Homework 5 - Predictive Modeling in Finance and Insurance

Dennis Goldenberg

2024-02-19

```
library(MASS, quietly = TRUE)
library(ggplot2)
library(ISLR)
suppressPackageStartupMessages(library(glmnet, quietly = TRUE))
suppressPackageStartupMessages(library(dplyr, quietly = TRUE))
suppressPackageStartupMessages(library(reshape, quietly = TRUE))
```

1. Cross Validation

1a. Compute LOOCV

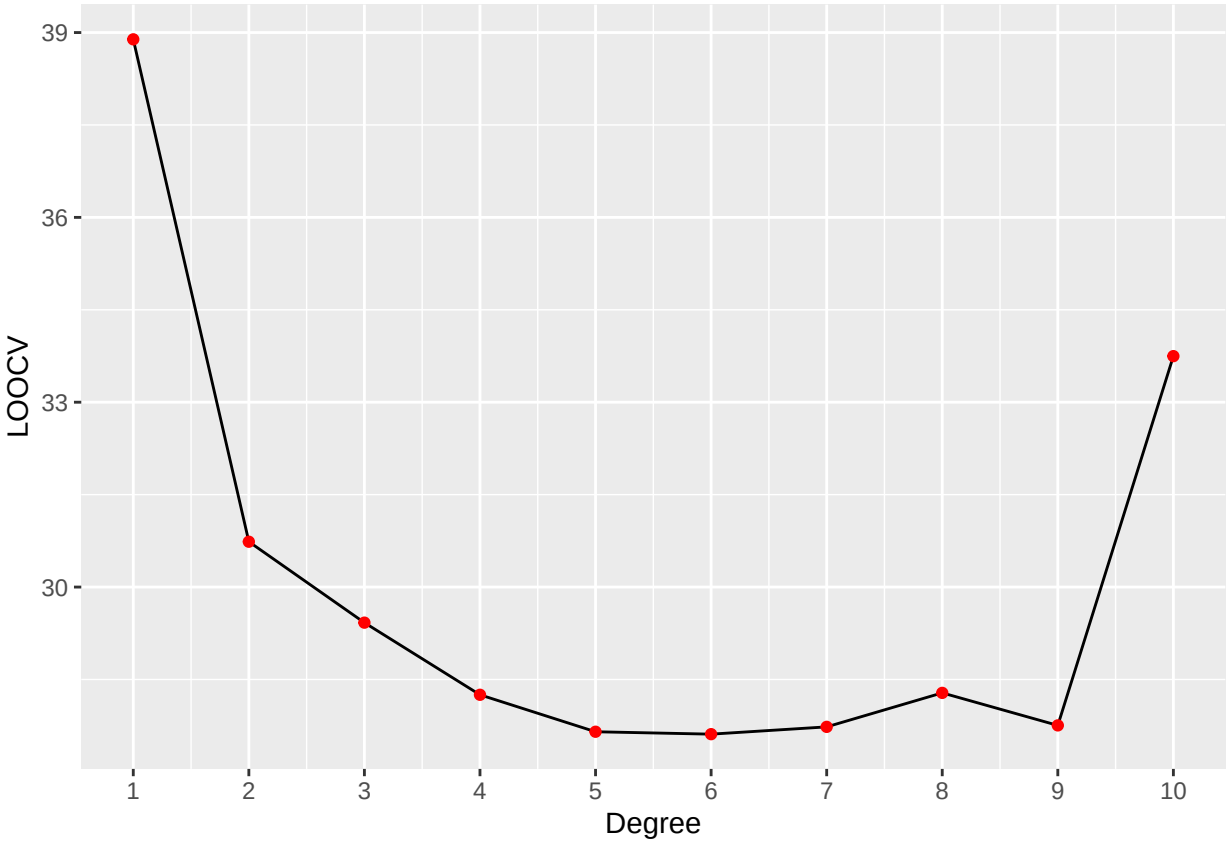
I use the method prescribed in the homework, noting that the default for the cross validation method is Leave-One-Out Cross validation and show the result:

```
LOOCV <- c()
for (i in 1:10){
  polyLCV <- glm(sprintf('medv ~ poly(lstat, %f)', i), data = Boston)
  LOOCV <- append(LOOCV,boot::cv.glm(data = Boston,glmfit = polyLCV)$delta[2])
}
LCVFrame <- data.frame(cbind(1:10, LOOCV))
colnames(LCVFrame) <- c("Degree", "LOOCV")
LCVFrame
```

```
##    Degree    LOOCV
## 1       1 38.88969
## 2       2 30.73581
## 3       3 29.42207
## 4       4 28.25120
## 5       5 27.65137
## 6       6 27.61226
## 7       7 27.73070
## 8       8 28.28218
## 9       9 27.75437
## 10     10 33.74790
```

