Homework 3

Tanner Huck

2023-05-10

Problem 1

Intro) We want to predict the quantitative response of y, the maximum speed of a train using the linear model $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$. Where x_1 is the weight of the train in tonnes and x_2 is the weight of the train in kilograms. Also suppose that $\hat{\beta}_0$, $\hat{\beta}_1$, and $\hat{\beta}_2$, are the least squares solutions to the model in our linear model.

a. First we want to explain why the least squares solution is not unique. We know that the least squares solution is not unique because there may be multiple combinations of the β coefficients that can minimize the residual sum of squares. We can show this by deriving a general expression for the set of least squares coefficient estimates in terms of $\hat{\beta}_0$, $\hat{\beta}_1$, and $\hat{\beta}_2$. First, if we rewrite x_2 in terms of x_1 we have $x_2 = 0.001x_1$ (tonnes to kilograms). Then we can substitute this into our linear model and get, $y = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2(0.001x_1)$, which can be simplified to $y = \hat{\beta}_0 + x_1(\hat{\beta}_1 + 0.001\hat{\beta}_2)$. Then we can rewrite this once again to $y = \hat{\beta}_0 + \hat{\beta}_3 x_1$ where $\hat{\beta}_3 = (\hat{\beta}_1 + 0.001\hat{\beta}_2)$.

Then to find a general expression for the set of least squares coefficient estimates that also minimizes the residual sum of squares, we can start with the definition of RSS,

$$RSS = \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_3 x_i)^2$$

$$\text{then finding the derivative and setting it equal to 0 to find the critical points}$$

$$\frac{d}{d\hat{\beta}_0}RSS = \frac{d}{d\hat{\beta}_0}\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_3 x_i)^2 = 0$$

$$\frac{d}{d\hat{\beta}_0}\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_3 x_i)^2 = 0$$

$$\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_3 x_i) = 0$$

$$\sum_{i=1}^{n} y_i - \sum_{i=1}^{n}\hat{\beta}_0 - \sum_{i=1}^{n}\hat{\beta}_3 x_i = 0$$

$$\sum_{i=1}^{n} y_i - n\hat{\beta}_0 - \hat{\beta}_3\sum_{i=1}^{n} x_i = 0$$

$$\hat{\beta}_0 = -\frac{-\sum_{i=1}^{n} y_i + \hat{\beta}_3\sum_{i=1}^{n} x_i}{n}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_3\bar{x}$$

Thus from our definition of $\hat{\beta}_3$, we also know that,

$$\frac{d}{d\hat{\beta}_3}\sum_{i=1}^{n} x_i(y_i - \hat{\beta}_0 - \hat{\beta}_3 x_i)^2 = 0$$

$$\text{plugging in our found value of } \hat{\beta}_0$$

$$\frac{d}{d\hat{\beta}_3}\sum_{i=1}^{n} x_i(y_i - (\bar{y} - \hat{\beta}_3\bar{x}) - \hat{\beta}_3 x_i)^2 = 0$$

$$\text{following similar steps as above we get,}$$

$$\sum_{i=1}^{n} x_i(y_i - \bar{y}) - \hat{\beta}_3\sum_{i=1}^{n} x_i(x_i - \bar{x}) = 0$$

$$\text{then solving for } \hat{\beta}_3$$

$$\hat{\beta}_3 = \frac{\sum_{i=1}^{n} x_i(y_i - \bar{y})}{\sum_{i=1}^{n} x_i(x_i - \bar{x})}$$

$$\hat{\beta}_3 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Hence we have that $y = \hat{\beta}_0 + (\hat{\beta}_1 + 0.001\hat{\beta}_2)x_1$ where $\hat{\beta}_1 + 0.001\hat{\beta}_2 = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}$.

b. Now, we can assume that we observe n data points $(x_{1i}, x_{2i}, y_i)_{i=1}^{n}$. We also assume that $\beta_0 = 0$, so the linear model is $y = \beta_1 x_1 + \beta_2 x_2$. We want to perform regularization using the new model. The expression for ridge regression can be written as the minimum of,

$$RSS + \lambda \cdot (\beta_1^2 + \beta_2^2)$$

Where $RSS = \sum_{i=1}^{n}(y_i - (\beta_1 x_{1i} + \beta_2 x_{2i}))$ and λ is the tuning parameter. Whereas the expression for Lasso regression can be written as the minimum of

$$RSS + \lambda \cdot (|\beta_1| + |\beta_2|)$$

Where RSS and λ are defined the same.

