Lab 10: Ridge Regression and the Lasso

PSTAT 131/231, Spring 2021

Learning Objectives

- Understanding the idea of shrinkage and its benefits
- Fitting ridge and lasso models by glmnet()
- Using cross-validation (CV) to find the best tuning parameter

---

We've already covered logistic regression extensively in lecture, lab, and on homeworks. The logistic regression model can work quite well when we have a large amount of data relative to the number of predictors ($n \gg p$), and the log-odds are approximately linear in the predictors. However, if there is no enough data, the estimates are very high variance and the error is typical quite large. We've explored notions of overfitting in past lectures (e.g. with perfect fitting the training data in a polynomial regression). In homework 4, problem 5e you should have seen that the magnitude of the coefficients in the higher order regression were much larger (e.g. high variance).

To fix these issues, we adopt the shrinkage method, which substantially reduces the variance (at the cost of increased bias).

In this lab, we are going to explore two of the most common *regularization* methods to logistic regression: ridge regression and LASSO regression.

Polynomial regression:

$$Y = \beta_0 + \beta_1 X + \cdots + \beta_p X^p + \epsilon$$

OLS:

$$\text{Minimize: } \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x^j)^2$$

Ridge regression:

$$\text{Minimize: } \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x^j)^2 + \lambda \sum_{j=1}^{p}\beta_j^2.$$

LASSO:

$$\text{Minimize: } \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x^j)^2 + \lambda \sum_{j=1}^{p}|\beta_j|.$$

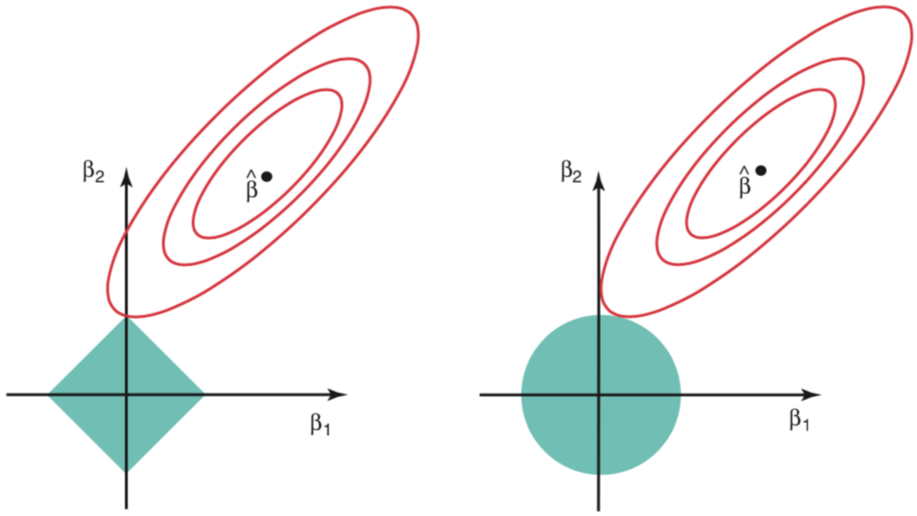


FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions, $|\beta_1| + |\beta_2| \leq s$ and $\beta_1^2 + \beta_2^2 \leq s$, while the red ellipses are the contours of the RSS.*

Ridge and LASSO can be accomplished using the `glmnet` package. The main function in this package is `glmnet()`, which can be used to fit ridge regression models, lasso models, and more. This function has slightly different syntax from other model-fitting functions. In particular, we must pass in an x matrix as well as a y vector, and we do not use the $y \sim x$ syntax.

The dataset for analysis is the Major League Baseball Data from the 1986 and 1987 seasons, which consists of 322 observations of major league players on 20 variables including the number of hits, number of errors, annual salary etc. Before proceeding, we first import required packages and ensure that the missing values have been removed.

```
# import required packages
library(ISLR)
library(glmnet)
library(dplyr)
library(tidyr)

# ensure that the missing values have been removed
Hitters = na.omit(Hitters)
```

We will now perform ridge regression and the lasso in order to predict Salary on the Hitters data. Let's set up our data.

```
x = model.matrix(Salary~., Hitters)[,-1]
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.

1. Basic Concepts and Functions

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. We first fit a ridge regression model.