I graph the LOOCV plot:

```
ggplot(data = LCVFrame) + geom_line(aes(x = Degree,y = LOOCV)) +
  geom_point(aes(x = Degree, y = LOOCV), color = 'red') +
  scale_x_continuous(breaks = seq(1,10, by = 1))
```



I find the degree of polynomial with the minimum error:

```
sprintf("Polynomial Degree with smallest Test MSE: %.0f",
        which.min(LOOCV))
```

```
## [1] "Polynomial Degree with smallest Test MSE: 6"
```

The **Polynmoial Degree 6** model is chosen.

1b. Compute 10-fold CV

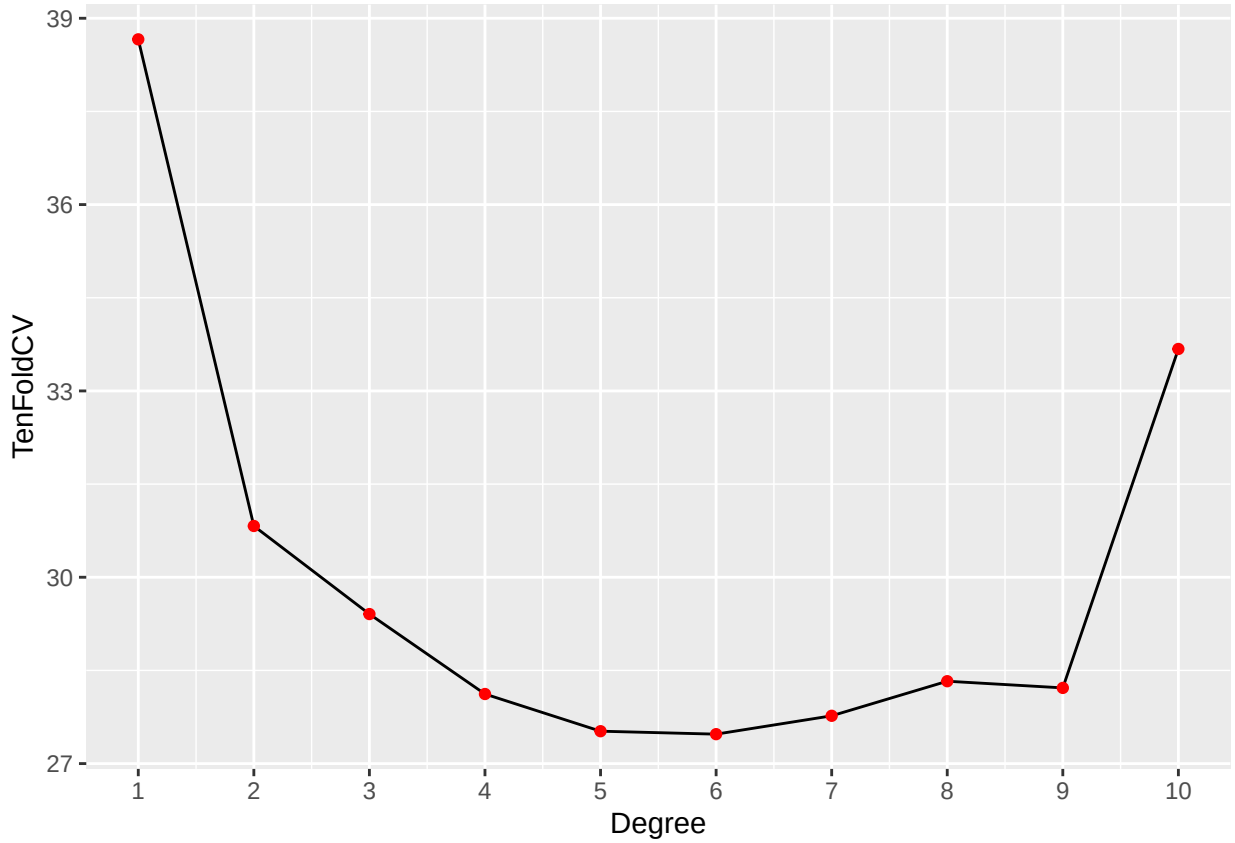
I do the same thing, but set $K = 10$:

```
set.seed(15)
TCV <- c()
for (i in 1:10){
  polyTCV <- glm(sprintf('medv ~ poly(lstat, %f)', i), data = Boston)
  TCV <- append(TCV,boot::cv.glm(data = Boston,glmfit = polyTCV,
                                 K = 10)$delta[2])
}
TCVFrame <- data.frame(cbind(1:10, TCV))
colnames(TCVFrame) <- c("Degree", "TenFoldCV")
TCVFrame
```

```
##    Degree TenFoldCV
## 1       1  38.66043
## 2       2  30.82544
## 3       3  29.40752
## 4       4  28.12013
## 5       5  27.52136
## 6       6  27.47382
## 7       7  27.76949
## 8       8  28.32627
## 9       9  28.21876
## 10     10  33.67553
```