c. Now assume that $\beta_1 + 1000\beta_2 = c$ where c is a fixed constant, then rearranging we get $\beta_1 = c - 1000\beta_2$. Rewritten the ridge regression in part b, and substituting in β_1 we get,

$$\begin{aligned} RSS + \lambda \cdot (\beta_1^2 + \beta_2^2) &= RSS + \lambda \cdot ((c - 1000\beta_2)^2 + \beta_2^2) \\ &= RSS + \lambda \cdot (c^2 - 2000c\beta_2 + 1000^2\beta_2^2 + \beta_2^2) \\ &= RSS + \lambda \cdot (c^2 - 2000c\beta_2 + (1000^2 + 1)\beta_2^2) \end{aligned}$$

Then to find the coefficients that solve our equation above (to minimize it) we can take the derivative with respect to β_2 and set it equal to 0.

$$\frac{d}{d\beta_2}RSS + \lambda \cdot (c^2 - 2000c\beta_2 + (1000^2 + 1)\beta_2^2 = 0$$

$$\frac{d}{d\beta_2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_{1i} - \beta_2 x_{2i})^2 + \lambda c^2 - \lambda 2000 c\beta_2 + \lambda(1000^2 + 1)\beta_2^2 = 0$$

$$\sum_{i=1}^{n}[(-2x_{2i})(y_i - \beta_0 - \beta_1 x_{1i} - \beta_2 x_{2i})] - \lambda 2000c + \lambda 2(1000^2 + 1)\beta_2 = 0$$

Hence the minimum of $\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_{1i} - \beta_2 x_{2i})^2 + \lambda c^2 - \lambda 2000 c\beta_2 + \lambda(1000^2 + 1)\beta_2^2$ or equivalently, solving the above equation for β_2 will be the coefficient that solves the ridge regression problem. This is not unique because we are dependent on the coefficient c. Also note that I solved for and substituted in β_1 at the beginning, where you could have solved for β_2 instead.

Problem 2

Intro) For this question I will be using the "Online News Popularity Data Set" data set by Kelwin Fernandes found on UCI Machine Learning Repository, which can be accessed here: https://canvas.uw.edu/courses/1635467/assignments/8196342 (https://canvas.uw.edu/courses/1635467/assignments/8196342).

a. This data set is meant to help summarize a set of features about articles published by "Mashable" from a period of two years and use this to predict the number of shares in a social network. The response variable in the data set is the number of shares a specific news article on a social media platform has. This may represent the popularity or how "viral" an article may be among readers. There are 60 predictors in this data set. Each predictor provides a feature related to the content of the article, for example keyword count, global subjectivity, or "avg_negative_polarity". These factors aim to determine what may lead an article to receive more or less shares. This may be beneficial analysis because online news creators, marketers, and publishers can see which features lead to a more popular article and target those features more frequently. Here is what the data looks like,

```
df <- read.csv("OnlineNewsPopularity.csv")
head(data)
```

```
##
## 1 function (..., list = character(), package = NULL, lib.loc = NULL,
## 2     verbose = getOption("verbose"), envir = .GlobalEnv, overwrite = TRUE)
## 3 {
## 4     fileExt <- function(x) {
## 5         db <- grepl("\\\\.[^.]+\\\\.(gz|bz2|xz)$", x)
## 6         ans <- sub(".*\\\\.", "", x)
```

At this point I spent way to long trying to get forward and backwards selection to work and it kept crashing my R. So instead I am going to start and generate my own data.