```
grid = 10^seq(10, -2, length = 100)
ridge_mod = glmnet(x, y, alpha = 0, lambda = grid)
```

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 (λ is very large) containing only the intercept, to the least squares fit (λ is 0). 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).

Since we have added penalty to the norm of the coefficients, 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. Consider first the 50th choice of λ, where $\lambda = 11498$.

```
ridge_mod$lambda[50] #Display 50th lambda value
```

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

```
coef(ridge_mod)[,50] # Display coefficients associated with 50th lambda value
```

```
##   (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
```

```
sqrt(sum(coef(ridge_mod)[-1,50]^2)) # Calculate l2 norm
```

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

Then compare the above result with the coefficients for the 60th choice of λ, where $\lambda = 705$.

```
ridge_mod$lambda[60] #Display 60th lambda value
```

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

```
coef(ridge_mod)[,60] # Display coefficients associated with 60th lambda value
```

```
##  (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
sqrt(sum(coef(ridge_mod)[-1,60]^2)) # Calculate l2 norm
```

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

We indeed observe that the l^2 norm of the coefficients decreases when λ value is large. An alternative way to obtain the ridge regression coefficients for a new value of λ is by using the `prediction` function. Say we want $\lambda = 50$.

```r
predict(ridge_mod, s = 50, type = "coefficients")[1:20,]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  4.876610e+01 -3.580999e-01  1.969359e+00 -1.278248e+00  1.145892e+00
##           RBI         Walks         Years        CAtBat         CHits
##  8.038292e-01  2.716186e+00 -6.218319e+00  5.447837e-03  1.064895e-01
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  6.244860e-01  2.214985e-01  2.186914e-01 -1.500245e-01  4.592589e+01
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -1.182011e+02  2.502322e-01  1.215665e-01 -3.278600e+00 -9.496680e+00
```

2. Ridge Regression

We now split the samples into a training set and a test set in order to estimate the test error of ridge regression and the lasso. There are two common ways to randomly split a data set. The first is to produce a random vector of TRUE, FALSE elements and select the observations corresponding to TRUE for the training data. The second is to randomly choose a subset of numbers between 1 and n; these can then be used as the indices of the training observations. The two approaches work equally well.

```r
set.seed(1)
train=sample(1:nrow(x), nrow(x)/2)
test=(-train)
y.test=y[test]
```

2.1 Comparison with Least Squares and Constant Model

Next we fit a ridge regression model on the training set, and evaluate its MSE on the test set, using $\lambda = 4$. Note the use of the `predict()` function again. This time we get the predictions for a test set. Be careful that here the parameter name is "newx", rather than "newdata" in the previous labs.

```r
ridge.mod=glmnet(x[train,],y[train],alpha=0,lambda=grid, thresh = 1e-12)
ridge.pred=predict(ridge.mod,s=4,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 142199.2
```

Now we try to fit a ridge regression model with a very large value of λ.

```r
ridge.pred=predict(ridge.mod,s=1e10,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 224669.8
```

Since λ is very large it tends to force all the coefficients to 0, therefore we would expect getting a model containing only an intercept. This is equivalent to fitting a constant model, whose best choice is the mean of the training data. The following results indicates our conjecture is valid.

```
mean((mean(y[train])-y.test)^2)
```

```
## [1] 224669.9
```

From the above results it is apparent that the ridge regression model with $\lambda = 4$ leads to a much lower test MSE than fitting a model with just an intercept. We now check whether there is any benefit to performing ridge regression with $\lambda = 4$ instead of just performing least squares regression, which corresponds to $\lambda = 0$.

