Linear Model Selection and Regularization

Debopriya Ghosh

2019-07-30

- Shrinkage Methods
  - Ridge Regression
  - LASSO Regression

In the last lecture we discussed some ways in which the simple linear model can be improved, by replacing plain least squares fitting with some alternative fitting procedures. There are many alternatives, both classical and modern, to using least squares to fit. Here, we discuss shrinkage methods.

Shrinkage Methods

- The subset selection methods described in last lecture involve using least squares to fit a linear model that contains a subset of the predictors.
- As an alternative, we can fit a model containing all p predictors using a technique that constrains or regularizes the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero.
- Shrinking the coefficient estimates can significantly reduce their variance.
- The two best-known techniques for shrinking the regression coefficients towards zero are ridge regression and the lasso.

Ridge Regression

- The least squares fitting procedure estimates $\beta_0, \beta_1, \ldots, \beta_p$ using the values that minimize *RSS*

$$RSS = \Sigma_{i=1}^{n}(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij})^2$$

- Ridge regression is very similar to least squares, except that the coefficients are estimated by minimizing a slightly different quantity.

$$\Sigma_{i=1}^{n}(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\Sigma_{i=1}^{p}\beta_j^2 = RSS + \lambda\Sigma_{i=1}^{p}\beta_j^2$$

- As with least squares, ridge regression seeks coefficient estimates that fit the data well, by making the *RSS* small.

- However, the second term, $\lambda\Sigma_{i=1}^{p}\beta_j^2$, called *shrinkage penalty*, is small when $\beta_0, \beta_1, \ldots, \beta_p$ are close to zero, and so it has the effect of shrinking the estimates of β_j towards zero.
- The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates.

- When $\lambda = 0$, the penalty term has no effect, and ridge regression will produce the least squares estimates.
- However, as $\lambda \to \infty$, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero.
- Unlike least squares, which generates only one set of coefficient estimates, ridge regression will produce a different set of coefficient estimates for each value of λ.
- Selecting a good value for λ is critical.

Example: Baseball dataset

- We will 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.
- We will perform *ridge* regression and the *lasso* in order to predict Salary on the Hitters data.
- Before proceeding ensure that the missing values have been removed from the data,

```
library(ISLR)
data(Hitters)
Hitters = na.omit(Hitters)
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.
- 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.

```
#install.packages("glmnet")
library(glmnet)
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 containing only the intercept, to the least squares fit.
- Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by coef().

```
dim(coef(ridge.mod))
```

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

- We expect the coefficient estimates to be much smaller when a large value of λ is used, as compared to when a small value of λ is used.

```
ridge.mod$lambda [50]
coef(ridge.mod)[,50]
```

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

```
ridge.mod$lambda [60]
coef(ridge.mod)[,60]
```

```
## [1] 705.4802
##  (Intercept)        AtBat         Hits        HmRun         Runs
##  54.32519950   0.11211115   0.65622409   1.17980910   0.93769713
##          RBI        Walks        Years       CAtBat        CHits
##   0.84718546   1.31987948   2.59640425   0.01083413   0.04674557
##       CHmRun        CRuns         CRBI       CWalks      LeagueN
##   0.33777318   0.09355528   0.09780402   0.07189612  13.68370191
##    DivisionW      PutOuts      Assists       Errors   NewLeagueN
## -54.65877750   0.11852289   0.01606037  -0.70358655   8.61181213
```

- We can use the predict() function for a number of purposes.
- For example, we can obtain the ridge regression coefficients for a new value of λ, say 50

```
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
```

- 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.

```
set.seed(1)
train = sample(1:nrow(x), nrow(x)/2)
```

```
test = (-train)
y.test=y[test]
```

- Next we fit a ridge regression model on the training set, and evaluate its MSE on the test set, using $\lambda = 4$.

```
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
```

- The test MSE is 101037.
- Note that if we had instead simply fit a model with just an intercept, we would have predicted each test observation using the mean of the training observations.
- In that case, we could compute the test set MSE like this:

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

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

- We could also get the same result by fitting a ridge regression model with a very large value of λ.

