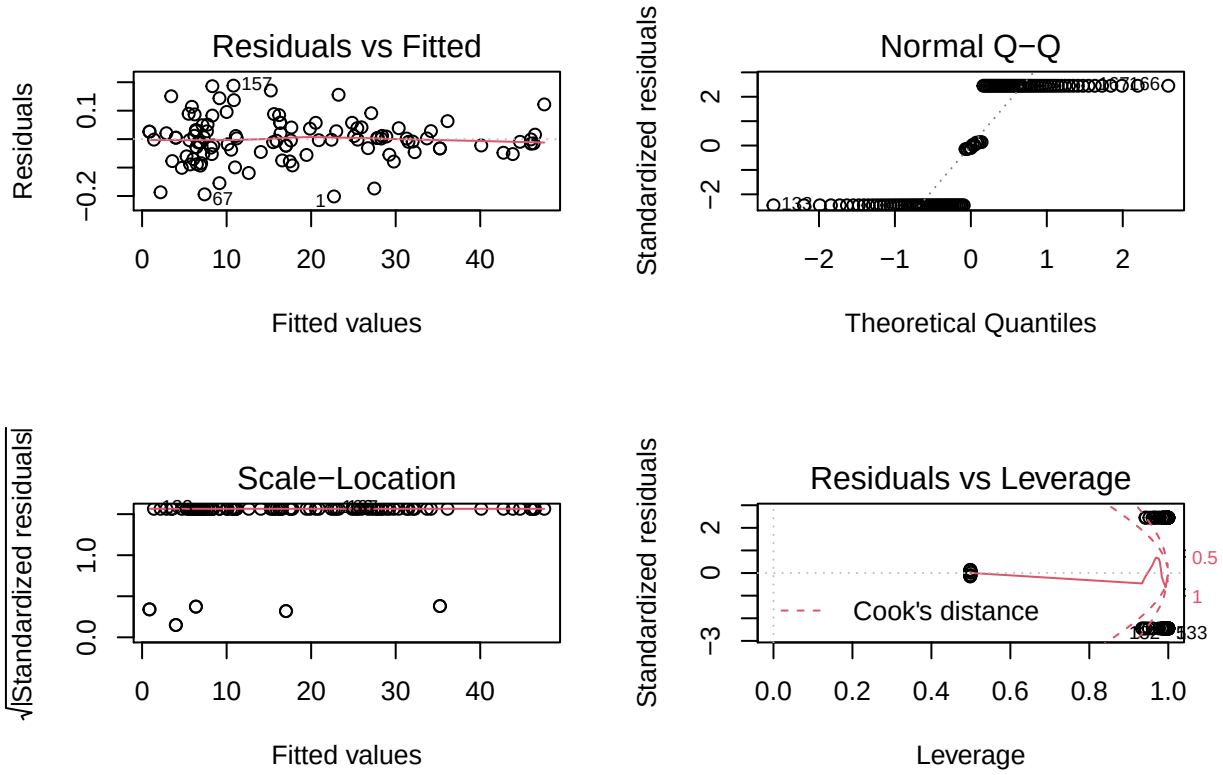
```
##
## Call:
## lm(formula = Fat ~ ., data = train_fat)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.201500 -0.041315 -0.001041  0.037636  0.187860
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.815e+01  5.488e+00  -3.306  0.01628 *
## Channel1     2.653e+04  1.126e+04   2.357  0.05649 .
## Channel2    -5.871e+04  3.493e+04  -1.681  0.14385
## Channel3     1.154e+05  7.373e+04   1.565  0.16852
## Channel4    -2.432e+05  1.175e+05  -2.070  0.08387 .
## Channel5     3.026e+05  1.193e+05   2.536  0.04430 *
## Channel6    -2.365e+05  8.160e+04  -2.898  0.02741 *
## Channel7     1.090e+05  3.169e+04   3.440  0.01380 *
## Channel8    -6.054e+04  1.508e+04  -4.015  0.00700 **
```

```
## Channel9     7.871e+04  2.160e+04   3.643  0.01079 *
## Channel10   -1.730e+04  1.640e+04  -1.055  0.33215
## Channel11    9.562e+04  3.529e+04   2.710  0.03512 *
## Channel12   -2.114e+05  6.198e+04  -3.410  0.01431 *
## Channel13    9.725e+04  4.424e+04   2.198  0.07026 .
## Channel14    5.296e+04  4.666e+04   1.135  0.29968
## Channel15   -7.855e+04  5.245e+04  -1.498  0.18491
## Channel16   -8.209e+03  1.893e+04  -0.434  0.67969
## Channel17    3.769e+04  1.987e+04   1.897  0.10666
## Channel18    3.306e+04  7.934e+03   4.167  0.00590 **
## Channel19   -8.405e+04  1.929e+04  -4.358  0.00478 **
## Channel20    1.510e+05  3.361e+04   4.492  0.00414 **
## Channel21   -2.069e+05  4.256e+04  -4.862  0.00282 **
## Channel22    1.348e+05  3.824e+04   3.526  0.01243 *
## Channel23   -4.094e+04  3.546e+04  -1.154  0.29222
## Channel24    2.023e+04  2.761e+04   0.733  0.49134
## Channel25    3.269e+03  1.071e+04   0.305  0.77045
## Channel26   -1.297e+04  7.636e+03  -1.699  0.14028
## Channel27    4.131e+03  1.422e+04   0.291  0.78120
## Channel28   -4.548e+03  2.988e+04  -0.152  0.88402
## Channel29    1.089e+04  1.768e+04   0.616  0.56072
## Channel30   -7.985e+04  2.653e+04  -3.010  0.02371 *
## Channel31    1.756e+05  5.279e+04   3.326  0.01589 *
## Channel32   -1.107e+05  2.904e+04  -3.813  0.00883 **
## Channel33   -6.525e+04  5.407e+04  -1.207  0.27294
## Channel34    1.007e+05  6.589e+04   1.528  0.17738
## Channel35   -2.841e+03  1.214e+04  -0.234  0.82266
## Channel36   -2.268e+04  2.295e+04  -0.988  0.36127
## Channel37   -4.479e+04  1.292e+04  -3.468  0.01334 *
## Channel38    3.209e+04  1.843e+04   1.742  0.13221
## Channel39    1.992e+04  2.067e+04   0.964  0.37246
## Channel40   -9.833e+03  2.431e+04  -0.404  0.69988
## Channel41    1.659e+04  3.648e+04   0.455  0.66531
## Channel42   -1.829e+04  3.528e+04  -0.519  0.62260
## Channel43   -2.423e+04  2.427e+04  -0.998  0.35669
## Channel44    3.246e+04  2.013e+04   1.613  0.15793
## Channel45   -8.089e+03  4.023e+04  -0.201  0.84728
## Channel46    7.065e+03  2.810e+04   0.251  0.80990
## Channel47   -4.062e+04  1.007e+04  -4.034  0.00685 **
## Channel48    9.080e+04  2.618e+04   3.469  0.01332 *
## Channel49   -6.647e+04  2.372e+04  -2.803  0.03105 *
## Channel50   -4.196e+04  2.856e+04  -1.469  0.19213
## Channel51    1.097e+05  5.572e+04   1.968  0.09661 .
## Channel52   -1.148e+05  6.376e+04  -1.800  0.12196
## Channel53    9.525e+04  7.450e+04   1.278  0.24830
## Channel54   -4.534e+04  7.363e+04  -0.616  0.56067
## Channel55   -1.535e+03  4.933e+04  -0.031  0.97618
## Channel56   -2.377e+03  2.109e+04  -0.113  0.91394
## Channel57    3.174e+04  1.005e+04   3.158  0.01961 *
## Channel58    2.221e+03  1.048e+04   0.212  0.83915
## Channel59   -8.504e+04  2.574e+04  -3.304  0.01634 *
## Channel60    6.382e+04  1.607e+04   3.972  0.00735 **
## Channel61    2.151e+04  1.234e+04   1.742  0.13211
## Channel62   -2.859e+04  1.065e+04  -2.685  0.03631 *
```

```
## Channel63    1.796e+04  9.187e+03   1.955  0.09838 .
## Channel64    5.759e+04  3.526e+04   1.633  0.15354
## Channel65   -1.470e+05  6.911e+04  -2.127  0.07752 .
## Channel66    9.121e+04  4.461e+04   2.045  0.08688 .
## Channel67   -5.733e+03  2.197e+04  -0.261  0.80288
## Channel68   -6.290e+04  2.192e+04  -2.870  0.02843 *
## Channel69    6.421e+04  2.074e+04   3.096  0.02121 *
## Channel70   -1.749e+04  1.581e+04  -1.106  0.31111
## Channel71   -7.248e+03  1.934e+04  -0.375  0.72075
## Channel72    3.406e+04  1.185e+04   2.873  0.02830 *
## Channel73   -2.100e+04  1.132e+04  -1.855  0.11308
## Channel74   -3.314e+04  1.220e+04  -2.717  0.03480 *
## Channel75    7.039e+04  2.054e+04   3.427  0.01402 *
## Channel76   -3.187e+04  1.736e+04  -1.836  0.11597
## Channel77    2.061e+04  1.810e+04   1.138  0.29832
## Channel78   -1.180e+04  2.273e+04  -0.519  0.62225
## Channel79    2.669e+04  2.997e+04   0.890  0.40750
## Channel80   -6.051e+04  1.483e+04  -4.080  0.00650 **
## Channel81    1.386e+03  2.628e+04   0.053  0.95966
## Channel82    1.020e+05  4.694e+04   2.173  0.07275 .
## Channel83   -1.706e+05  4.688e+04  -3.640  0.01083 *
## Channel84    1.097e+05  2.892e+04   3.792  0.00905 **
## Channel85   -1.294e+05  3.600e+04  -3.594  0.01145 *
## Channel86    2.130e+05  4.345e+04   4.903  0.00270 **
## Channel87   -1.198e+05  3.818e+04  -3.139  0.02011 *
## Channel88   -2.199e+04  6.085e+04  -0.361  0.73021
## Channel89    7.974e+04  5.077e+04   1.571  0.16733
## Channel90   -1.711e+05  5.499e+04  -3.112  0.02079 *
## Channel91    2.107e+05  6.406e+04   3.289  0.01663 *
## Channel92   -1.959e+05  7.171e+04  -2.733  0.03407 *
## Channel93    2.874e+05  9.937e+04   2.892  0.02762 *
## Channel94   -3.064e+05  9.601e+04  -3.191  0.01881 *
## Channel95    2.048e+05  6.220e+04   3.292  0.01656 *
## Channel96   -5.600e+04  2.929e+04  -1.912  0.10441
## Channel97   -1.318e+04  3.050e+04  -0.432  0.68065
## Channel98   -2.724e+04  2.107e+04  -1.292  0.24375
## Channel99    3.556e+04  1.382e+04   2.573  0.04218 *
## Channel100  -1.206e+04  4.264e+03  -2.828  0.03006 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3191 on 6 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:  0.9994
## F-statistic:  1651 on 100 and 6 DF,  p-value: 1.058e-09

## Training MSE:  0.005709117 . Test MSE:  722.4294
```