```
ridge.pred=predict(ridge.mod,s=0,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 167789.8
```

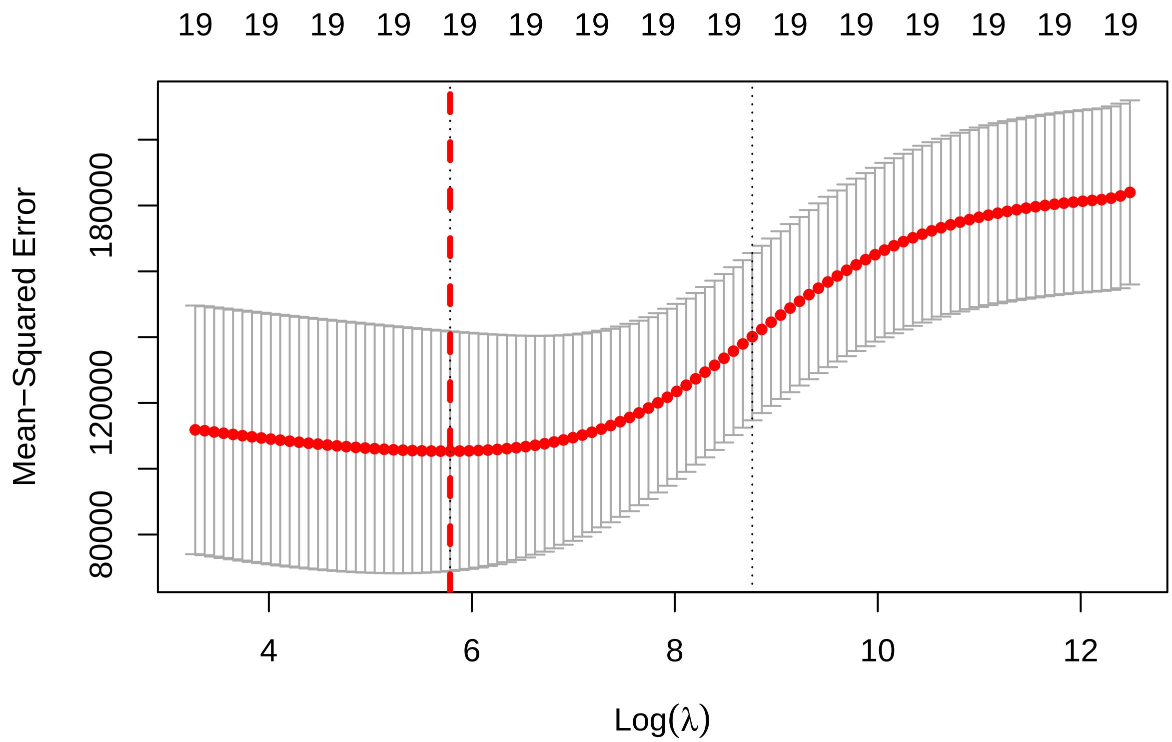
So the model obtained by $\lambda = 4$ has smaller MSE on the test set than both least squares regression and the constant model.

Remark: In general, if we want to fit a (unpenalized) least squares model, then we should use the `lm()` function, since that function provides more useful outputs, such as standard errors and p-values for the coefficients.

2.2 Cross-validation to Choose the Best Tuning Parameter

Instead of arbitrarily choosing $\lambda = 4$, 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 ten-fold cross-validation, though this can be changed using the argument "folds". Note that we set a random seed first so our results will be reproducible.

```
set.seed(1)
cv.out.ridge=cv.glmnet(x[train,], y[train], alpha = 0)
plot(cv.out.ridge)
abline(v = log(cv.out.ridge$lambda.min), col="red", lwd=3, lty=2)
```




```r
bestlam = cv.out.ridge$lambda.min
bestlam
```

```
## [1] 326.0828
```

Therefore, we see that the value of λ that results in the smallest cross-validation error is 212. What is the test MSE associated with this value of λ?

