Gov 2018: Lab 4 Cross Validation

Adeline Lo

Tuesday February 15, 2022

This lab on Ridge Regression and the Lasso in R is based off of p. 251-255 of "Introduction to Statistical Learning with Applications in R" by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani.

CV Ridge Regression and the Lasso

```
rm(list=ls())
library(ISLR)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-3
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
#set seed
lab.seed<-202202
```

Use the `glmnet` package in order to perform ridge regression and the lasso. The main function in this package is `glmnet()`, which can be used to fit ridge regression models, lasso models, and more.

Load and remove `NA`s.

```
Hitters = na.omit(Hitters)
```

Execute a ridge regression and the lasso in order to predict `Salary` on the `Hitters` data.

Set up data:

```
x = model.matrix(Salary~., Hitters)[,-1] # trim off the first column
                                         # leaving only the predictors
y = Hitters$Salary
```

The `model.matrix()` function is particularly useful for creating x; not only does it produce a matrix corresponding to the 19 predictors but it also automatically transforms any qualitative variables into dummy variables. The latter property is important because `glmnet()` can only take numerical, quantitative inputs.

Question 1. Ridge Regression

The `glmnet()` function has an alpha argument that determines what type of model is fit. If `alpha = 0` then a ridge regression model is fit, and if `alpha = 1` then a lasso model is fit. Fit a ridge regression model on `x` and `y` using the grid of lambda values from below.

```
grid = 10^seq(10, -2, length = 100)
# your.model.object <- glmnet(...)
# x, y, alpha, lambda
model <- glmnet(x, y, alpha = 0, lambda = grid)

print(model)
```

```
##
## Call:  glmnet(x = x, y = y, alpha = 0, lambda = grid)
##
##      Df  %Dev    Lambda
## 1    19  0.00 1.000e+10
## 2    19  0.00 7.565e+09
## 3    19  0.00 5.722e+09
## 4    19  0.00 4.329e+09
## 5    19  0.00 3.275e+09
## 6    19  0.00 2.477e+09
## 7    19  0.00 1.874e+09
## 8    19  0.00 1.417e+09
## 9    19  0.00 1.072e+09
## 10   19  0.00 8.111e+08
## 11   19  0.00 6.136e+08
## 12   19  0.00 4.642e+08
## 13   19  0.00 3.511e+08
## 14   19  0.00 2.656e+08
## 15   19  0.00 2.009e+08
## 16   19  0.00 1.520e+08
## 17   19  0.00 1.150e+08
## 18   19  0.00 8.697e+07
## 19   19  0.00 6.579e+07
## 20   19  0.01 4.977e+07
## 21   19  0.01 3.765e+07
## 22   19  0.01 2.848e+07
## 23   19  0.01 2.154e+07
## 24   19  0.02 1.630e+07
## 25   19  0.02 1.233e+07
## 26   19  0.03 9.326e+06
## 27   19  0.04 7.055e+06
## 28   19  0.05 5.337e+06
```

```
## 29  19  0.07 4.037e+06
## 30  19  0.09 3.054e+06
## 31  19  0.12 2.310e+06
## 32  19  0.16 1.748e+06
## 33  19  0.21 1.322e+06
## 34  19  0.27 1.000e+06
## 35  19  0.36 7.565e+05
## 36  19  0.48 5.722e+05
## 37  19  0.63 4.329e+05
## 38  19  0.83 3.275e+05
## 39  19  1.09 2.477e+05
## 40  19  1.43 1.874e+05
## 41  19  1.88 1.417e+05
## 42  19  2.46 1.072e+05
## 43  19  3.21 8.111e+04
## 44  19  4.16 6.136e+04
## 45  19  5.38 4.642e+04
## 46  19  6.90 3.511e+04
## 47  19  8.77 2.656e+04
## 48  19 11.03 2.009e+04
## 49  19 13.70 1.520e+04
## 50  19 16.75 1.150e+04
## 51  19 20.11 8.697e+03
## 52  19 23.69 6.579e+03
## 53  19 27.31 4.977e+03
## 54  19 30.81 3.765e+03
## 55  19 34.04 2.848e+03
## 56  19 36.88 2.154e+03
## 57  19 39.28 1.630e+03
## 58  19 41.25 1.233e+03
## 59  19 42.84 9.330e+02
## 60  19 44.12 7.060e+02
## 61  19 45.16 5.340e+02
## 62  19 46.01 4.040e+02
## 63  19 46.73 3.050e+02
## 64  19 47.37 2.310e+02
## 65  19 47.95 1.750e+02
## 66  19 48.50 1.320e+02
## 67  19 49.04 1.000e+02
## 68  19 49.57 7.600e+01
## 69  19 50.11 5.700e+01
## 70  19 50.66 4.300e+01
## 71  19 51.19 3.300e+01
## 72  19 51.71 2.500e+01
## 73  19 52.19 1.900e+01
## 74  19 52.63 1.400e+01
## 75  19 53.03 1.100e+01
## 76  19 53.36 8.000e+00
## 77  19 53.65 6.000e+00
## 78  19 53.88 5.000e+00
## 79  19 54.07 4.000e+00
## 80  19 54.21 3.000e+00
## 81  19 54.32 2.000e+00
## 82  19 54.41 2.000e+00
```

```
## 83  19 54.47 1.000e+00
## 84  19 54.51 1.000e+00
## 85  19 54.55 1.000e+00
## 86  19 54.57 0.000e+00
## 87  19 54.58 0.000e+00
## 88  19 54.59 0.000e+00
## 89  19 54.60 0.000e+00
## 90  19 54.60 0.000e+00
## 91  19 54.60 0.000e+00
## 92  19 54.60 0.000e+00
## 93  19 54.61 0.000e+00
## 94  19 54.61 0.000e+00
## 95  19 54.61 0.000e+00
## 96  19 54.61 0.000e+00
## 97  19 54.61 0.000e+00
## 98  19 54.61 0.000e+00
## 99  19 54.61 0.000e+00
## 100 19 54.61 0.000e+00
```