I graph the TCV plot:

```
ggplot(data = TCVFrame) + geom_line(aes(x = Degree,y = TenFoldCV)) +
  geom_point(aes(x = Degree, y = TenFoldCV), color = 'red') +
  scale_x_continuous(breaks = seq(1,10, by = 1))
```



I find the degree of polynomial with the minimum error:

```
sprintf("Polynomial Degree with smallest Test MSE: %.0f",
        which.min(TCV))
```

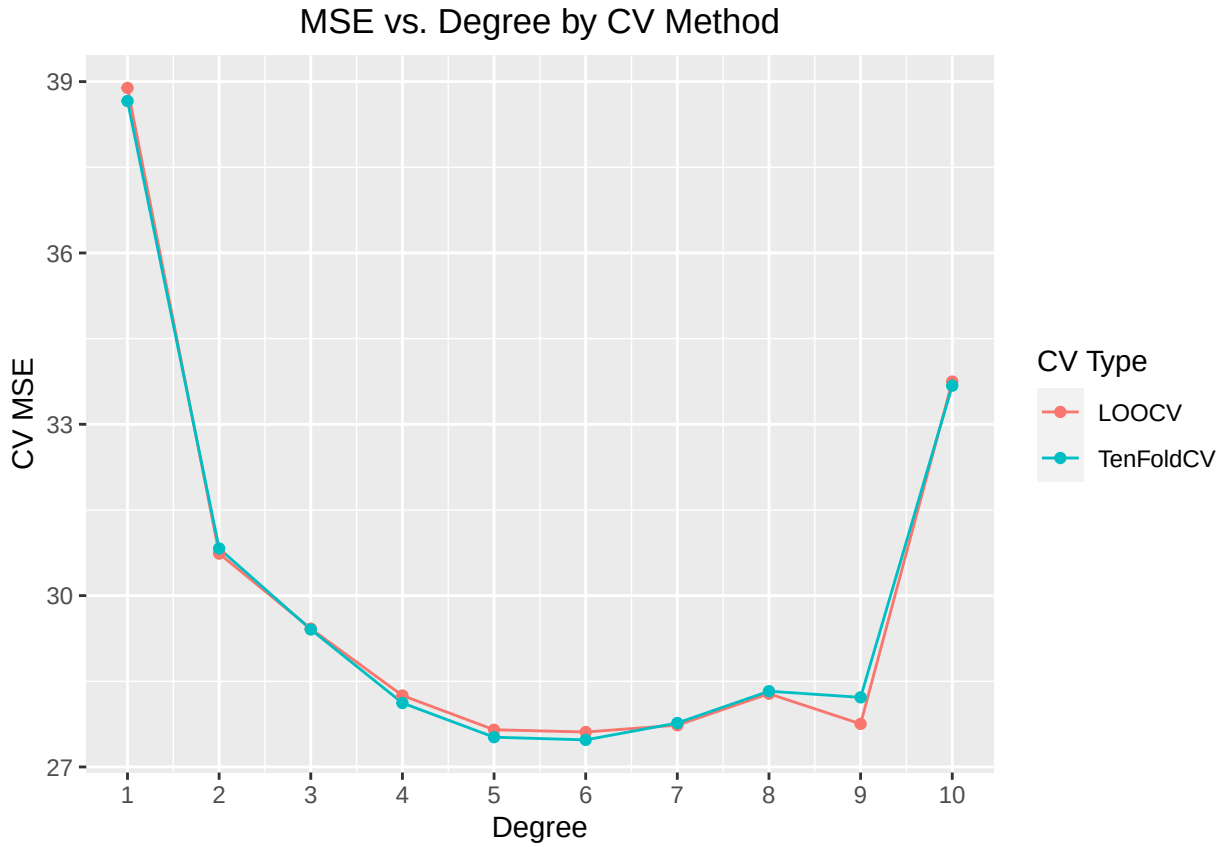
```
## [1] "Polynomial Degree with smallest Test MSE: 6"
```