With the very low training MSE and comparably high test MSE we have reason to believe that there is overfitting occurring with the model. From viewing the summary for the training model it was found to give a adjusted R-square of 0.9994% which gives an clear indication that the model is overfit.

```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced

## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
```



Analyzing the plots we find that the assumptions that the model residuals are i.i.d can be assumed to be approximatly fullfilled, it's clear that the residuals do not follow anything similar to a normal distribution with mean 0 and variance 1 when observing the QQ-plot. Hence the assumptions for the model are not fulfilled so we should not draw any conclusions from the model.

3.2

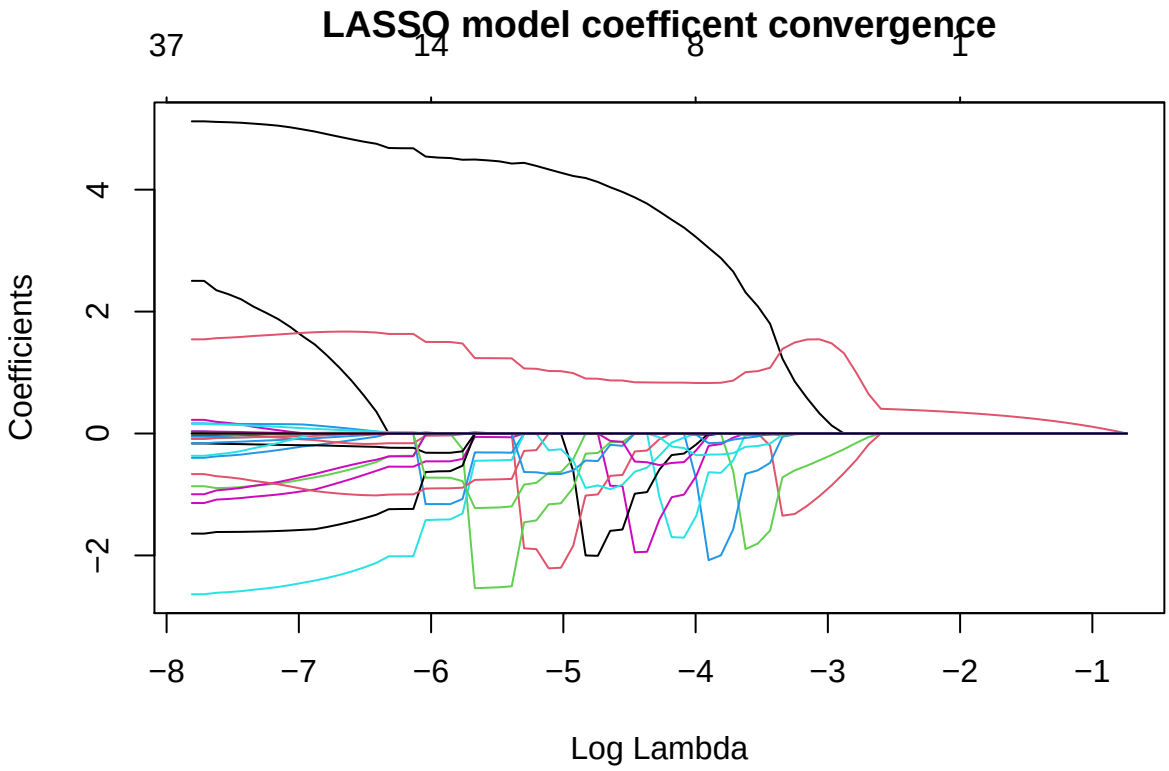
The function we are looking to optimize in this case is loss function of the LASSO model.

$$\sum_{i=1}^{N}(y_i - \mathbb{X}_i\boldsymbol{\beta})^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

Where N is the number of observations and $\mathbb{X}$ is the matrix of features and y is the dependent variable. β are the parameters that the shrinkage function right part of the plus sign. With λ being the shrinkage parameter and P the number of features. Where the chosen λ will shrink the β paramters.