By default the `glmnet()` function performs ridge regression for an automatically selected range of λ values. However, here we have chosen to implement the function over a grid of values ranging from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit.

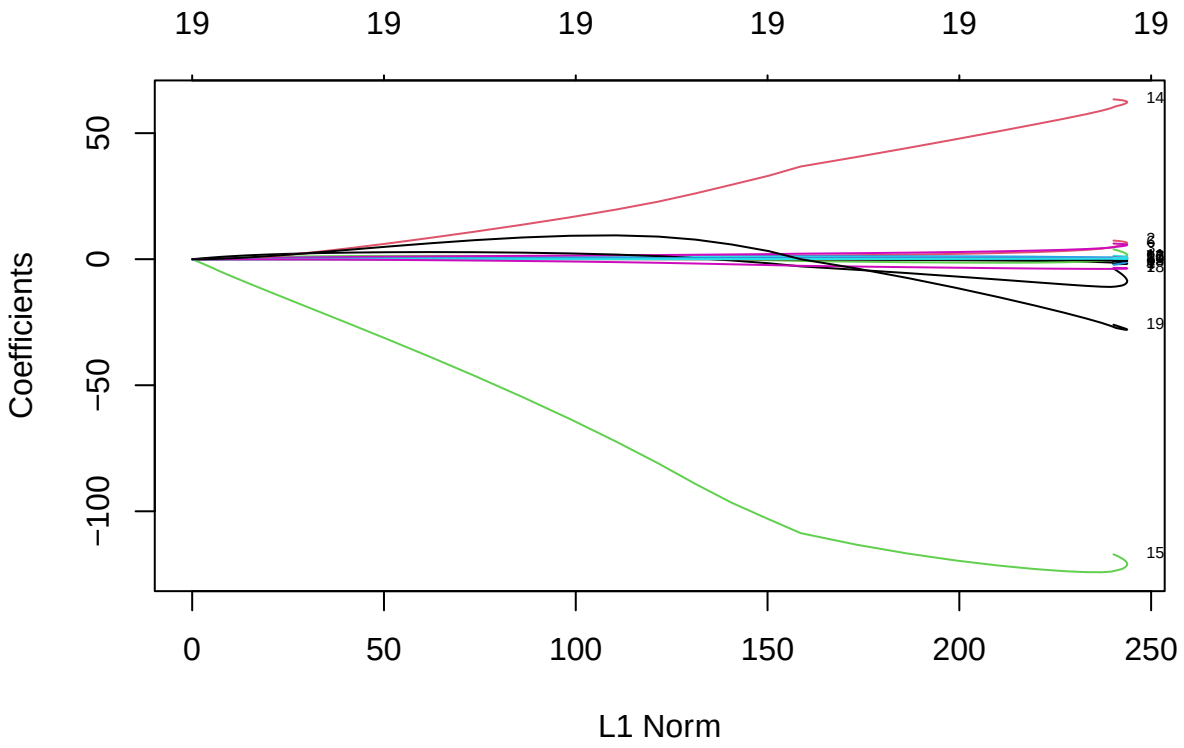
As we will see, we can also compute model fits for a particular value of λ that is not one of the original grid values. Note that by default, the `glmnet()` function standardizes the variables so that they are on the same scale. To turn off this default setting, use the argument `standardize = FALSE`.

Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by `coef()`. In this case, it is a 20×100 matrix, with 20 rows (one for each predictor, plus an intercept) and 100 columns (one for each value of λ). Plot your coefficients from the ridge regression output with `plot(your.model.object)`.

```
# dim(coef(your.model.object))
# plot(your.model.object)

dim(coef(model))
```

```
## [1]  20 100
```

```
plot(model, label = TRUE)
```



We expect the coefficient estimates to be much smaller, in terms of l_2 norm, when a large value of λ is used, as compared to when a small value of λ is used. Set λ to its 50th value. What are the coefficients at this value? What's their l_2 norm (remove the intercept value)?

```
# Hint: use coef()

# 50th lambda
model$lambda[50]
```

```
## [1] 11497.57
```

```
# coefficients for 50th lambdas
coef(model, s = grid[50])
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) 407.356050200
## AtBat         0.036957182
## Hits          0.138180344
## HmRun         0.524629976
## Runs          0.230701523
## RBI           0.239841459
## Walks         0.289618741
## Years         1.107702929
## CAtBat        0.003131815
## CHits         0.011653637
## CHmRun        0.087545670
## CRuns         0.023379882
## CRBI          0.024138320
## CWalks        0.025015421
## LeagueN       0.085028114
## DivisionW    -6.215440973
```

```
## PutOuts       0.016482577
## Assists       0.002612988
## Errors       -0.020502690
## NewLeagueN    0.301433531
```

```
# create a vector so I can make l2
vector <- coef(model, s = grid[50])[-1]
sqrt(sum(vector^2))
```

```
## [1] 6.360612
```

In contrast, what are the coefficients when λ is at its 60th value? Their l_2 norm? Note the much larger l_2 norm of the coefficients associated with this smaller value of λ.

```
# 60th lambda
model$lambda[60]
```

```
## [1] 705.4802
```

```
# coefficients for 60th lambdas
coef(model, s = grid[60])
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  54.32519950
## AtBat         0.11211115
## Hits          0.65622409
## HmRun         1.17980910
## Runs          0.93769713
## RBI           0.84718546
## Walks         1.31987948
## Years         2.59640425
## CAtBat        0.01083413
## CHits         0.04674557
## CHmRun        0.33777318
## CRuns         0.09355528
## CRBI          0.09780402
## CWalks        0.07189612
## LeagueN      13.68370191
## DivisionW   -54.65877750
## PutOuts       0.11852289
## Assists       0.01606037
## Errors       -0.70358655
## NewLeagueN    8.61181213
```

```
# create a vector so I can make l2
vector2 <- coef(model, s = grid[60])[-1]
sqrt(sum(vector2^2))
```

```
## [1] 57.11001
```