The **Polynmoial Degree 6** model is chosen.

1c. Comparison of 10-fold CV, LOOCV

I graph both MSE lines atop each other:

```
cFrame <- left_join(LCVFrame, TCVFrame, by=c('Degree'))
cFrame <- melt.data.frame(cFrame, id.vars = 'Degree', variable_name = "CV Type")
ggplot(data = cFrame) +
  geom_line(aes(x = Degree,y = value, color = `CV Type`)) +
  geom_point(aes(x = Degree, y = value, color = `CV Type`)) +
  scale_x_continuous(breaks = seq(1,10, by = 1)) +
  labs(title = "MSE vs. Degree by CV Method", y = "CV MSE") +
  theme(plot.title = element_text(hjust = 0.5))
```



The two methods for MSE follow very similar patterns, with the 10-Fold CV having a slightly lower MSE for most polynomial degrees. This is due to LOOCV having slightly higher variance due to the potential for overfitting caused by using all but one observation in the training data set.

2. Shrinkage Method - Ridge Regression

2a. Varying Training Set Size

I find the optimal tuning parameter and minimum MSE for each of the different proportions for the training set:

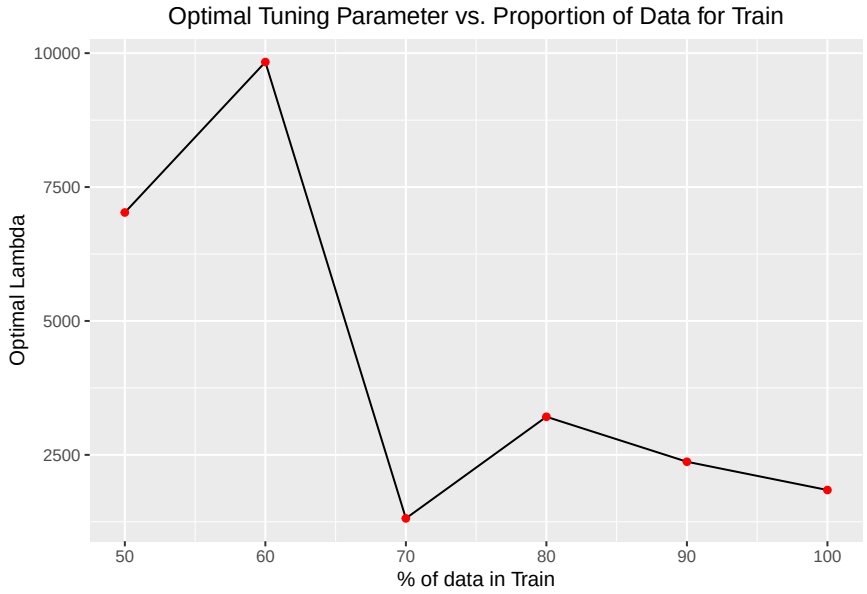
```
set.seed(1)
train_p <- seq(.5, 1, by = .1)

#get all non-NA salary data
nonNAIndices <- which(!is.na(Hitters$Salary))
data_Hit <- Hitters[nonNAIndices,]
x <- model.matrix(Salary ~., data = data_Hit)
y <- data_Hit$Salary
lambdas <- c()
MSE <- c()

#find optimal lambda, MSE for for each training proportion
for (prop in train_p){
  train_samp <- sample(1:length(nonNAIndices),
                       size = as.integer(prop * length(rownames(data_Hit))),
                       replace = F)
  test_samp <- setdiff(1:length(nonNAIndices), train_samp)
  optMod <- cv.glmnet(x[train_samp,],y[train_samp],type.measure =
                      "mse",alpha = 0,family="gaussian")
  lambdas <- append(lambdas, optMod$lambda.1se)
  if (prop < 1){
  predictions <- predict(optMod, newx = x[test_samp,], s = optMod$lambda.1se)
  MSE <- append(MSE,sum((predictions - y[test_samp])^2)/(length(y[test_samp])))
  } else {
    MSE <- append(MSE, 0)
  }
}
```