Take 2

a. To start again, I will generate my own data. For my data I chose to have 32 predictors with an $n = 500$ number of observations. To do this, I will generate 30 of the predictors at random (with rnorm) and manually create 2 additional rows that are linear combinations of previous rows to have them be linearly dependent.

```r
set.seed(7979)

predictors <- 30
n <- 500

x <- matrix(rnorm(n * predictors), ncol = predictors)
y <- rnorm(n)
data <- data.frame(y = y, x)

# Lin. Dep. vars
data$X31 <- data$X1 * 2
data$X32 <- data$X1 * 3 + 1

head(data)
```

```
##             y          X1          X2          X3            X4         X5
## 1  0.19592898  1.32920433 -0.70274024 -0.39076051  0.0009280358  0.4583434
## 2 -1.27161244 -0.37575061  0.02457686  0.39789883 -0.6526186541 -1.1547437
## 3 -0.26918827  0.07677833  0.92925357  0.86838341  0.6188688144  1.8973996
## 4  0.97835373 -0.62838240 -1.55824275 -1.73329611  0.5068670072 -0.6063509
## 5  0.61408458 -0.54552321  0.41006828  0.05526675 -1.9227525873 -0.4370071
## 6  0.04310178 -0.77710822 -0.45614593  0.16977756 -2.0253881075  1.2671688
##           X6          X7         X8         X9         X10        X11
## 1  0.8934638 -0.09900639  0.6305907 -0.9408223  0.07234861 -0.2278760
## 2 -1.8314121  0.48033961  0.7751511  0.2485555 -0.61018911 -0.9157603
## 3 -0.2958261 -0.03431408 -0.2054836  0.9715199  0.65406316  0.3461215
## 4 -1.1944676 -1.18259731  0.4176352  1.0872698  0.94419439  0.3075700
## 5 -0.1681683 -0.29730285 -0.5947640 -0.8215694 -0.44528691  0.9671268
## 6 -0.3452754  0.05376824 -0.2984747 -0.5549735 -0.48213645 -1.2689856
##           X12        X13         X14        X15          X16        X17
## 1 -1.74254198 -1.6673753  0.11436561  1.1011058  0.751703404  0.7349271
## 2  0.19678644 -1.5300916 -0.92473578 -0.2038395  0.609066117  0.2875268
## 3 -0.07056667 -0.3023053 -0.47232161  0.4316470 -0.106430728 -1.6283870
## 4 -0.05652196 -0.1419161  0.50206411  0.5430492 -0.006192663  0.2215094
## 5  0.76065786 -0.3186166 -0.01083247  0.1687437  0.405511949 -0.5913592
## 6 -0.01485145  0.5802625  0.95774623 -0.5443340 -1.945166711 -0.4804747
##          X18        X19         X20        X21         X22        X23
## 1  0.3871093 -0.1231682 -0.07991744  0.3034309 -0.02686622  0.3961666
## 2  0.6186720 -0.3278889 -0.47613033  2.1055592  1.53462449  0.2515168
## 3  1.2160243  2.2818617 -0.13771832 -0.7518732  0.59124053 -0.0217126
## 4 -0.1699155 -0.8764134 -0.22647199 -1.0223246 -0.71681166  0.1304916
## 5  0.4274583  0.6914035 -0.15472626  0.6170310  1.14051480  1.1302236
## 6  1.7628491 -0.1895912 -0.48772767 -0.6908660 -2.05910790  0.3323941
##           X24        X25        X26        X27        X28         X29
## 1 -1.00903729  0.5303098  0.8721359 -1.3999787  0.2495790 1.337701283
## 2  1.78305061  0.9981188  0.4917011  0.6007708 -0.3923382 0.899125681
## 3  0.02043608  0.8467333 -0.6518837 -1.1626578  0.7777994 0.005714963
## 4 -0.91126407  0.5868384  0.4752895  1.4954949  0.5966291 1.124929552
## 5 -1.25154826  1.4112339 -1.2595297 -0.5167007  0.7390225 0.730674001
## 6 -1.89590017 -1.0585016 -0.4177909 -0.2706461 -1.9777192 0.078577274
##          X30        X31        X32
## 1  1.8158248  2.6584087  4.9876130
## 2  1.4284953 -0.7515012 -0.1272518
## 3 -1.0557566  0.1535567  1.2303350
## 4  2.1110370 -1.2567648 -0.8851472
## 5 -0.1259502 -1.0910464 -0.6365696
## 6  0.2135843 -1.5542164 -1.3313247
```

Thus we have 32 predictors $x1, \dots, x32$ that can be used to predict th y variable.

b. Now I will split the generated data into 80% training and 20% testing.

```
set.seed(7979)

train_index <- sample(1:nrow(data), 0.8 * nrow(data))
train <- data[train_index, ]
test<- data[-train_index, ]

dim(train)
```

```
## [1] 400  33
```

Hence we can see that there are 400 observations in our training set with 32 predictors and 1 response.

