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

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

```
## Loading required package: Matrix

## Loaded glmnet 4.1-3
```

```r
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
```

```r
library(tidyr)
```

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

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

```r
#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.

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

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

Set up data:

```r
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.

```r
grid = 10^seq(10, -2, length = 100)

# Defining model with alpha = 0 for ridge, and grid of lambdas
ridge <- glmnet(x, y, lambda = grid, alpha = 0)
```

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.

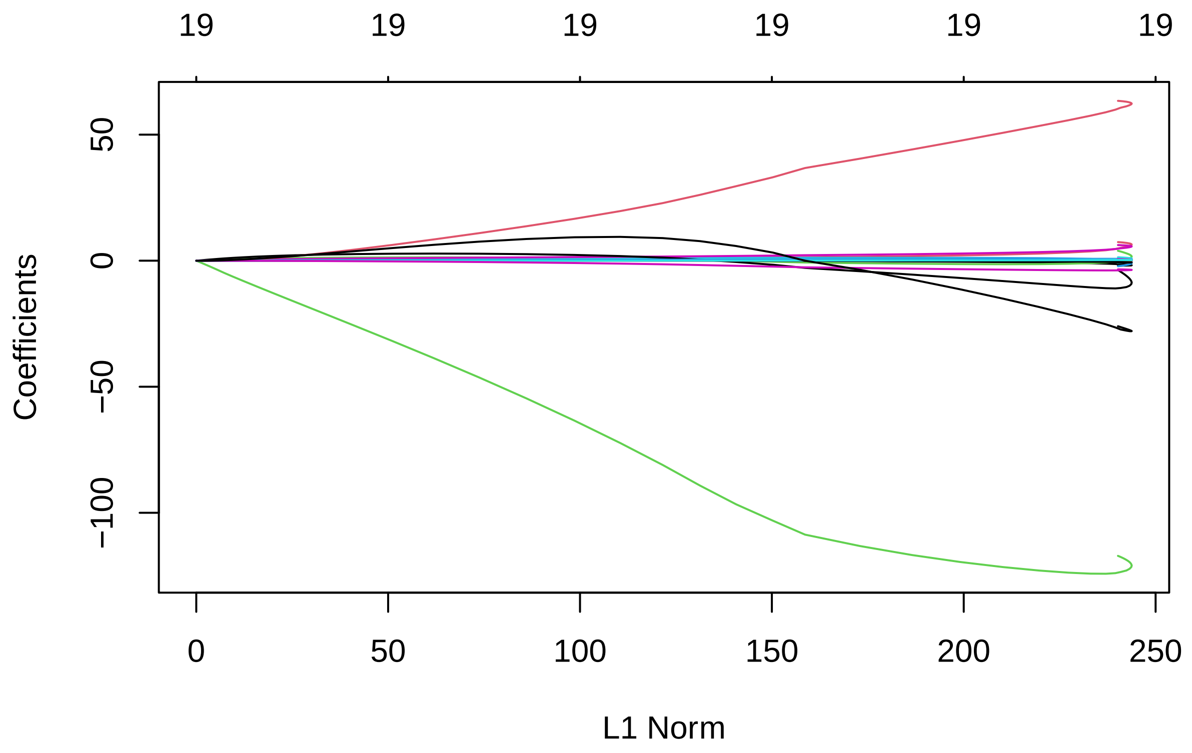
As we will see, we can also compute model fits for a particular value of λ that is not one of the original grid values. Note that by default, the `glmnet()` function standardizes the variables so that they are on the same scale. To turn off this default setting, use the argument `standardize = FALSE`.

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)`.

```r
# Dimensions of the coefs of the ridge model
dim(coef(ridge))
```

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

```r
# Plot of the l1 norm
plot(ridge)
```



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)?

```r
# 50th lambda value
ridge$lambda[50]
```

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

```r
# Coefficients of model at the 60th lambda value
coef(ridge)[,50]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs 
## 407.356050200   0.036957182   0.138180344   0.524629976   0.230701523 
##           RBI         Walks         Years        CAtBat         CHits 
##   0.239841459   0.289618741   1.107702929   0.003131815   0.011653637 
##        CHmRun         CRuns          CRBI        CWalks       LeagueN 
##   0.087545670   0.023379882   0.024138320   0.025015421   0.085028114 
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN 
##  -6.215440973   0.016482577   0.002612988  -0.020502690   0.301433531
```

```r
# Coefficients at 50 with the intercept removed
coefs <- coef(ridge)[-1,50]

 # Calculate l2 norm
sqrt(sum(coefs^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.