I plot lambdas vs. proportion of data set:

```
lFrame <- cbind.data.frame(round(train_p * 100, 0), lambdas)
names(lFrame) <- c("% of data in Train", "Optimal Lambda")
ggplot(data = lFrame) +
  geom_line(aes(x = `% of data in Train`, y = `Optimal Lambda`)) +
  geom_point(aes(x = `% of data in Train`, y = `Optimal Lambda`), color='red') +
  labs(title = "Optimal Tuning Parameter vs. Proportion of Data for Train") +
  theme(plot.title = element_text(hjust = 0.5))
```



Then, I plot MSE vs. proportion of dataset:

```
lFrameMSE <- cbind.data.frame(round(train_p * 100, 0), MSE)
names(lFrameMSE) <- c("% of data in Train", "MSE")
ggplot(data = lFrameMSE) +
  geom_line(aes(x = `% of data in Train`, y = MSE)) +
  geom_point(aes(x = `% of data in Train`, y = MSE), color='red') +
  labs(title = "Mean Squared Error vs. Proportion of Data for Train") +
  theme(plot.title = element_text(hjust = 0.5))
```

It seems as though using **70%** of the data seems to minimize test Mean Squared error (100% train data cannot be considered, as no test MSE can be calculated when there is no test data).

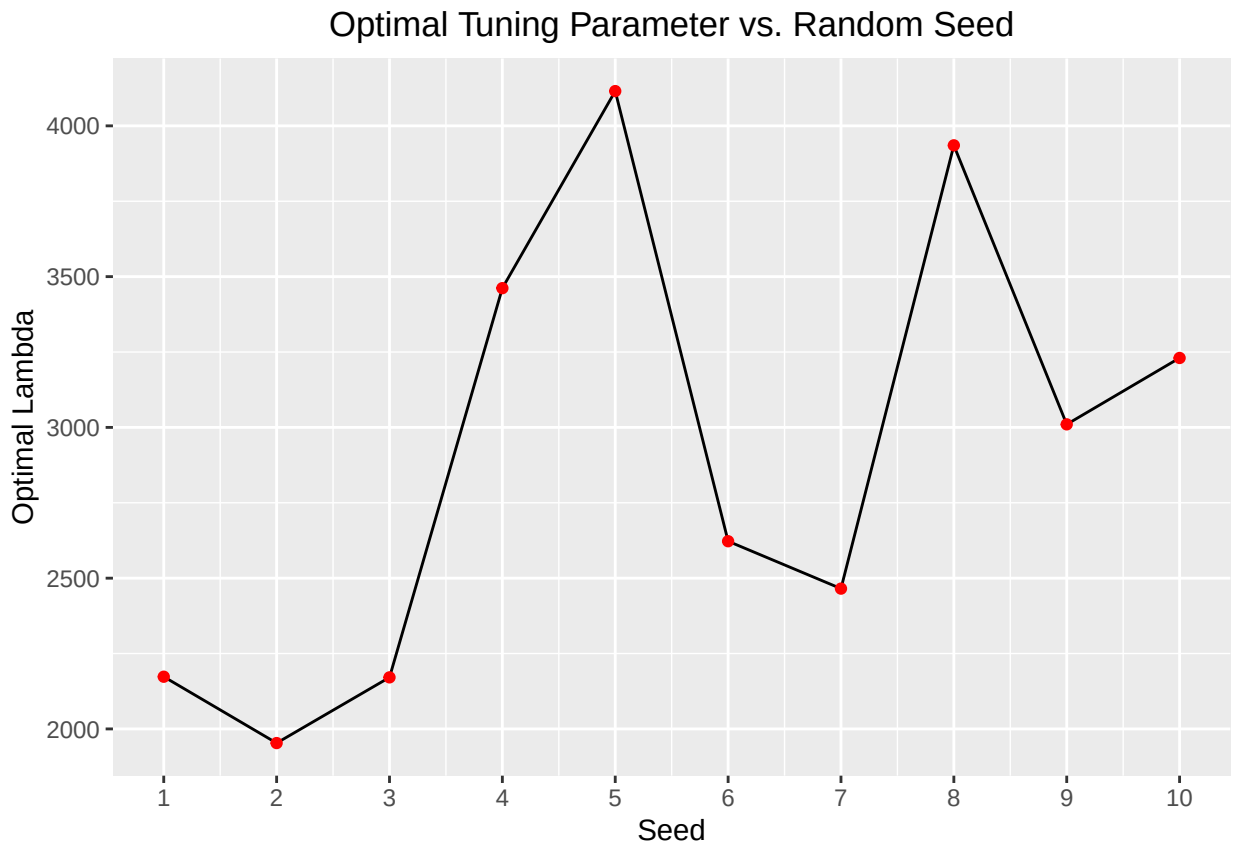
2b. Varying Random Seed

I run essentially the same method as above, but this time I vary the seed:

```
lambdas2 <- c()
for (j in 1:10){
  set.seed(j)
  train_samp2 <- sample(1:length(nonNAIndices),
                        size = as.integer(0.85 * length(rownames(data_Hit))),
                        replace = F)
  set.seed(1)
  optMod2 <- cv.glmnet(x[train_samp2,],y[train_samp2],type.measure =