Split the samples into a 80% training set and a 20% test set in order to estimate the test error of ridge regression and the lasso.

```
set.seed(lab.seed)

# randomly sample so I can slice data
data <- sample_frac(Hitters, 1L)
```

```
# create train and test data
train <- slice(data, 1:210)
test <- slice(data, 211:263)

# create x and y values for training and test data
x_train = model.matrix(Salary~., train)[,-1]
x_test = model.matrix(Salary~., test)[,-1]
y_train = train$Salary
y_test = test$Salary
```

Next fit a ridge regression model on the training set, and evaluate its MSE (mean squared error) on the test set, using $\lambda = 40$. Note the use of the `predict()` function again: get predictions for a test set making sure to use`newx` argument.

```
set.seed(lab.seed)

# new model
ridge2 <- glmnet(x_train, y_train, alpha = 0, lambda = grid, thresh = 1e-12)

# create prediction
pred <- predict(ridge2, s = 40, newx = x_test)

# calculate mse
mse1 <- mean((pred - y_test)^2)
mse1
```

```
## [1] 58860.58
```

The test MSE is 5.8860581×10^4.

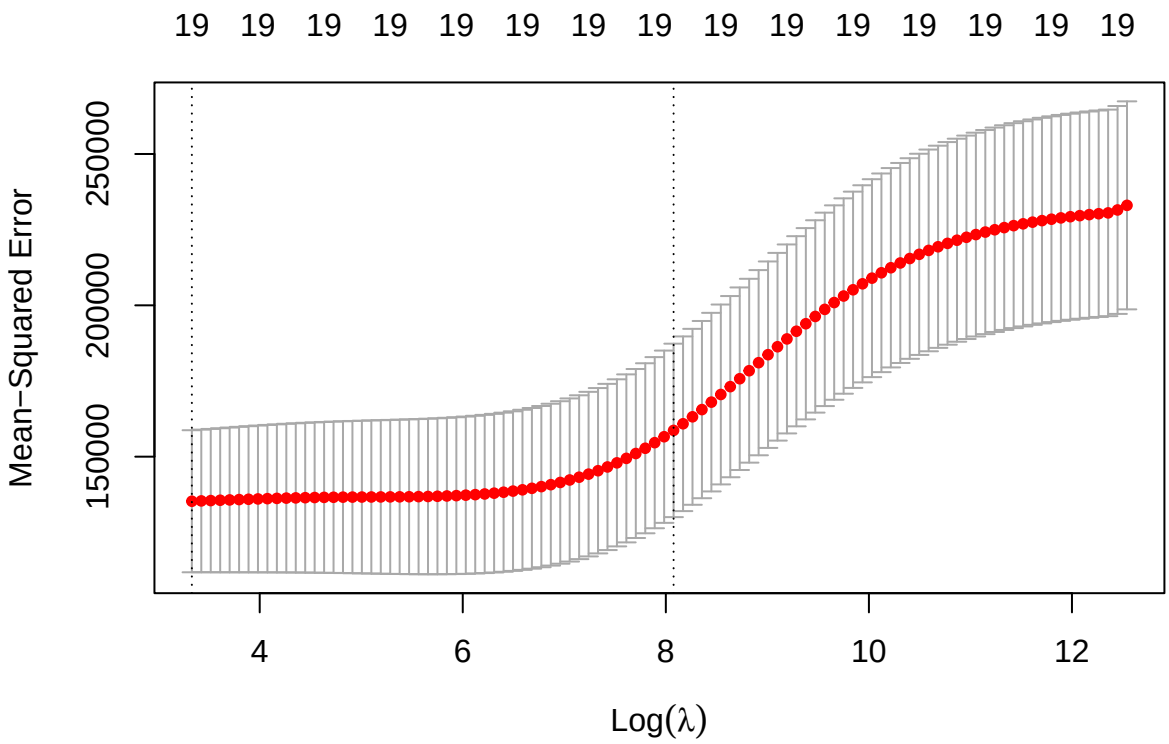
Instead of arbitrarily choosing $\lambda = 40$, it would be better to use cross-validation to choose the tuning parameter λ. We can do this using the built-in cross-validation function, `cv.glmnet()`. By default, the function performs 10-fold cross-validation, though this can be changed using the argument `nfolds`. Set folds to 10 and calculate the λ that best minimizes the training MSE (`lambda.min` item in object returned from `cv.glmnet()`).

```
set.seed(lab.seed)

# Fit ridge regression model on training data
cross_mod = cv.glmnet(x_train, y_train, alpha = 0, nfolds=10)

# find the best lambda value
bestlam = cross_mod$lambda.min
bestlam
```

```
## [1] 27.98773
```