```r
# 60th lambda value
ridge$lambda[60]
```

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

```r
# Coefficients to 60 with the intercept removed
coef(ridge)[,60]
```

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

```r
# Coefficients at 60 with the intercept removed
coefs <- coef(ridge)[-1,60]

 # Calculate l2 norm
sqrt(sum(coefs^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.

```r
set.seed(lab.seed)

# 80% of the data to train
train = sample_frac(Hitters, 0.8)

# 20% to test
test = setdiff(Hitters, train)

# Making X's using same function as before
x_train = model.matrix(Salary~., train)[,-1]
x_test = model.matrix(Salary~., test)[,-1]

# Making Y's
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.

```r
set.seed(lab.seed)

# Defining ridge regression with lambda = 40
ridge = glmnet(x_train, y_train, lambda = grid, alpha = 0, thresh = 1e-12)

# Predicting on the test set
ridge_pred = predict(ridge, s = 40, newx = x_test)

# Calculating MSE
mean((ridge_pred - y_test)^2)
```

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

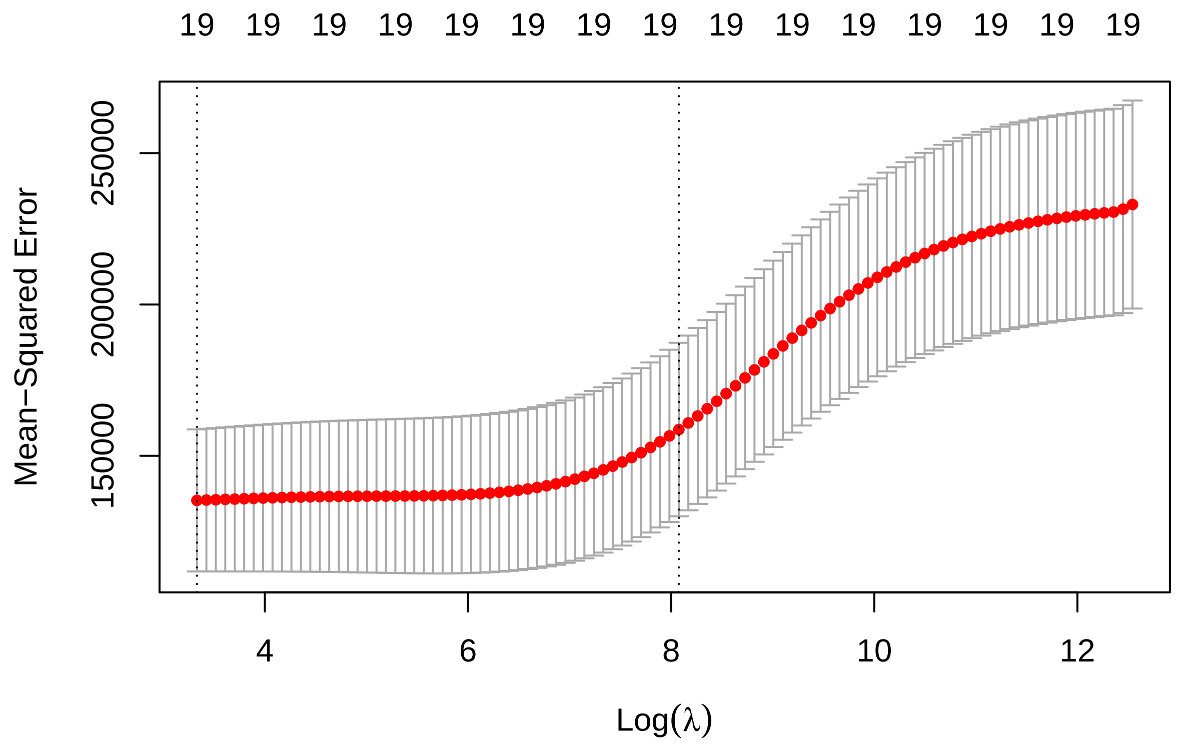
Instead of arbitrarily choosing $\lambda = 40$, it would be better to use cross-validation to choose the tuning parameter λ. We can do this using the built-in cross-validation function, `cv.glmnet()`. By default, the function performs 10-fold cross-validation, though this can be changed using the argument `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)

# Cross validation ridge regression
cv.fit <- cv.glmnet(x_train, y_train, alpha = 0, nfolds = 10)

# Finding best lambda
best_lamb <- cv.fit$lambda.min
best_lamb
```