                       "mse",alpha = 0,family="gaussian")
  lambdas2 <- append(lambdas2, optMod2$lambda.1se)
}
```

Now I plot the optimal lambda value against the seed values:

```
lFrame2 <- cbind.data.frame(1:10, lambdas2)
names(lFrame2) <- c("Seed", "Optimal Lambda")
ggplot(data = lFrame2) +
  geom_line(aes(x = Seed, y = `Optimal Lambda`)) +
  geom_point(aes(x = Seed, y = `Optimal Lambda`), color='red') +
  scale_x_continuous(breaks = seq(1,10, by = 1)) +
  labs(title = "Optimal Tuning Parameter vs. Random Seed") +
  theme(plot.title = element_text(hjust = 0.5))
```



2c. Takeaways, Observations From Above

From the graphs plotting optimal lambdas against training proportion and seed, it seems as though proportion of training data has a noticeable impact on optimal lambda, while seed does not. As the proportion of data being trained on gets higher, there seems to be a downward trend in optimal lambda; a reason for this is that there is more data to glean which variables are truly important and which aren't, so higher betas on the important variables decrease the error function by enough to compensate for the penalty. However, the fact that there is no significant trend with respect to the seed may be the result of the fact that 10-fold cross validation is being used; there is significant similarity between splits of the data under different random seeds, so the Cross Validation error calculations for each seed are likely very similar.

3. Shrinkage Method

3a. Calculating the MSE errors for different alphas

I find the MSE under different alphas:

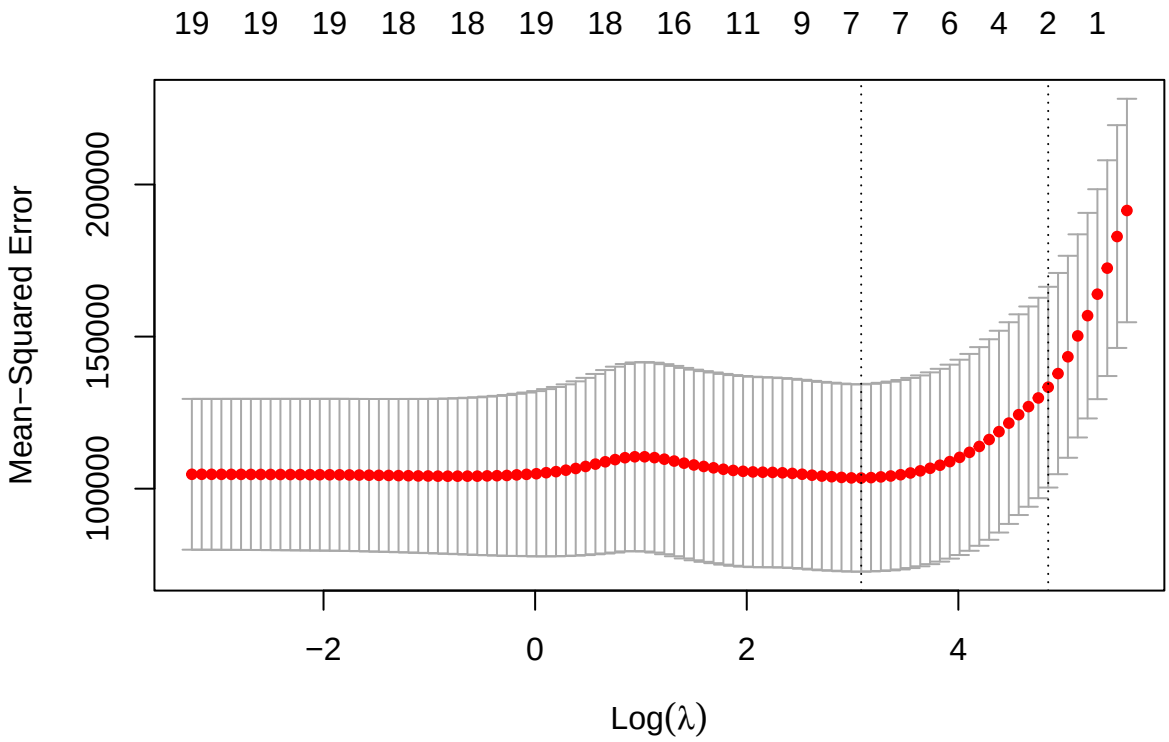
```
set.seed(1)
train_samp3 <- sample(1:length(nonNAIndices),
                      size = as.integer(2/3 * length(rownames(data_Hit))),
                      replace = F)
test_samp3 <- setdiff(1:length(nonNAIndices), train_samp3)
alphas <- seq(0,1, by = 1/20)
MSE3 <- c()
for (a in alphas){
  optMod3 <- cv.glmnet(x[train_samp3,],y[train_samp3],type.measure =
                       "mse",alpha = a,family="gaussian")
  predictions3 <- predict(optMod3, newx = x[test_samp3,], s= optMod3$lambda.1se)
  MSE3 <- append(MSE3,
                 sum((predictions3 - y[test_samp3])^2)/(length(y[test_samp3])))
}
alphaFrame <- cbind.data.frame(alphas, MSE3)
names(alphaFrame) <- c("alpha", "MSE")
alphaFrame
```

```
##    alpha      MSE
## 1   0.00 150740.8
## 2   0.05 156274.6
## 3   0.10 164116.4
## 4   0.15 153694.4
## 5   0.20 155879.4
## 6   0.25 149662.9
## 7   0.30 158197.0
## 8   0.35 157770.2
## 9   0.40 161130.6
## 10  0.45 150621.0
## 11  0.50 160051.9
## 12  0.55 146710.8
## 13  0.60 154175.7
## 14  0.65 149011.5
## 15  0.70 149243.3
## 16  0.75 149410.1
## 17  0.80 151939.9
## 18  0.85 159223.7
## 19  0.90 159357.4
## 20  0.95 159018.0
## 21  1.00 161382.9
```

3b. Plotting fit when alpha = 1

I plot the final fit of my for loop, which uses $\alpha = 1$:

```
plot(optMod3)
```



3c. I find the smallest MSE

I find the alpha with the smallest MSE, and said smallest MSE:

```
sprintf("Alpha with smallest MSE: %.2f",
        alphaFrame$alpha[which.min(alphaFrame$MSE)])
```

```
## [1] "Alpha with smallest MSE: 0.55"
```

```
sprintf("Minimum MSE: %.2f",min(alphaFrame$MSE))
```

```
## [1] "Minimum MSE: 146710.84"
```