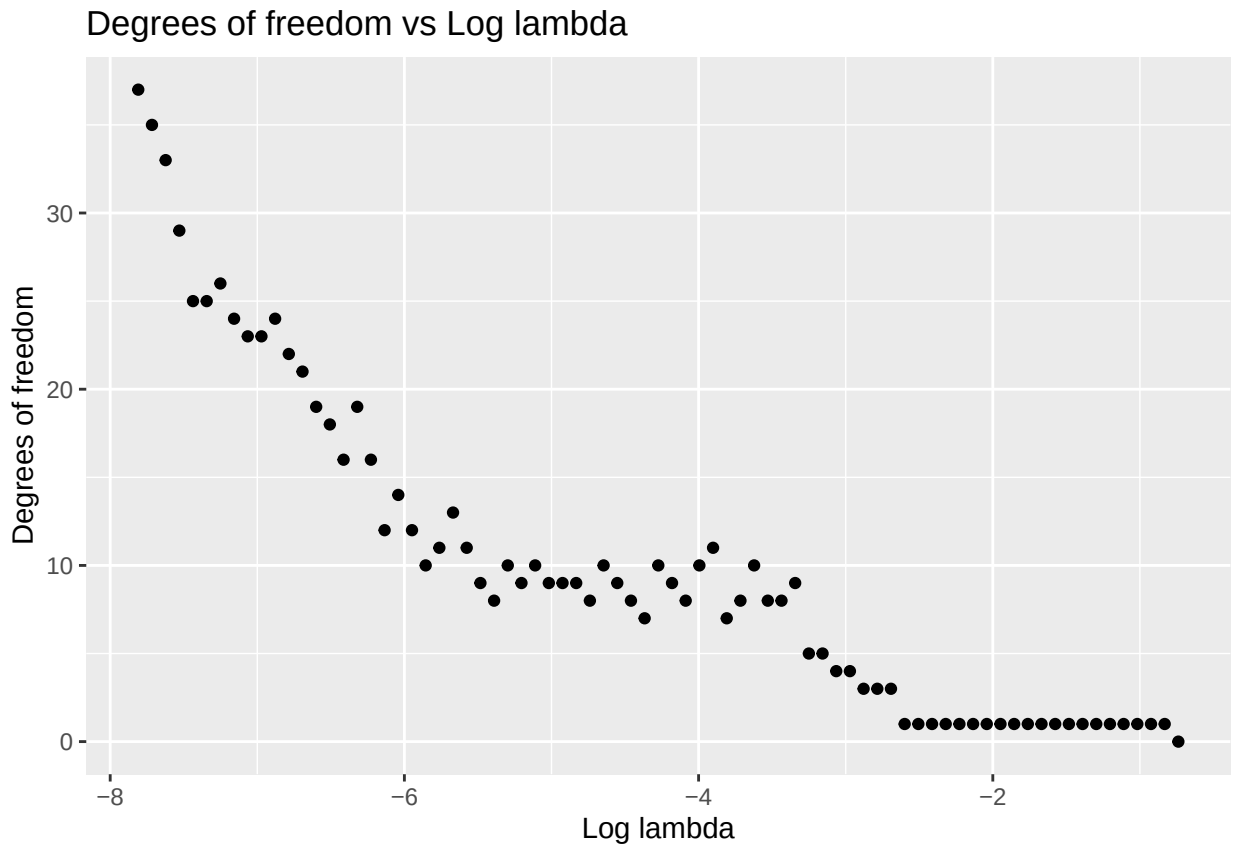
3.3

```
## [1] -2.688248
```

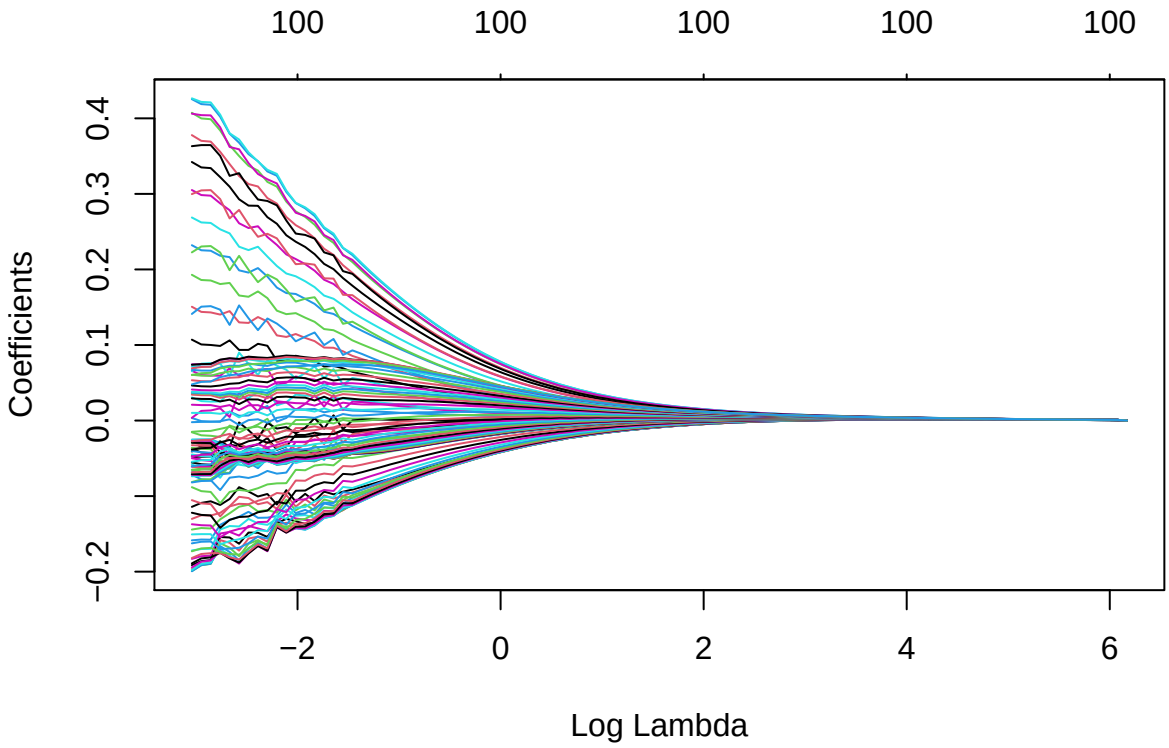


From the plot we can see that where the function converges to 3 features that have non zero coefficients is approximately at $e^{[}-2.6]$ and it was numerically shown that at $log(\lambda) = -2.688248$ the function would have a df of 3 and 3 features with non zero coefficents.