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

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

- So fitting a 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.

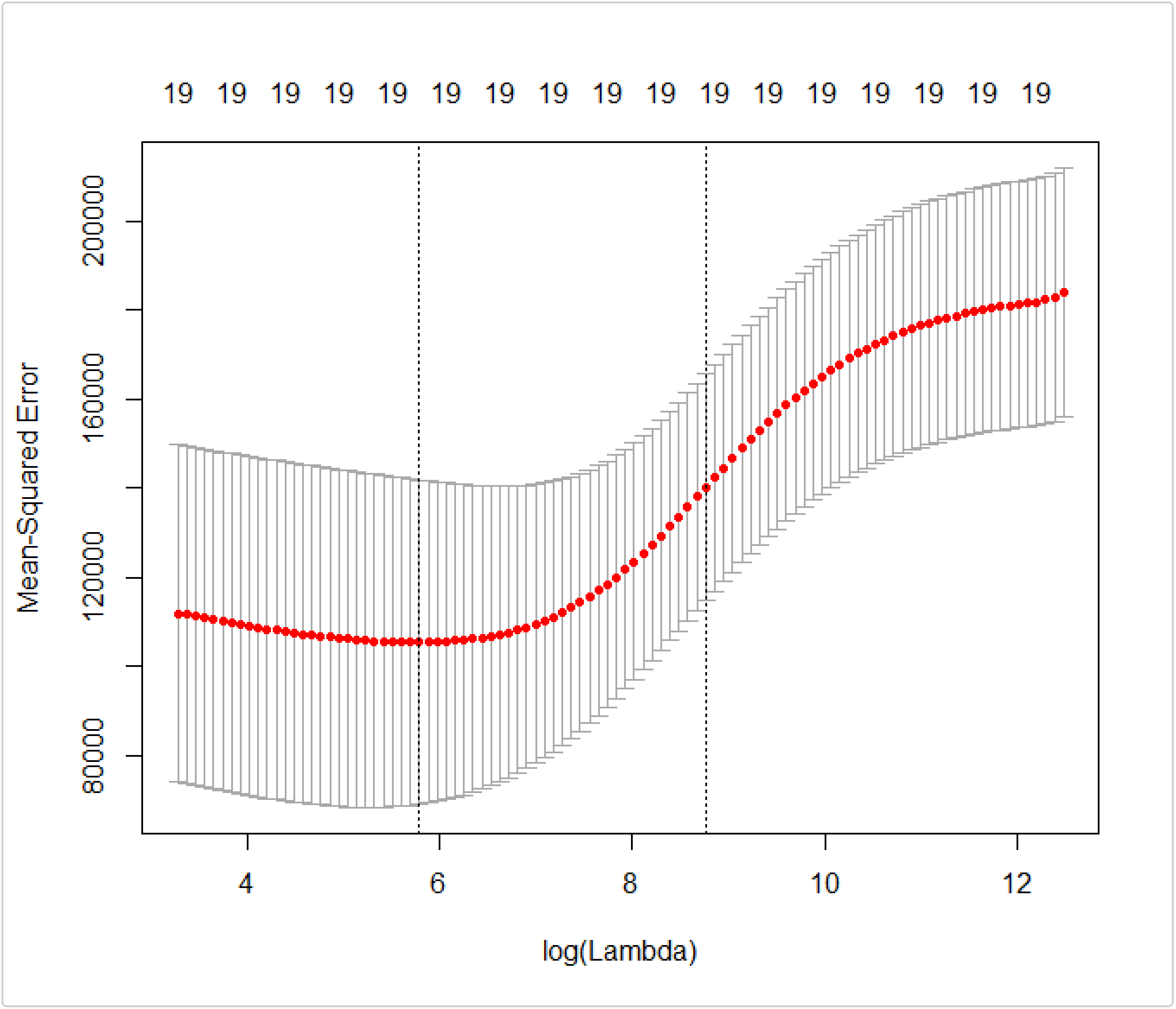
```
ridge.pred = predict(ridge.mod,s = 0,newx = x[test ,])
mean((ridge.pred-y.test)^2)

lm(y ~ x, subset = train)
predict(ridge.mod,s = 0,type = "coefficients")[1:20,]
```

```
## [1] 167789.8
##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
## (Intercept)       xAtBat         xHits        xHmRun         xRuns
##    274.0145      -0.3521      -1.6377       5.8145       1.5424
##        xRBI       xWalks       xYears      xCAtBat       xCHits
##      1.1243       3.7287     -16.3773      -0.6412       3.1632
##     xCHmRun       xCRuns        xCRBI       xCWalks     xLeagueN
##      3.4008      -0.9739      -0.6005       0.3379     119.1486
##  xDivisionW     xPutOuts     xAssists      xErrors  xNewLeagueN
##   -144.0831       0.1976       0.6804      -4.7128     -71.0951
##
##  (Intercept)        AtBat          Hits        HmRun         Runs
##  274.2089049   -0.3699455    -1.5370022    5.9129307    1.4811980
##          RBI        Walks         Years       CAtBat        CHits
##    1.0772844    3.7577989   -16.5600387   -0.6313336    3.1115575
##       CHmRun        CRuns          CRBI       CWalks      LeagueN
##    3.3297885   -0.9496641    -0.5694414    0.3300136  118.4000592
##    DivisionW      PutOuts       Assists       Errors   NewLeagueN
## -144.2867510    0.1971770     0.6775088   -4.6833775  -70.1616132
```