Now I will use forward selection to fit a reduced linear model on the training data.

```
forward <- regsubsets(y ~ ., data = train, nbest = 1, nvmax = 10, method = "forward",
really.big = T)

coef(forward, 10)
```

```
## (Intercept)          X4         X12         X17         X18         X19
## -0.02395332  0.06849586  0.06121637 -0.05133198  0.05357114  0.05704991
##         X22         X23         X26         X27         X30
## -0.10303088 -0.07720288 -0.10283434  0.09284493  0.06766847
```

Here we can see the 10 predictors that were chosen by the forward selection method to include in the linear model along with their coefficients.

Now to find the error on the test data.

```
model <- lm(y ~ X4 + X12 + X17 + X18 + X19 + X22 + X23 + X26 + X27 + X30, data=train)
y_hat <- predict(model, test)
MSE <- mean((test$y - y_hat)^2)
MSE
```

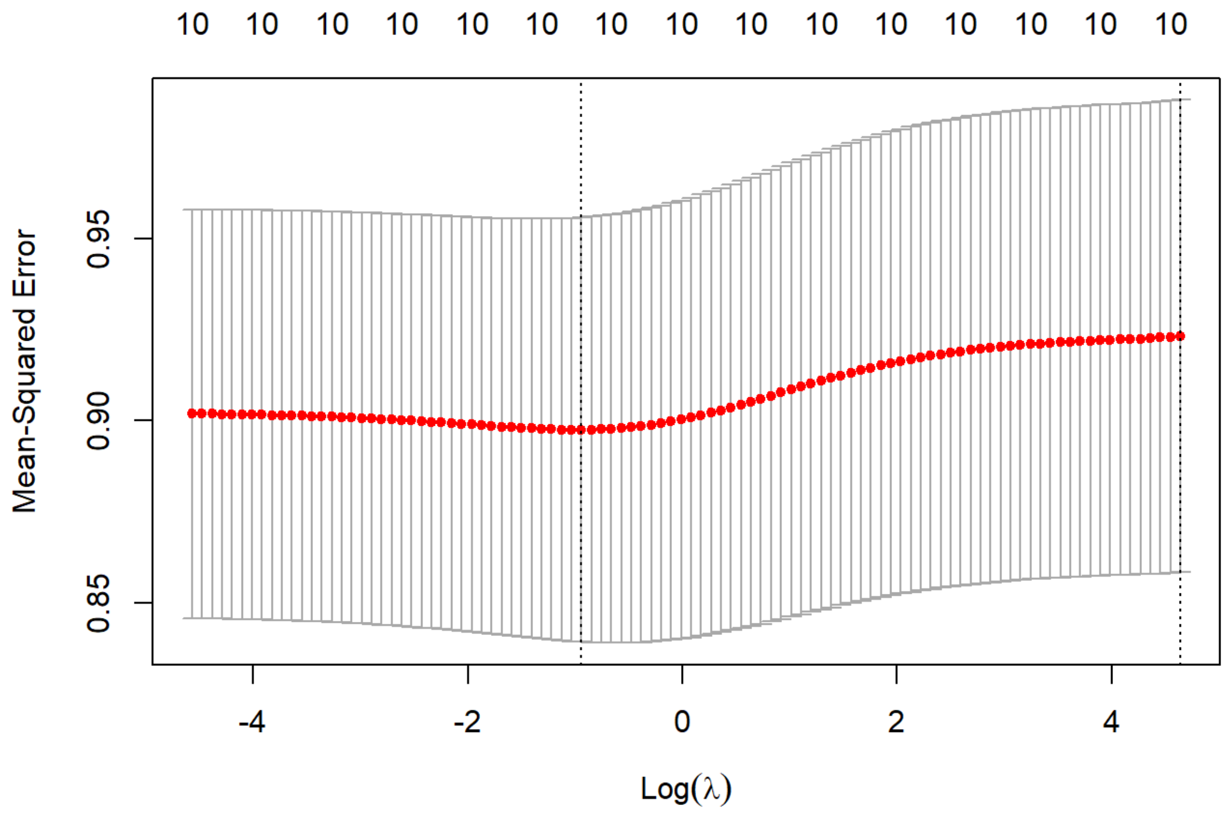
```
## [1] 0.917617
```

Hence the error we find on the test data from our linear model is about 0.92.