```r
ridge.pred=predict(ridge.mod,s=bestlam ,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 139856.6
```

This represents a further improvement over the test MSE that we got using $\lambda = 4$. Finally, we refit our ridge regression model on the full data set, using the value of λ chosen by cross-validation, and examine the coefficient estimates.

```r
out = glmnet(x,y,alpha=0)
predict(out,type="coefficients",s=bestlam)[1:20,]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs           RBI
##   15.44383135    0.07715547    0.85911581    0.60103107    1.06369007    0.87936105
##         Walks         Years        CAtBat         CHits        CHmRun         CRuns
##    1.62444616    1.35254780    0.01134999    0.05746654    0.40680157    0.11456224
##          CRBI        CWalks       LeagueN     DivisionW       PutOuts       Assists
##    0.12116504    0.05299202   22.09143189  -79.04032637    0.16619903    0.02941950
##        Errors    NewLeagueN
```

```
##   -1.36092945   9.12487767
```

This is the best model given from the ridge regression. Notice that none of the coefficients are zero. Ridge regression does not perform variable selection!

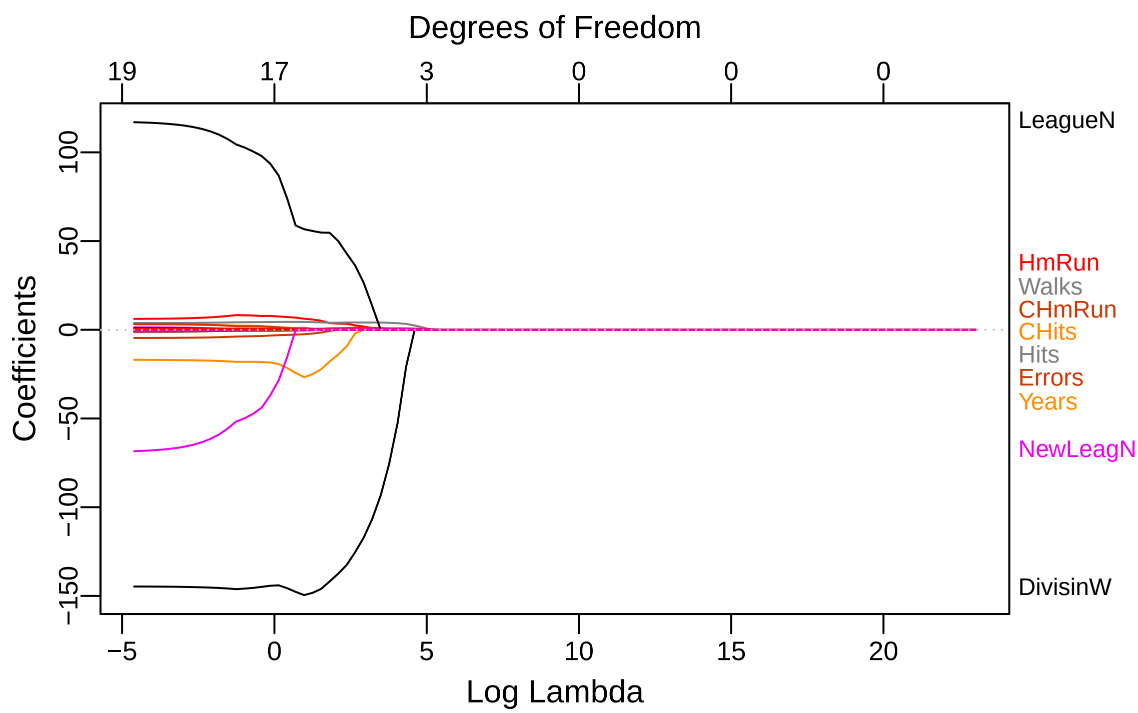
3. The Lasso

We saw that ridge regression with a wise choice of λ can outperform least squares as well as the null model on the Hitters data set. We now ask whether the lasso can yield either a more accurate or a more interpretable model than ridge regression. In order to fit a lasso model, we once again use the `glmnet()` function; however, this time we use the argument alpha = 1. Other than that change, we proceed just as we did in fitting a ridge model.

The LASSO has the nice feature that it will estiamte many of the coefficients as exactly 0. This is useful when some of the variables used in a multiple regression model are in fact not associated with the response. By removing these variables (by setting the corresponding coefficient estimates to zero), we obtain a model that is more interpretable. This is sometimes referred to as *variable selection*.

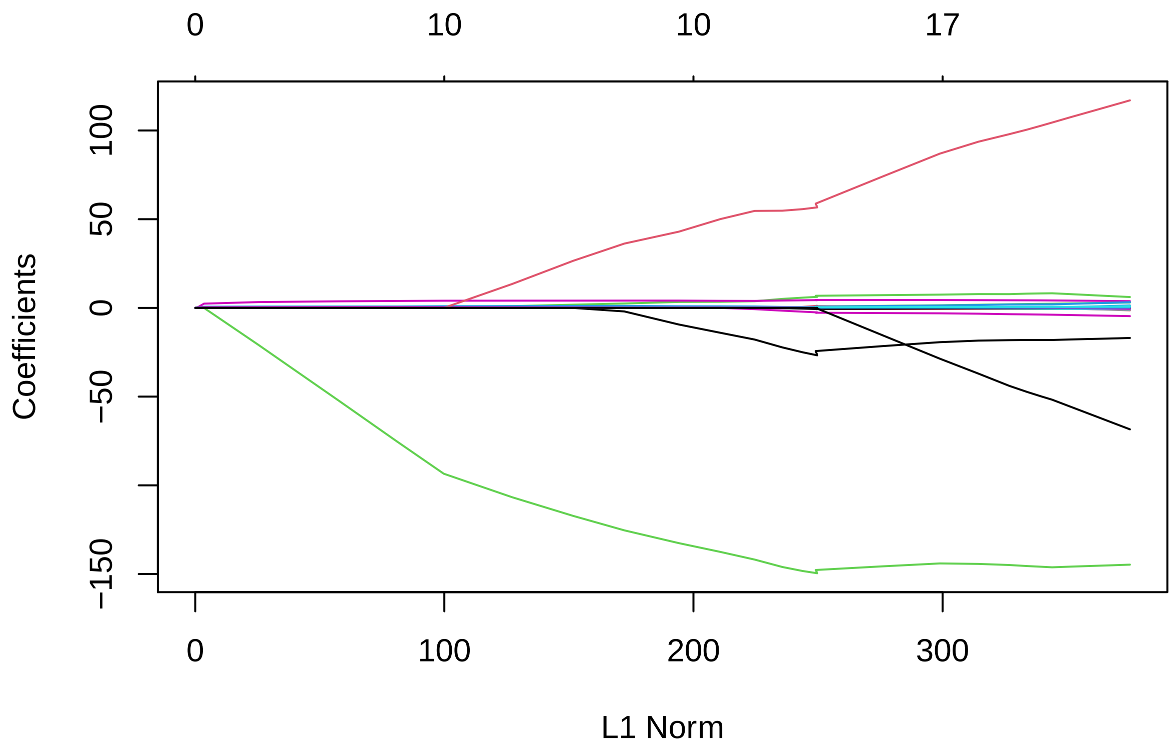
```r
lasso.mod <- glmnet(x[train ,], y[train], alpha=1, lambda=grid)
# plot(lasso.mod, xvar="lambda", label = TRUE)