Plot the MSE as a function of λ by using `plot()` on our returned object from our call to `cv.glmnet`.

```
plot(cross_mod)
```



What is the test MSE associated with this value of λ (λ that best minimizes the training MSE)?

```
# do the same thing as before the calculate mse
# create prediction
pred2 <- predict(cross_mod, s = bestlam, newx = x_test)

# calculate mse
mse2 <- mean((pred2 - y_test)^2)
mse2
```

```
## [1] 60795.79
```

The second MSE is smaller, meaning that this lambda value is better.

Refit the ridge regression model on the full data set, using the value of λ chosen by cross-validation, and examine the coefficient estimates.

```
# use all of the data
out = glmnet(x, y, alpha = 0)

# create predictions with full data and chosen lambda value
predict(out, type = "coefficients", s = bestlam)[1:20,]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  7.651347e+01 -6.321105e-01  2.643699e+00 -1.387966e+00  1.045364e+00
##           RBI         Walks         Years        CAtBat         CHits
##  7.313519e-01  3.279191e+00 -8.727792e+00  1.099591e-04  1.319473e-01
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  6.896576e-01  2.831579e-01  2.515565e-01 -2.602097e-01  5.234755e+01
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -1.224219e+02  2.623847e-01  1.629869e-01 -3.644495e+00 -1.703875e+01
```

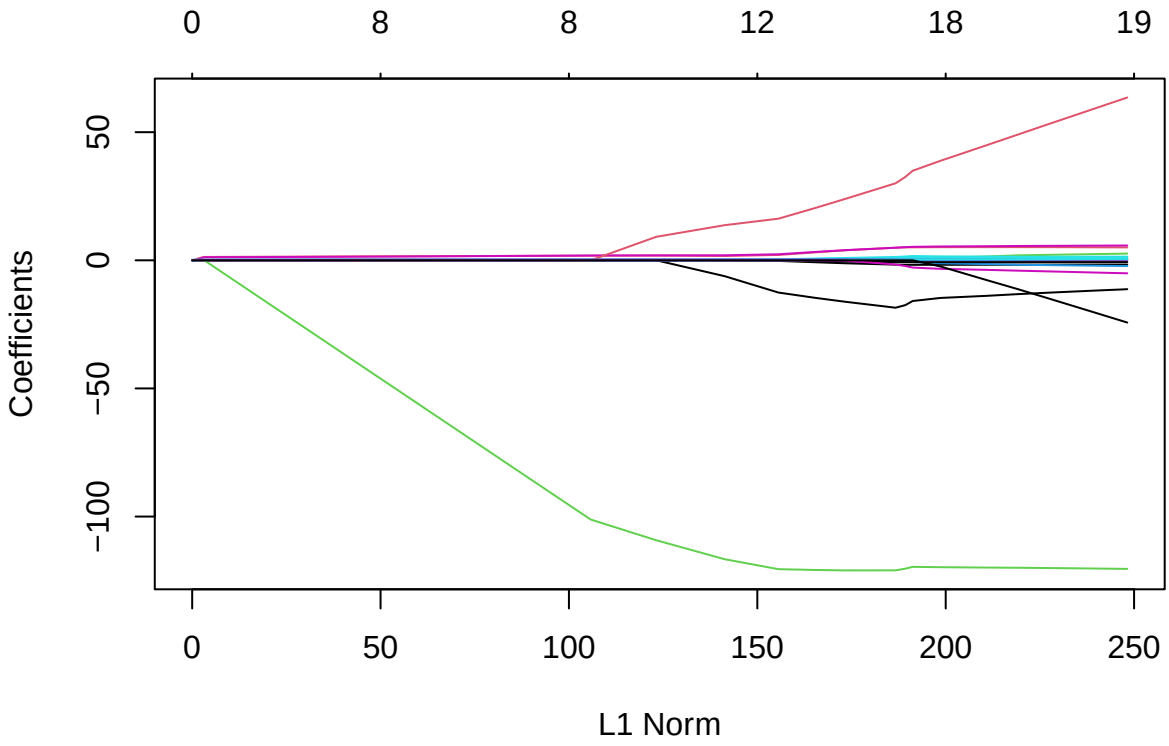
All of the values are very close to zero.