- 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().

```
set.seed(1)
cv.out = cv.glmnet(x[train,],y[train],alpha = 0)
plot(cv.out)
```



- By default, the function performs ten-fold cross-validation, though this can be changed using the argument folds.

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

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

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

```
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.

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

```
##  (Intercept)        AtBat         Hits        HmRun         Runs 
##  15.44383135   0.07715547   0.85911581   0.60103107   1.06369007 
##          RBI        Walks        Years       CAtBat        CHits 
##   0.87936105   1.62444616   1.35254780   0.01134999   0.05746654 
##       CHmRun        CRuns         CRBI       CWalks      LeagueN 
##   0.40680157   0.11456224   0.12116504   0.05299202  22.09143189 
##    DivisionW      PutOuts      Assists       Errors   NewLeagueN 
## -79.04032637   0.16619903   0.02941950  -1.36092945   9.12487767
```

Why Does Ridge Regression Improve Over Least Squares?

- Ridge regression's advantage over least squares is rooted in the bias-variance trade-off.
- As λ increases, the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias.
- The least squares coefficient estimates, which correspond to ridge regression with $\lambda = 0$, the variance is high but there is no bias.
- But as λ increases, the shrinkage of the ridge coefficient estimates leads to a substantial reduction in the variance of the predictions, at the expense of a slight increase in bias.