library(plotmo)
plot_glmnet(lasso.mod, xvar="lambda")
```



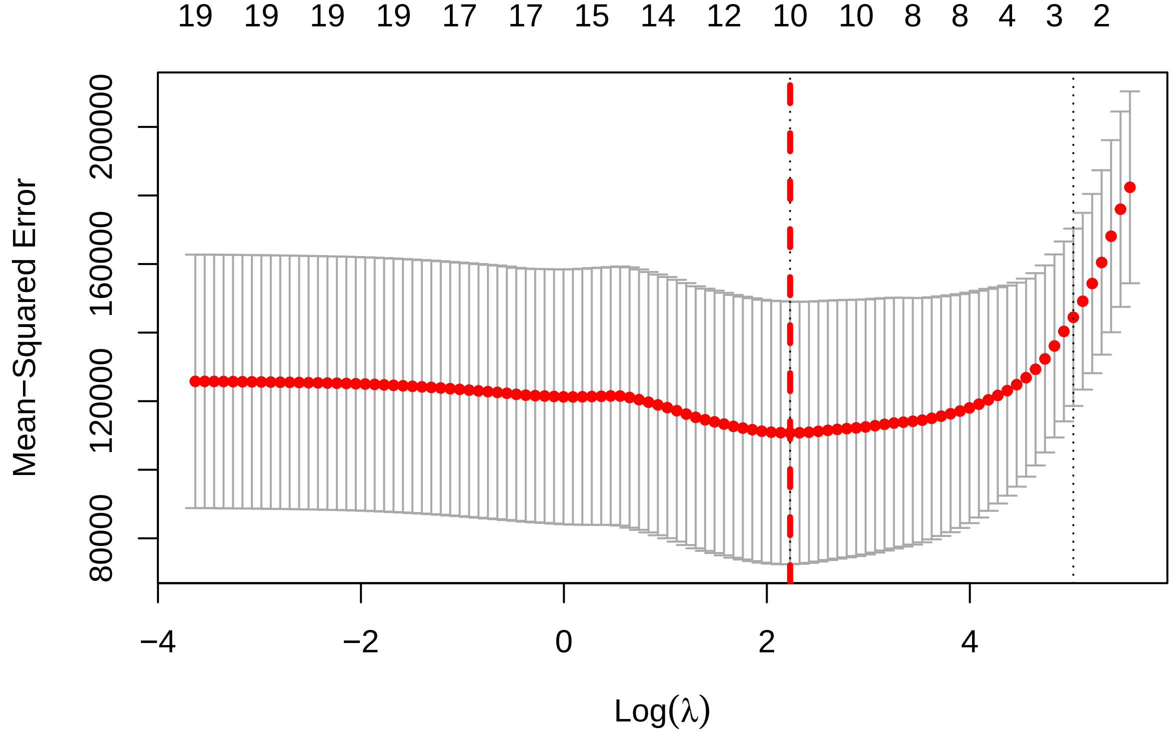

```r
plot(lasso.mod)
```

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



We can see from the coefficient plot that depending on the choice of tuning parameter, some of the coefficients will be exactly equal to zero. We now perform cross-validation and compute the associated test error.

```
set.seed(1)
cv.out.lasso = cv.glmnet(x[train,], y[train], alpha = 1)
plot(cv.out.lasso)
abline(v = log(cv.out.lasso$lambda.min), col="red", lwd=3, lty=2)
```




```r
bestlam = cv.out.lasso$lambda.min
lasso.pred = predict(lasso.mod, s = bestlam, newx = x[test,])
mean((lasso.pred-y.test)^2)
```

```
## [1] 143673.6
```

This is substantially lower than the test set MSE of the null model and of least squares, and very similar to the test MSE of ridge regresion with λ chosen by cross-validation.

However, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that 12 of the 19 coefficient estimates are exactly zero. So the lasso model with λ chosen by cross-validation contains only seven variables.

```r
out=glmnet(x,y,alpha=1,lambda=grid)
lasso.coef=predict(out,type="coefficients",s=bestlam)[1:20,]
lasso.coef
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##    1.27479059   -0.05497143    2.18034583    0.00000000    0.00000000
##           RBI         Walks         Years        CAtBat         CHits
##    0.00000000    2.29192406   -0.33806109    0.00000000    0.00000000
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##    0.02825013    0.21628385    0.41712537    0.00000000   20.28615023
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -116.16755870    0.23752385    0.00000000   -0.85629148    0.00000000
```