c. Now with the subset of predictors from our linear model in part b, we want to fit a ridge regression model on the training data with a range of values for the regularization parameter λ (which we will find using the grid function using k-fold cross validation with $k = 10$).

```
best_predictors <- train %>% dplyr::select(y, X4, X12, X17, X18, X19, X22, X23, X26, X27, X30)
X <- model.matrix(y ~ ., data = best_predictors)[,-1]
Y <- best_predictors$y
grid <- 10^seq(10, -2, length = 500)
ridge.model <- glmnet(X, Y, alpha = 0, lambda = grid)

cv.out <- cv.glmnet(X, Y, alpha = 0)
plot(cv.out)
```



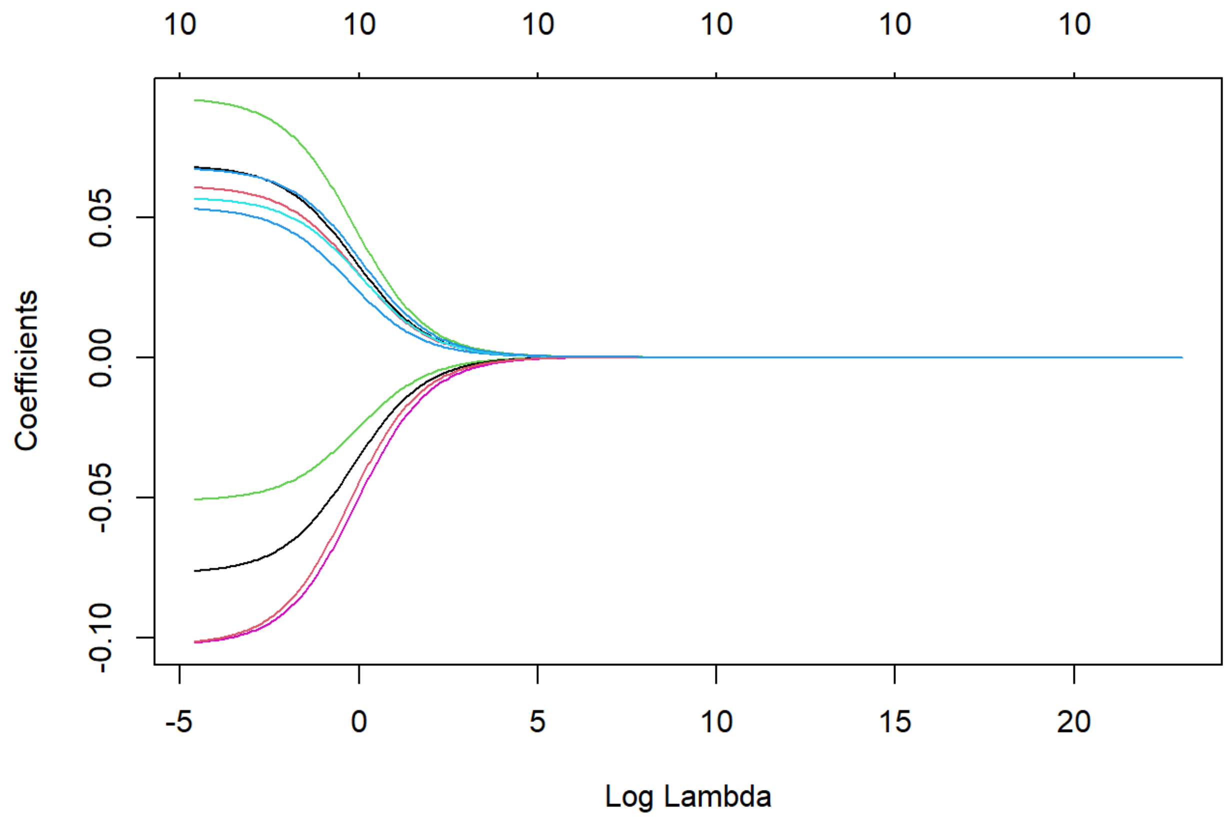
```
best_lambda <- cv.out$lambda.min
best_lambda
```

```
## [1] 0.3901575
```

In this plot we can see the mean squared error for different values of the log of λ. We also learn the optimal value of λ is 0.3901575.

Now creating a ridge regression model on the training data with a range of λ values as the regularization parameter.

```
plot(ridge.model, xvar="lambda")
```



This plot has the log values of λ on the x-axis and the Coefficients along the y-axis. We can see that regardless of the initial λ value or coefficient value, all the coefficient values merge to 0 around log λ values of about 3 to 4.

d. Now using the optimal $\lmbda$ value that we found, we want to find the test error of the ridge regression model.