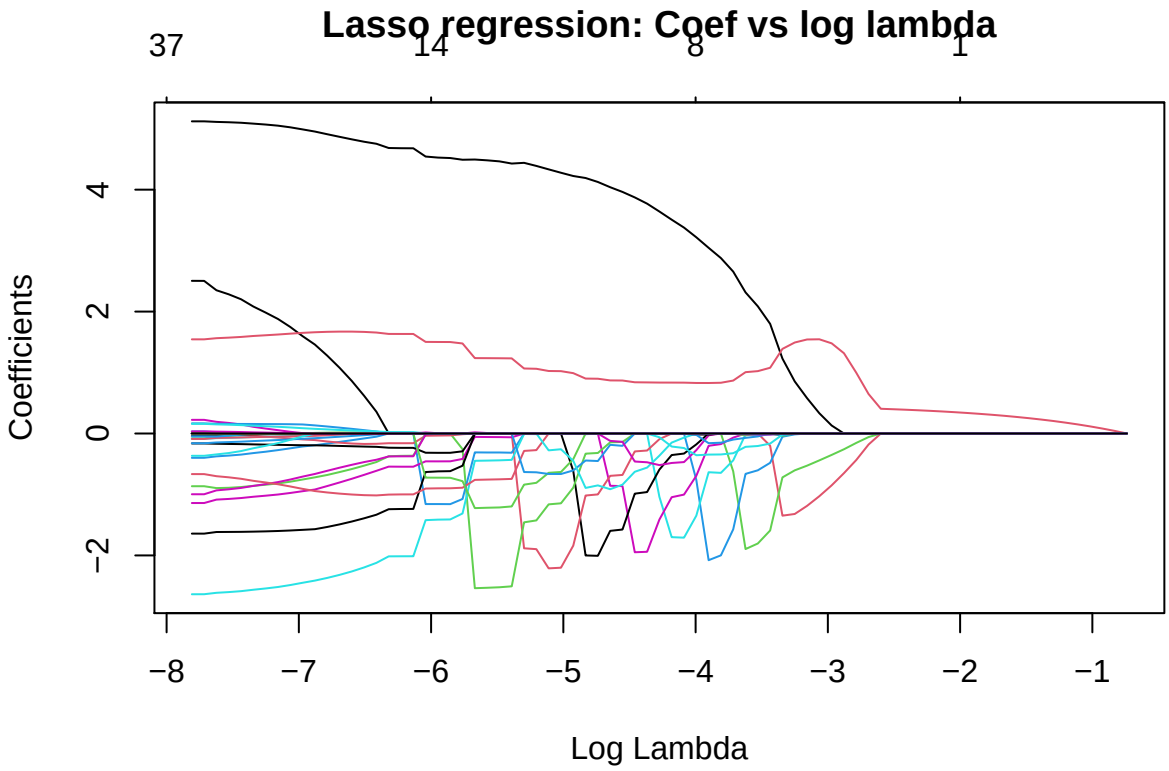
3.4

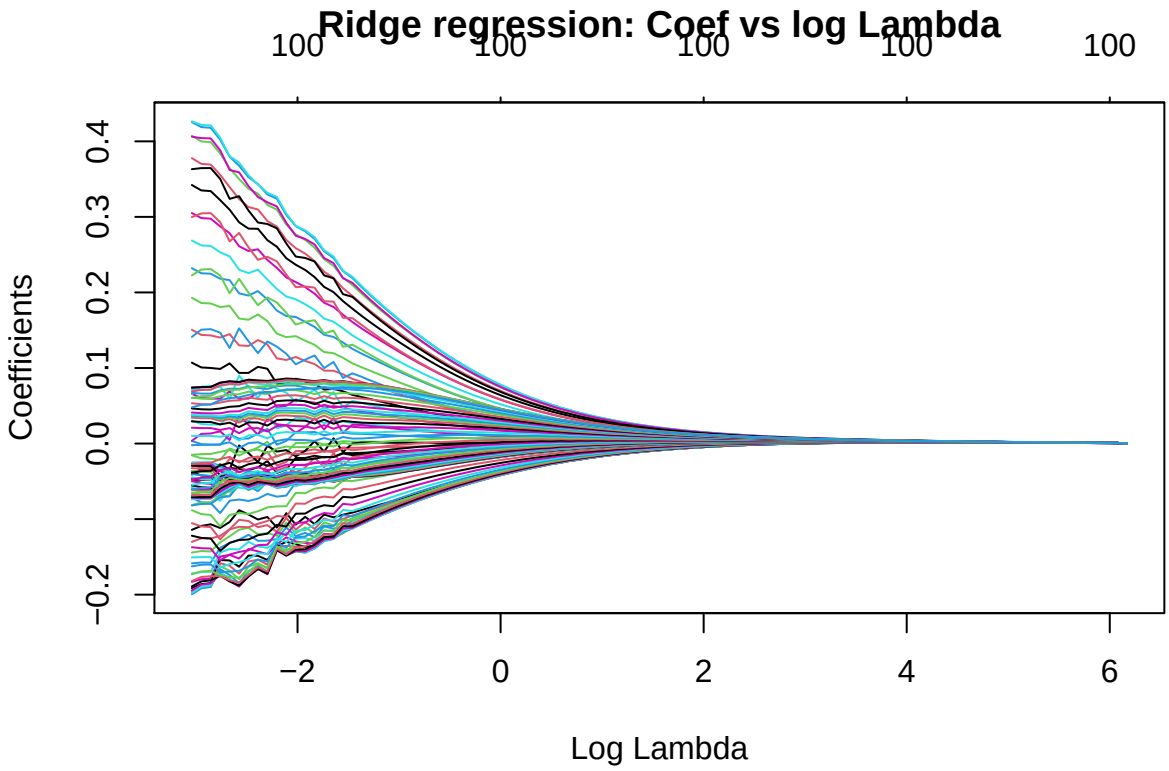


The graph shows that as the value of the shrinkage parameter lambda increases it reduces the number of features in the model. Which is exactly as expected from the given function with an sufficiently large λ it will reduce the number of features included to 0 as such decreasing the degrees of freedom.


100
100
100
100
100
Coefficients
0.4
0.3
0.2
0.1
0.0
−0.2
−2
0
2
4
6
Log Lambda

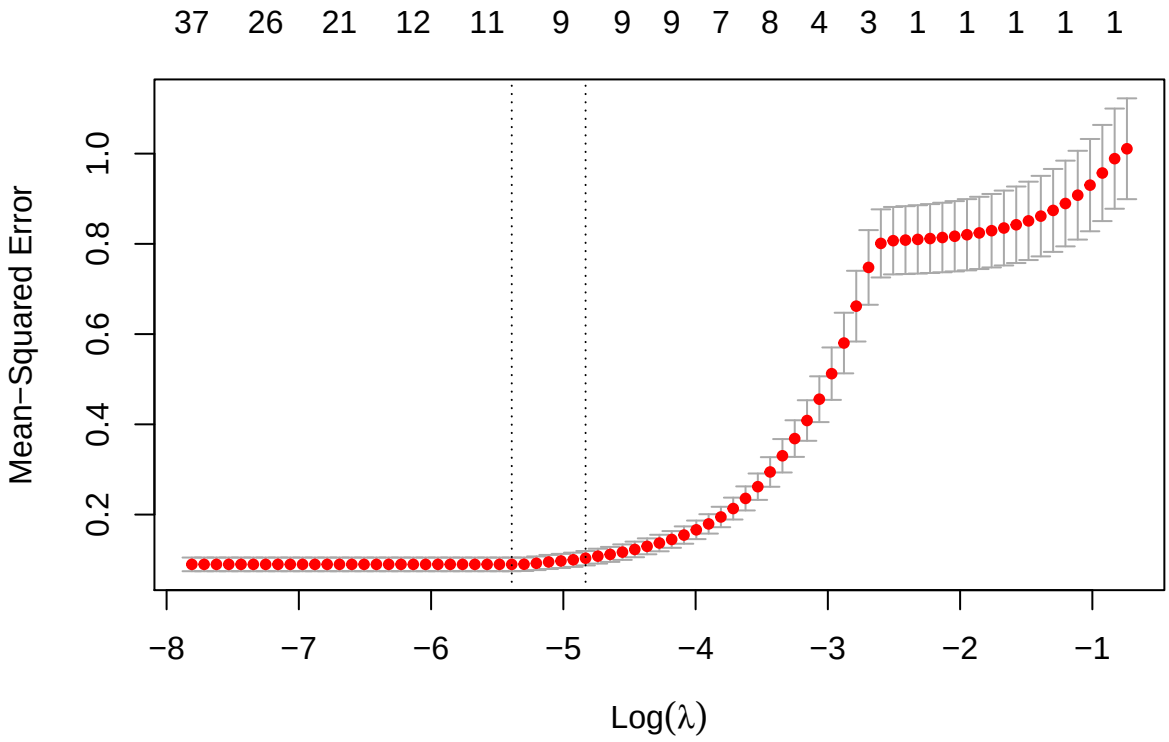
```
## [1] -2.688248
```


Lasso regression: Coef vs log lambda
37
14
8
1
Coefficients
4
2
0
−2
−8
−7
−6
−5
−4
−3
−2
−1
Log Lambda