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

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

```
# plotting ridge
plot(cv.fit)
```



What is the test MSE associated with this value of λ?

```
set.seed(lab.seed)

# Predicting with best lambda
preds_best = predict(ridge, s = best_lamb, newx = x_test)

# MSE
mean((preds_best - y_test)^2)
```

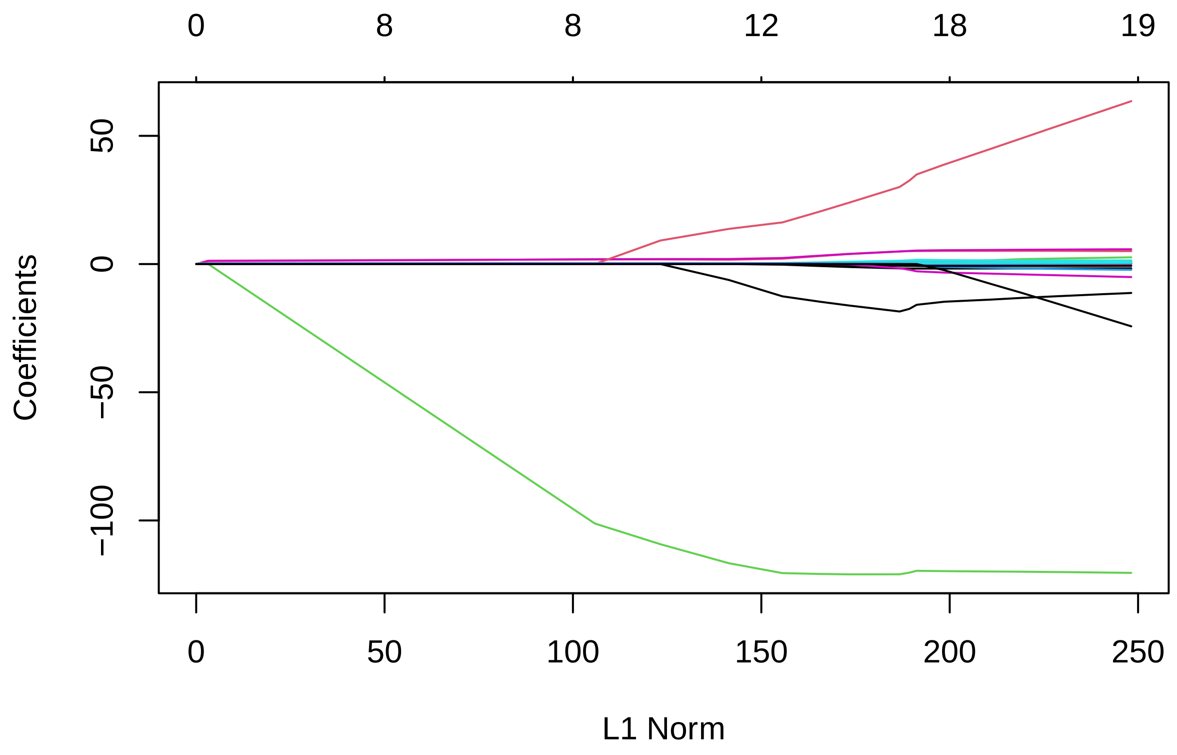
```
## [1] 60761.34
```

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.