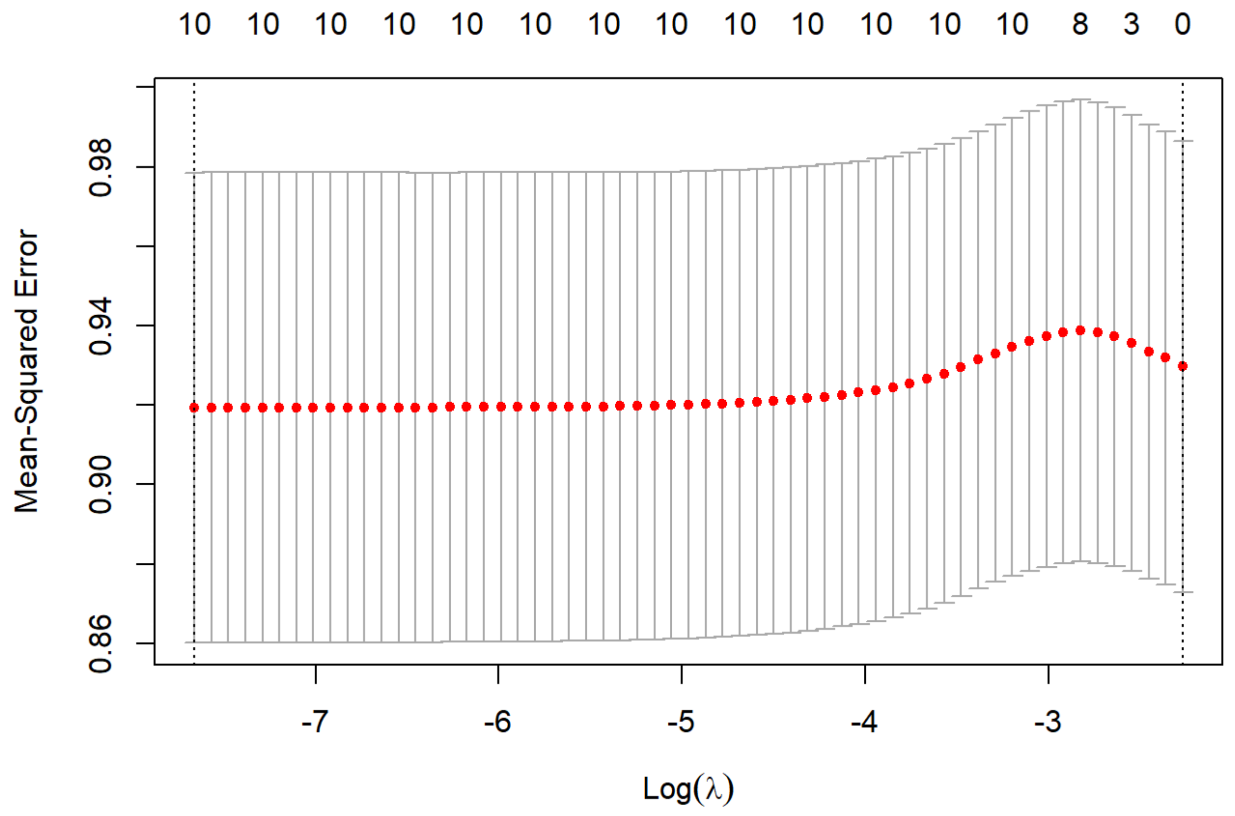
```
y_hat <- predict(ridge.model, s = best_lambda, newx = X)
MSE <- mean((y_hat - Y)^2)
MSE
```

```
## [1] 0.8712206
```

We can see that the error from the ridge regression model is about 0.873, which is a bit better than the error from the linear model in part b which had an error of about 0.92.

e. Now we will repeat parts c though d with a lasso regression instead of a ridge regression.