As seen in the graphs the lasso function will converge the coefficients of the model to 0 whereas the ridge regression model will reduce the coefficients close to zero but never to zero.

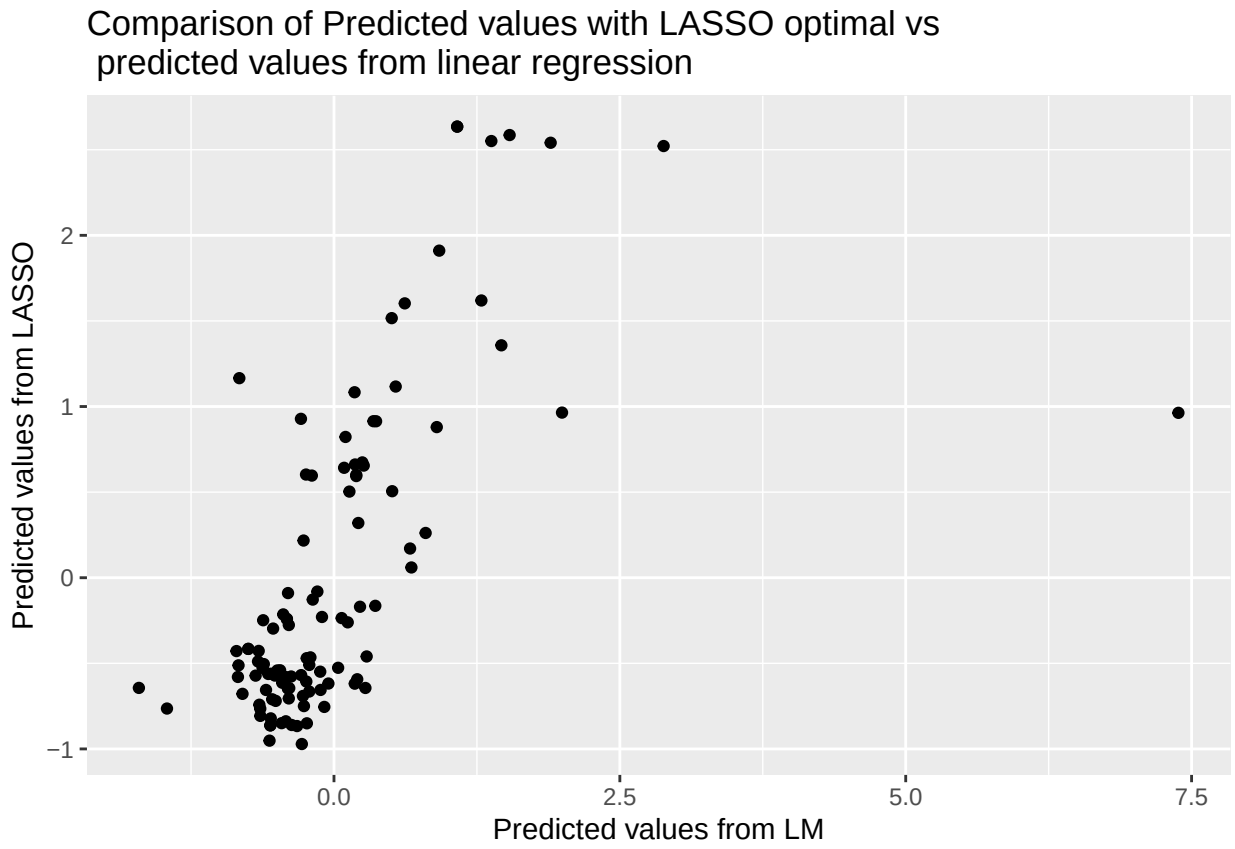
```
## [1] 0.004561105
```



The graph shows that the lowest MSE is found at $\lambda = 0.004561105$ which is at $log(\lambda) = -5.39019$ as shown in the graph. As the $log(\lambda)$ increases the MSE will increase with a linear rate from -8 to approx -5 and then with a non-linear increase until approximately -2.8 and then where it will slowdown the increasing rate of MSE.