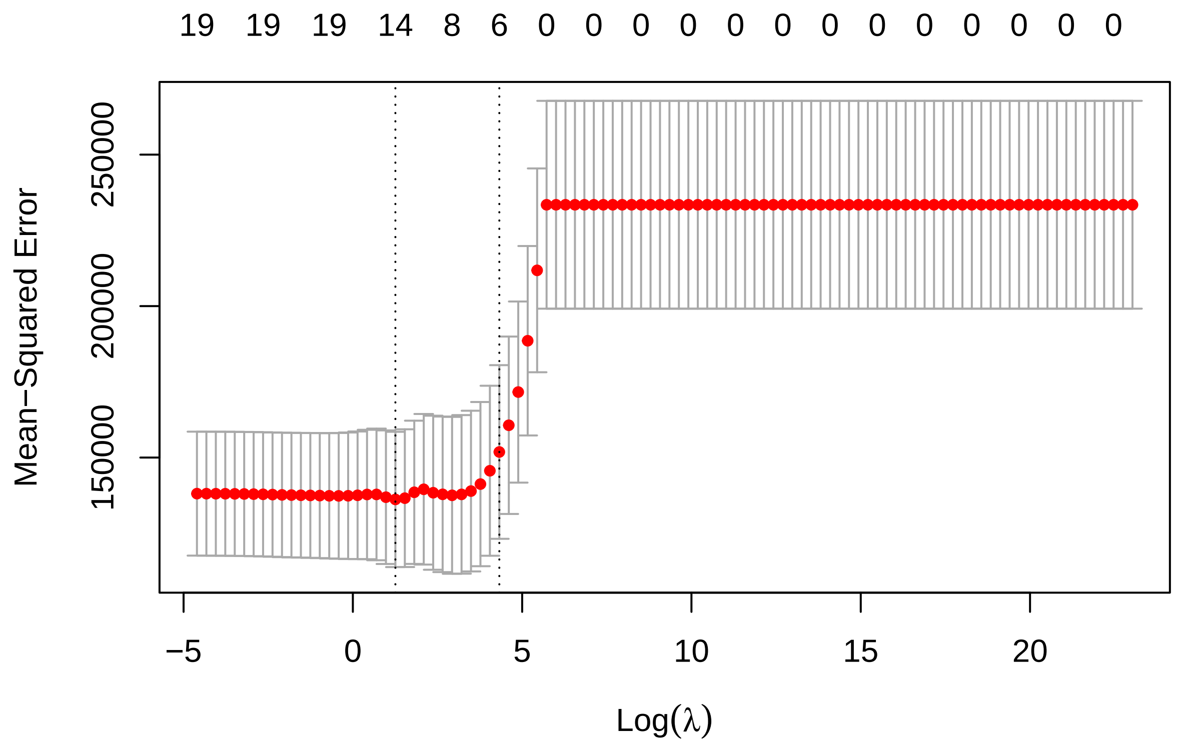
```
set.seed(lab.seed)

grid = 10^seq(10, -2, length = 100)

# Fitting lasso model with lambda grid

lasso <- glmnet(x_train, y_train, lambda = grid, alpha = 1)

plot(lasso)
```

```
## 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)

# CV Lasso
lasso_cv <- cv.glmnet(x_train, y_train, lambda = grid, alpha = 1)

#Plot
plot(lasso_cv)
```



```r
# Best lambda
bestlam = lasso_cv$lambda.min

# Predict with best lambda
lasso_pred = predict(lasso, s = bestlam, newx = x_test)


mean((lasso_pred - y_test)^2)
```

```
## [1] 64176.69
```

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.

```r
set.seed(lab.seed)

# To store data over loops
ks <- c(5,7,9,11,13,15)
lasso_mse <- c()
ridge_mse <- c()

for(k in ks) {
  # Define CV models
  cv.lasso = cv.glmnet(x_train, y_train,
                       alpha = 1, nfolds = k)
  cv.ridge = cv.glmnet(x_train, y_train,
                       alpha = 0, nfolds = k)
  # Predict with the best lambdas
  preds.lasso = predict(cv.lasso,
                        s = cv.lasso$lambda.min,
                        newx = x_test)
  preds.ridge = predict(cv.ridge,
                        s = cv.ridge$lambda.min,
                        newx = x_test)

  # Adding MSEs to list
  lasso_mse <- c(lasso_mse,
                 mean((preds.lasso - y_test)^2))

  ridge_mse <- c(ridge_mse,
                 mean((preds.ridge - y_test)^2))
}

cbind(ks, lasso_mse, ridge_mse)
```

```
##      ks lasso_mse ridge_mse
## [1,]  5  53761.81  60795.79
## [2,]  7  75016.24  60795.79
## [3,]  9  76616.80  60795.79
## [4,] 11  64513.56  60795.79
## [5,] 13  64170.13  60795.79
## [6,] 15  64513.56  60795.79
```

```
cat("Best ridge k is: ",ks[which(ridge_mse==min(ridge_mse))],"\n")
```

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

```
cat("Best lasso k is: ",ks[which(lasso_mse==min(lasso_mse))],"\n")
```

```
## Best lasso k is:  5
```

The best overall model is a lasso with k = 7. All ridge MSEs ended up the same.