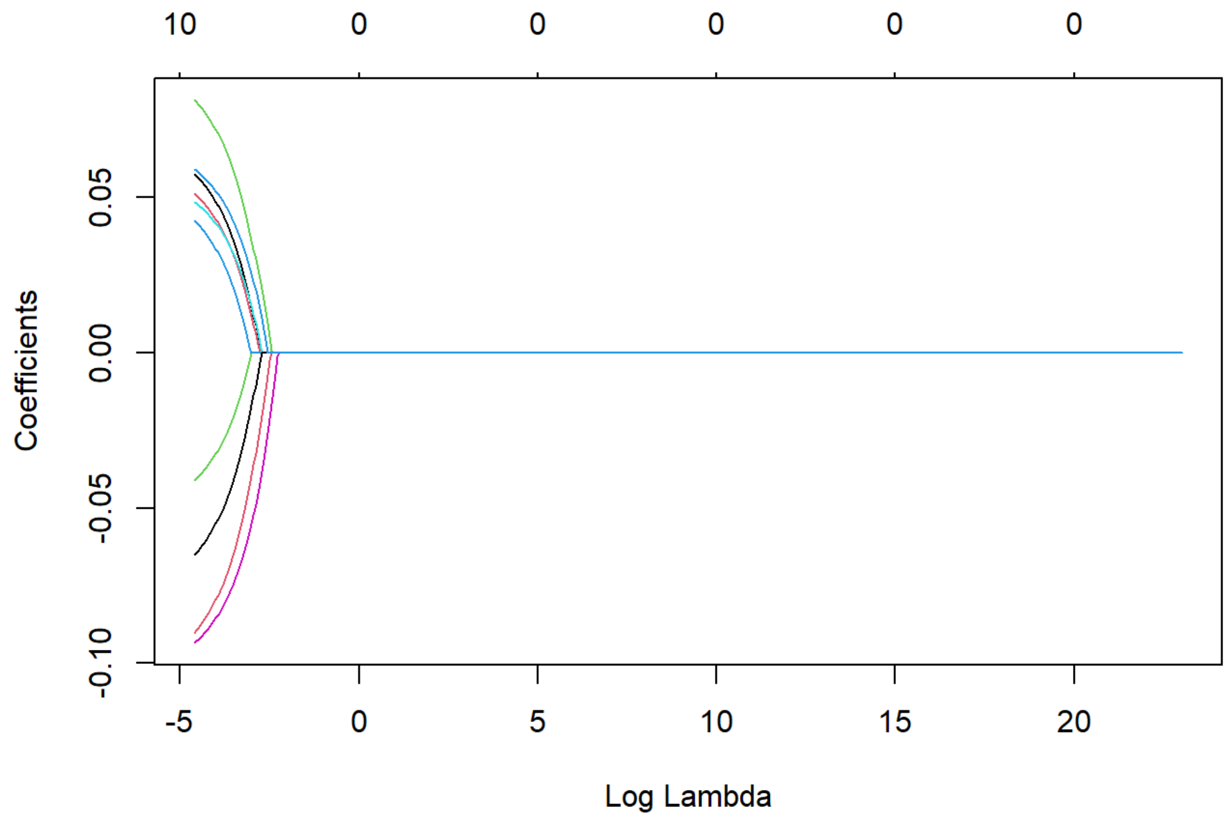
```
lasso.model <- glmnet(X, Y, alpha = 1, lambda = grid)
cv.out <- cv.glmnet(X, Y, alpha = 1)
plot(cv.out)
```



```
best_lambda_lasso <- cv.out$lambda.min
best_lambda_lasso
```

```
## [1] 0.0004699461
```

```
plot(lasso.model, xvar="lambda")
```



```
y_hat <- predict(lasso.model, s = best_lambda_lasso, newx = X)
MSE <- mean((y_hat - Y)^2)
MSE
```

```
## [1] 0.8673004
```

In summary, we can see that the optimal value for λ for the lasso model is much smaller at, 0.0017, which makes sense because we expect the tuning parameter to be very close to 0 for lasso regression. This is also shown in the graph with the lasso regression coefficients, where we can see very small values for λ. Finally, we can see that the error from the lassso regression model is about 0.867, which is a better than both the ridge and linear models.

```
coef(cv.out, s=best_lambda_lasso)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept) -0.02401335
## X4           0.06798709
## X12          0.06074940
## X17         -0.05084662
## X18          0.05305464
## X19          0.05665437
## X22         -0.10257986
## X23         -0.07663269
## X26         -0.10223603
## X27          0.09230811
## X30          0.06727056
```

From the lasso regression, all of the features I included from forward selection were determined to be important, a non-zero estimated coefficient.