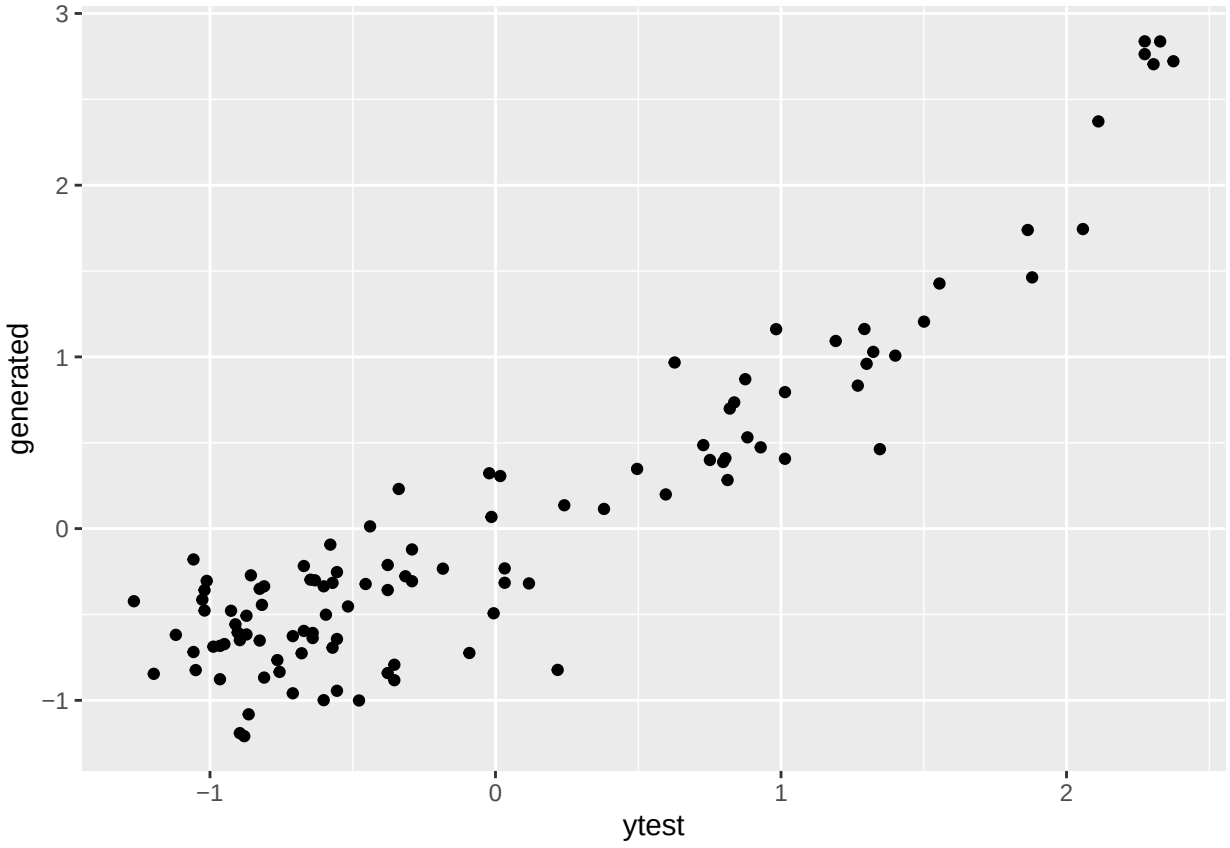
```
## [1] 7
```

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



3.7

```
preds <- predict(lasso3_6_opt, test_scale[,-101])

sigma<- sqrt(mean((train_scale[,101] - predict(lasso3_6_opt,train_scale[,-101]))^2 ))
new_targets <- rnorm(n = nrow(test_scale),preds, sigma )

generated_data <- data.frame("ytest" = test_scale[,101], "generated" = new_targets )
ggplot(generated_data, aes(x = ytest, y = generated)) + geom_point()
```