Question 2. The Lasso

You just executed ridge regression with a wise choice of λ. Can lasso yield either a more accurate or a more interpretable model than ridge regression? Fit a lasso model, however, this time use the argument `alpha=1`. Other than that change, proceed just as you did in fitting a ridge model. Plot your model object coefficients.

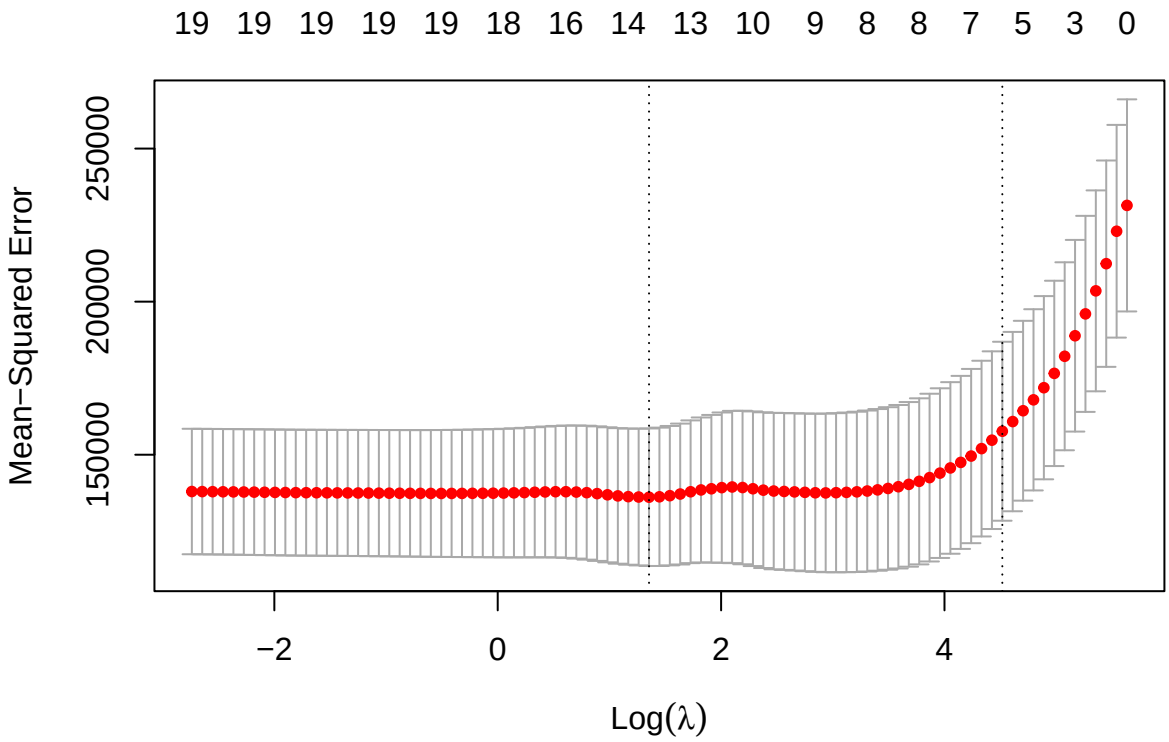
```
set.seed(lab.seed)

# create a lasso model with training data
lasso_mod = glmnet(x_train, y_train, alpha = 1, lambda = grid)

plot(lasso_mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```



Now run the model again, this time performing cross-validation with folds equal to 10. Plot the model object. Then using the lambda that minimizes your cross validated training MSE, compute the associated testing MSE:

```
set.seed(lab.seed)

# fit model with cross validation on training data - the same as was done in earlier question
cv.out = cv.glmnet(x_train, y_train, alpha = 1, nfolds = 10)
# plot model
plot(cv.out)
```




```
# find the best lambda - minimze MSE
bestlam = cv.out$lambda.min

# predict
lasso_pred = predict(lasso_mod, s = bestlam, newx = x_test)

# calculate mse value
mse3 <-mean((lasso_pred - y_test)^2)
mse3
```

```
## [1] 63811.21
```

Question 3. K-fold cross validation

Conduct K-fold cross validation for the ridge and lasso, with $k \in \{5, 7, 9, 11, 13, 15\}$. Assess the models at each k value by calculating the risk/prediction error in the test set and suggest the best k for each. Suggest the best final model.

```
set.seed(lab.seed)

# create k-values
K<-c(5,7,9,11,13,15)

# create a blank data frame
mse <- data.frame(ridge=rep(NA,length(K)),lasso=rep(NA,length(K)))

# create a for loop to run the different k-values
# use the same models as before

for(k in 1:length(K)){
```

```
  cv.lasso = cv.glmnet(x_train, y_train, alpha = 1, nfolds = K[k])
  cv.ridge = cv.glmnet(x_train, y_train, alpha = 0, nfolds = K[k])
  lasso_pred = predict(lasso_mod, s = cv.lasso$lambda.min, newx = x_test)
  ridge_pred = predict(model, s = cv.ridge$lambda.min, newx = x_test) # Use best lambda to predict test
  mse$ridge[k]<-mean((pred - y_test)^2) # Calculate test MSE
  mse$lasso[k]<-mean((pred2 - y_test)^2)
  cat("For k=",K[k],", MSE for ridge is = ", mse$ridge[k],"\n")
  cat("For k=",K[k],", MSE for lasso is = ", mse$lasso[k],"\n")
}
```

```
## For k= 5 , MSE for ridge is =  58860.58
## For k= 5 , MSE for lasso is =  60795.79
## For k= 7 , MSE for ridge is =  58860.58
## For k= 7 , MSE for lasso is =  60795.79
## For k= 9 , MSE for ridge is =  58860.58
## For k= 9 , MSE for lasso is =  60795.79
## For k= 11 , MSE for ridge is =  58860.58
## For k= 11 , MSE for lasso is =  60795.79
## For k= 13 , MSE for ridge is =  58860.58
## For k= 13 , MSE for lasso is =  60795.79
## For k= 15 , MSE for ridge is =  58860.58
## For k= 15 , MSE for lasso is =  60795.79
```

```
cat("Best ridge k is: ",K[which(mse$ridge==min(mse$ridge))],"\n")
```

```
## Best ridge k is:  5 7 9 11 13 15
```

```
cat("Best lasso k is: ",K[which(mse$lasso==min(mse$lasso))],"\n")
```

```
## Best lasso k is:  5 7 9 11 13 15
```

```
min(mse)
```

```
## [1] 58860.58
```

Interestingly, it seems that all of the k values are the best for both ridge and lasso. I don't think this is correct.