Appendix: Code

```
library(readr)
library(kknn)
library(ggplot2)
library(data.table)
library(glmnet)

#Assignment 1
#1.1
digits <- read.csv("optdigits.csv", header = F)
colnames(digits)[ncol(digits)] <- "number"
digits$number <- as.factor(digits$number)
n <- dim(digits)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- digits[id,]
id1 <- setdiff(1:n, id)
set.seed(12345)
id2 <- sample(id1, floor(n*0.25))
validation <- digits[id2,]
id3 <- setdiff(id1,id2)
test <- digits[id3,]
```

```r
#1.2
knn_train <- kknn(number ~., train, train, k = 30, kernel = "rectangular")
knn_test <- kknn(number ~., train, test, k = 30, kernel = "rectangular")
#Confusion matrix
pred_train<- fitted(knn_train)
confusion_train <- table(train$number, pred_train)
pred_test <- fitted(knn_test)
confusion_test <- table(test$number, pred_test)
print(confusion_train)
print(confusion_test)
print(diag(confusion_train)/rowSums(confusion_train))
print(diag(confusion_test)/rowSums(confusion_test))
#Misclassification errors
missclass <- function(X,X1) {
  n = length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
mismatch_rate_train <- missclass(train$number, pred_train) #0.04500262
mismatch_rate_test <- missclass(test$number, pred_test) #0.05329154
cat("The mismatch rate for train data is:", mismatch_rate_train,
    "\nThe mismatch rate for test data is:", mismatch_rate_test)

#1.3
train_eight <- train[which(train$number==8),]
train_eight$prob <- knn_train$prob[which(train$number==8), "8"]
sortedResult <- sort(train_eight[, "prob"], index.return = T)
sortedResult$x[1:3] #0.1000000 0.1333333 0.1666667
hard_ids <- sortedResult$ix[1:3] #50  43 136
tail(sortedResult$x, n=c(2)) #1 1
easy_ids <- tail(sortedResult$ix, n=c(2)) #179 183
plotHeatmap <- function(index){
  heatmap(matrix(as.numeric(train_eight[index, 1:64]), 8, 8, byrow = T), Colv = NA, Rowv = NA)
}
for (i in c(hard_ids, easy_ids)) {
  plotHeatmap(i)
}

#1.4
e_t <- 0
e_v <- 0
for (k in 1:30) {
  kt <- kknn(number ~., train, train, k = k, kernel = "rectangular")
  e_t[k] <- missclass(train$number, fitted(kt))
  kv <- kknn(number ~., train, validation, k = k, kernel = "rectangular")
  e_v[k] <- missclass(validation$number, fitted(kv))
}
plot(1:30, e_v, ylim = c(0,0.06), type = "l", col = "red", xlab = "K", ylab = "miss-classification erro
lines(1:30, e_t, ylim = c(0,0.06), col = "blue")
legend(2, 0.06, legend=c("validation", "train"),
       col=c("red", "blue"), lty=1:1, cex=0.8)

k_test_optik <- kknn(number ~., train, test, k = 3, kernel = "rectangular")
cat("Test mis-classification rate when k=3 is:", missclass(test$number, fitted(k_test_optik)))
```

```r
cat("Train mis-classification rate when k=3 is:", e_t[3])
cat("Validation mis-classification rate when k=3 is:", e_v[3])

#1.5
r_v <- 0
for (k in 1:30) {
  kv <- kknn(number ~., train, validation, k = k, kernel = "rectangular")
  s <- 0
  for (i in 0:9) {
    s <- s + sum(log2(kv$prob[which(validation$number==i), toString(i)] + 1e-15)/nrow(kv$prob))
  }
  r_v[k] <- -s/10
}
plot(1:30, r_v, ylim = c(0,0.14), type = "l", xlab = "K", ylab = "cross entropy")


### Question 2
# read in data
parkinsons <- read_csv("parkinsons.csv", col_types = cols())

# drop columns not to be used for prediction
parkinsons <- parkinsons[, -c(1, 2, 3, 4, 6)]

# bayesian model

## $$\mathbf{y} | \mathbb{X}, w_0, \mathbf{w} \sim N(w_0 + \mathbb{X}\mathbf{w}, \,
## \sigma^2 \mathbb{I})$$
## $\boldsymbol{w}$ is: $$\mathbf{w} \sim N(0, \, \frac{\sigma^2}{\lambda}
## \mathbb{I})$$

# create train/test partitions
all_indices <- 1:(nrow(parkinsons))
set.seed(12345)
train_indices <- sample(all_indices, ceiling(0.6*nrow(parkinsons)))
test_indices <- all_indices[!(all_indices %in% train_indices)]
train_data <- parkinsons[train_indices, ]
test_data <- parkinsons[test_indices, ]

# scale data -- see https://sebastianraschka.com/faq/docs/scale-training-test.html
# for why training mu/sigma are used for scaling test data
n_col <- ncol(parkinsons)
train_mu <- sapply(1:n_col, function(x) mean(unlist(train_data[x])))
train_sigma <- sapply(1:n_col, function(x) sd(unlist(train_data[x])))

train_data <- as.matrix(as.data.frame(
  sapply(1:n_col, function(x) (train_data[x] - train_mu[x])/train_sigma[x])))
# as.matrix has no effect without as.data.frame!!!
test_data <- as.matrix(as.data.frame(
  sapply(1:n_col, function(x) (test_data[x] - train_mu[x])/train_sigma[x])))

# set up helper functions

model_log_likelihood <- function(w_vector, sigma, X) { # built for no intercept
```

```
  # models, as asked

  n <- nrow(X)
  term1 <- -n/2*log(2*pi*sigma^2)
  term2 <- -sum(sapply(1:n, function(x) X[x, 1] - X[x, -1] %*% w_vector)^2)/
    2/sigma^2 # assumes y = X[, 1]

  return(term1 + term2)

}

ridge <- function(w_vector_sigma, X, lambda) { # built for no intercept models
  # as asked

  n <- ncol(X)
  sigma <- w_vector_sigma[n]
  w_vector <- w_vector_sigma[1:(n-1)]
  return(lambda*sum(w_vector^2) - model_log_likelihood(w_vector, sigma, X))

}

ridge_opt <- function(X, lambda) {

  n <- ncol(X)
  par <- rnorm(n-1) # initial parameter values (to seed optimisation); random
  par[n] <- 1 # initialise with positive value
  return(optim(par = par, fn = ridge, method = "BFGS", X = X, lambda = lambda)$par)

}

degrees_of_freedom <- function(X, num_samples,
                               num_simulations, optimal_weigths_sigma) {

  n <- ncol(X)
  r <- nrow(X)
  optimal_weigths <- optimal_weigths_sigma[1:(n-1)] # assumes no intercept
  optimal_sigma <- optimal_weigths_sigma[n]
  sum_cov = 0

  while (num_simulations > 0) {

    set.seed(num_simulations)
    sample_data <- X[sample(1:r, num_samples), ]
    sum_cov <- sum_cov + cov(sample_data[, 1],
                             sample_data[, -1] %*% optimal_weigths)
    num_simulations = num_simulations - 1

  }

  return(as.numeric(sum_cov/optimal_sigma^2))

}
```

```r
MSE <- function(X, optimal_weights) { # based on definition here: https://en.wikipedia.org/wiki/Mean_sq

  n <- nrow(X)
  return(sum((X[, 1] - X[, -1] %*% optimal_weights)^2)/n) # built for
  # no-intercept models as asked

}

# model 1: lambda = 1
model1_optimal_weights_sigma <- ridge_opt(train_data, lambda = 1)
model1_optimal_weights <- model1_optimal_weights_sigma[1:(n_col - 1)]
model1_optimal_sigma <- model1_optimal_weights_sigma[n_col]
model1_train_MSE <- MSE(train_data, model1_optimal_weights)
model1_test_MSE <- MSE(test_data, model1_optimal_weights)
model1_AIC <- -2*model_log_likelihood(model1_optimal_weights, model1_optimal_sigma, as.matrix(train_data
  2*degrees_of_freedom(as.matrix(train_data), num_samples = 1000,
                       num_simulations = 50, model1_optimal_weights_sigma)

# model 2: lambda = 100
model2_optimal_weights_sigma <- ridge_opt(train_data, lambda = 100)
model2_optimal_weights <- model2_optimal_weights_sigma[1:(n_col - 1)]
model2_optimal_sigma <- model2_optimal_weights_sigma[n_col]
model2_train_MSE <- MSE(train_data, model2_optimal_weights)
model2_test_MSE <- MSE(test_data, model2_optimal_weights)
model2_AIC <- -2*model_log_likelihood(model2_optimal_weights, model2_optimal_sigma, as.matrix(train_data
  2*degrees_of_freedom(as.matrix(train_data), num_samples = 1000,
                       num_simulations = 50, model2_optimal_weights_sigma)

# model 3: lambda = 1000
model3_optimal_weights_sigma <- ridge_opt(train_data, lambda = 1000)
model3_optimal_weights <- model3_optimal_weights_sigma[1:(n_col - 1)]
model3_optimal_sigma <- model3_optimal_weights_sigma[n_col]
model3_train_MSE <- MSE(train_data, model3_optimal_weights)
model3_test_MSE <- MSE(test_data, model3_optimal_weights)
model3_AIC <- -2*model_log_likelihood(model3_optimal_weights, model3_optimal_sigma, as.matrix(train_data
  2*degrees_of_freedom(as.matrix(train_data), num_samples = 1000,
                       num_simulations = 50, model3_optimal_weights_sigma)

# plot performance
MSE_data <- data.frame(log_10_lambda = c(0, 0, 2, 2, 3, 3),
                       Legend = rep(c("Training", "Test"), 3),
                       value = c(model1_train_MSE, model1_test_MSE,
                                 model2_train_MSE, model2_test_MSE,
                                 model3_train_MSE, model3_test_MSE))

ggplot(MSE_data) + geom_line(aes(log_10_lambda, value, colour = Legend) +
  theme_bw() +
  geom_vline(xintercept = 2, linetype = "dotted") +
  xlab("Log (base-10) of Hyper-Parameter 'lambda'") +
  ylab("Mean Squared Error") +
  ggtitle("Finding the Optimal Hyper-Parameter")

AIC_data <- data.frame(log_10_lambda = c(0, 2, 3),
```

```
                        value = c(model1_AIC, model2_AIC, model3_AIC))

ggplot(AIC_data) + geom_line(aes(log_10_lambda, value)) + theme_bw() +
  geom_vline(xintercept = 2, linetype = "dotted") +
  xlab("Log (base-10) of Hyper-Parameter 'lambda'") + ylab("Model AIC") +
  ggtitle("Finding the Optimal Hyper-Parameter")


### Question 3

# read in data
tecator <- read_csv("tecator.csv", col_types = cols())

# drop columns not to be used for prediction
tecator <- tecator[, -c(1, 103, 104)]

# split into training/test
all_indices <- 1:(nrow(tecator))
set.seed(12345)
train_indices <- sample(all_indices, ceiling(0.5*nrow(tecator)))
test_indices <- all_indices[!(all_indices %in% train_indices)]
train_data <- tecator[train_indices, ]
test_data <- tecator[test_indices, ]

## linear regression

# model

# $$\mathbf{y} \sim N(\boldsymbol{\beta}\mathbb{X}, \, \sigma^2\mathbb{I})$$

# fit
linreg <- lm(Fat~., train_data)
y_train_pred <- predict(linreg)
y_test_pred <- predict(linreg, test_data)
train_mse <- sum((train_data$Fat - y_train_pred)^2)/nrow(train_data)
test_mse <- sum((test_data$Fat - y_test_pred)^2)/nrow(test_data)

cat("Training MSE is:", train_mse)
cat("\nTest MSE is: ", test_mse)

## lasso regression

# objective function

## $$\sum\limits_{i=1}^N{(y_i - \boldsymbol{\beta}\mathbb{X}_i)^2} + \lambda\sum\limits_{j=1}^p{|\beta_

# helper function: degrees of freedom
lasso_degrees_of_freedom <- function(lasso_model, y, X,
                                     num_simulations, num_samples) {

  X <- as.matrix(X)
  y <- unlist(y)
  y_pred <- as.numeric(predict(lasso, X))
```

```
  r <- nrow(X)
  sum_cov = 0

  while (num_simulations > 0) {

    set.seed(num_simulations)
    sample_indices <- sample(1:r, num_samples)
    sample_X <- X[sample_indices, ]
    sample_y <- y[sample_indices]
    sample_y_pred <- y_pred[sample_indices]
    sum_cov <- sum_cov + cov(sample_y, sample_y_pred)
    num_simulations = num_simulations - 1

  }

  sigma_hat_square <- sum((y - y_pred)^2)/num_samples

  return(sum_cov/sigma_hat_square)

}

# fit
lambda_choices <- c(10^-3, 10^-2, 10^-1, 1, 10, 100, 1000)
zero_coeff <- c()
sum_abs_val_params <- c()
lasso_deg_freedoms <- c()
for (lam in lambda_choices){

  lasso <- glmnet(x = as.matrix(subset(train_data, select = -Fat)),
                  y = unlist(train_data[, "Fat"]), alpha = 1, lambda = lam)
  sum_abs_val_params <- c(sum_abs_val_params, sum(abs(lasso$beta)))
  zero_coeff <- c(zero_coeff, sum(lasso$beta == 0))
  lasso_deg_freedoms <- c(lasso_deg_freedoms,
                          lasso_degrees_of_freedom(
                            lasso, tecator[, "Fat"],
                            subset(tecator, select = -Fat),
                            num_simulations = 50,
                            num_samples = 100))
  # using full dataset rather than train / test, because why not!

}

# plot results
lasso_results <- data.frame(log_10_lambda = rep(-3:3, 2),
                            Legend = c(
                              rep("# Parameters with Coefficients = 0", 7),
                              rep("Sum of absolute values of Coefficients", 7)),
                            value = c(zero_coeff, sum_abs_val_params))

ggplot(lasso_results) + geom_line(aes(log_10_lambda, value, colour = Legend) +
  theme_bw() +
  scale_x_continuous(breaks = -3:3) + theme(legend.position="bottom") +
  xlab("Log (base-10) of LASSO Hyper-Parameter 'lambda'") +
```

```
  ylab("Parameter Attributes") +
  ggtitle("Relationship between LASSO Lambda and parameter attributes")

lasso_deg_freedom <- data.frame(log_10_lambda = rep(-3:3, 2),
                                value = lasso_deg_freedoms)

ggplot(lasso_deg_freedom) + geom_line(aes(log_10_lambda, value)) + theme_bw() +
  scale_x_continuous(breaks = -3:3) +
  xlab("Log (base-10) of LASSO Hyper-Parameter 'lambda'") +
  ylab("Model Degrees of Freedom") +
  ggtitle("Relationship between LASSO Lambda and Model Degrees of Freedom")

## ridge regression

# fit
zero_coeff <- c()
sum_params_squared <- c()
for (lam in lambda_choices){

  ridge <- glmnet(x = as.matrix(subset(train_data, select = -Fat)),
                  y = unlist(train_data[, "Fat"]), alpha = 0, lambda = lam)
  sum_params_squared <- c(sum_params_squared, sum(ridge$beta^2))
  zero_coeff <- c(zero_coeff, sum(ridge$beta == 0))

}

# plot results
ridge_results <- data.frame(log_10_lambda = rep(-3:3, 2),
                            Legend = c(
                              rep("# Parameters with Coefficients = 0", 7),
                              rep("Sum of squared values of Coefficients", 7)),
                            value = c(zero_coeff, sum_params_squared))

ggplot(ridge_results) + geom_line(aes(log_10_lambda, value, colour = Legend)) +
  theme_bw() +
  scale_x_continuous(breaks = -3:3) + theme(legend.position="bottom") +
  xlab("Log (base-10) of Ridge Hyper-Parameter 'lambda'") +
  ylab("Parameter Attributes") +
  ggtitle("Relationship between Ridge Lambda and parameter attributes")

## lasso with CV

# fit
cv_means <- c()
cv_min_lambda <- c()
for (i in 1:100) {

  cv_lasso <- cv.glmnet(x = as.matrix(subset(train_data, select = -Fat)),
                        y = unlist(train_data[, "Fat"]), alpha = 1,
                        lambda = lambda_choices, nfolds = 3)
  cv_means <- cbind(cv_means, cv_lasso$cvm)
  cv_min_lambda <- c(cv_min_lambda, cv_lasso$lambda.min)
}
```

```r
# plot
ggplot(data.frame(log_10_lambda = -3:3, cv_score = rev(rowMeans(cv_means)))) +
  geom_line(aes(log_10_lambda, cv_score)) + scale_x_continuous(breaks = -3:3) +
  geom_vline(xintercept = -3, linetype = "dotted") +
  xlab("Log (base-10) of LASSO Hyper-Parameter 'lambda'") +
  ylab("Mean squared error across CV folds") +
  ggtitle("Relationship between LASSO Lambda and CV Scores")

opt_lambda <- cv_lasso$lambda.min
cat("Optimal Lambda is:", opt_lambda)

model_optimal_lambda <- glmnet(x = as.matrix(subset(train_data, select = -Fat)),
                               y = unlist(train_data[, "Fat"]), alpha = 1,
                               lambda = opt_lambda)
num_non_zero_params_opt_lambda <- sum(model_optimal_lambda$beta != 0)
cat("Number of variables chosen in the model:", num_non_zero_params_opt_lambda)

# comparing LASSO regression model with optimal lambda and
# another with log (base-10) lambda = -2
model_log_lambda_minus_2 <- glmnet(
  x = as.matrix(subset(train_data, select = -Fat)),
  y = unlist(train_data[, "Fat"]), alpha = 1, lambda = 10^-2)
y_test <- unlist(test_data[, "Fat"])
y_test_pred_opt_lambda <- predict(model_optimal_lambda,
                                  as.matrix(subset(test_data, select = -Fat)))
y_test_log_lambda_minus_2 <- predict(model_log_lambda_minus_2,
                                  as.matrix(subset(test_data, select = -Fat)))

num_non_zero_params_log_lambda_minus_2 <-
  sum(model_log_lambda_minus_2$beta != 0)

opt_lambda_SSE <- sum((y_test - y_test_pred_opt_lambda)^2)
log_lambda_minus_2_SSE <- sum((y_test - y_test_log_lambda_minus_2)^2)

opt_lambda_deg_freedom <-
  nrow(test_data) - num_non_zero_params_opt_lambda - 1
log_lambda_minus_2_deg_freedom <-
  nrow(test_data) - num_non_zero_params_log_lambda_minus_2 - 1

opt_lambda_MSE <- opt_lambda_SSE / opt_lambda_deg_freedom
log_lambda_minus_2_MSE <-
  log_lambda_minus_2_SSE / log_lambda_minus_2_deg_freedom
test_statistic <- opt_lambda_MSE / log_lambda_minus_2_MSE

# Null Hypothesis: opt_lambda_MSE = log_lambda_minus_2_MSE
# Alternative Hypothesis: opt_lambda_MSE < log_lambda_minus_2_MSE

p_value <- pf(q = test_statistic,
              df1 = opt_lambda_deg_freedom,
              df2 = log_lambda_minus_2_deg_freedom)

cat("The p-value for the test of null hypothesis that optimal",
    "lambda works similar to log(lambda) = -2\n",
```

```
    "vs. the alternative that the former is better is:", p_value)
cat("We, therefore, cannot conclude confidently that the optimal",
    "lambda works significantly better\n than log(lambda) = -2")

# plot predictions
ggplot(data.frame(y_test, y_test_pred_opt_lambda)) +
  geom_point(aes(y_test, y_test_pred_opt_lambda)) +
  xlab("Test Labels") + ylab("Test Predictions") +
  ggtitle("Goodness of fit - LASSO model with optimised lambda")

## generative model

# estimate sigma: use MLE now ;-)
y_train <- unlist(train_data[, "Fat"])
y_train_pred <- predict(model_optimal_lambda,
                        as.matrix(subset(train_data, select = -Fat)))
MLE_sigma_estimate <- sqrt(sum((y_train - y_train_pred)^2)/nrow(train_data))

# generate labels from distribution N(y_train_pred, MLE_sigma_estimate*I)
y_generated <- rnorm(length(y_test), y_test_pred_opt_lambda, MLE_sigma_estimate)

# plot generated labels vs. original labels
ggplot(data.frame(y_test, y_generated)) +
  geom_point(aes(y_test, y_generated)) +
  xlab("Test Labels") + ylab("Generated Labels") +
  ggtitle("Goodness of fit - Generative LASSO model with optimised lambda")
```