- In situations where the relationship between the response and the predictors is close to linear, the least squares estimates will have low bias but may have high variance.
- This means that a small change in the training data can cause a large change in the least squares coefficient estimates.
- In particular, when the number of variables p is almost as large as the number of observations n, the least squares estimates will be extremely variable.
- And if $p > n$, then the least squares estimates do not even have a unique solution, whereas ridge regression can still perform well by trading off a small increase in bias for a large decrease in variance.
- Hence, ridge regression works best in situations where the least squares estimates have high variance.
- Ridge regression also has substantial computational advantages over best subset selection.

LASSO Regression

- Ridge regression does have one obvious disadvantage.
- Unlike best subset, forward stepwise, and backward stepwise selection, which will generally select models that involve just a subset of the variables, ridge regression will include all p predictors in the final model.
- The penalty will shrink all of the coefficients towards zero, but it will not set any of them exactly to zero.

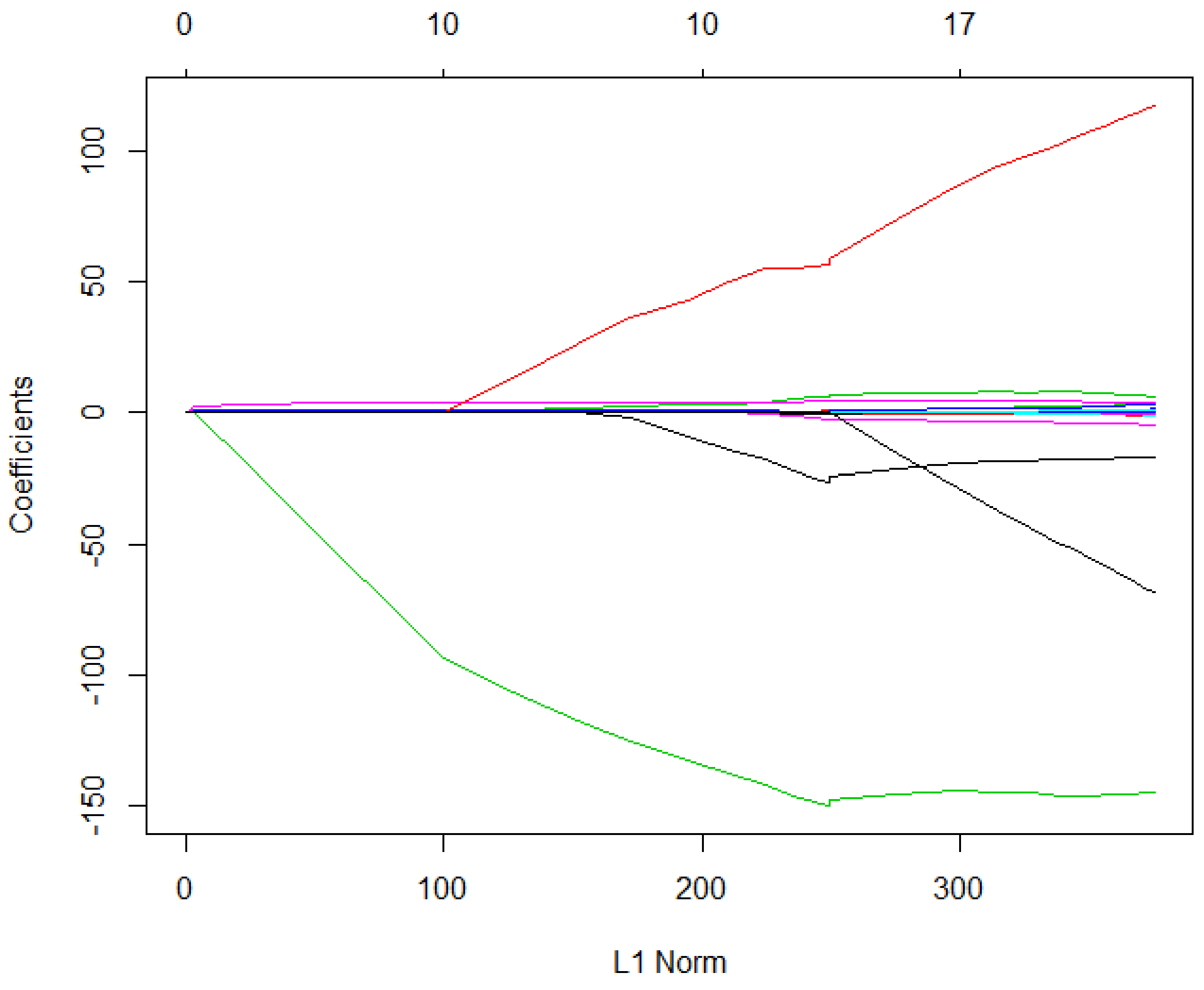
- This may not be a problem for prediction accuracy, but it can create a challenge in model interpretation in settings in which the number of variables p is quite large.
- The lasso is a relatively recent alternative to ridge regression that overcomes this disadvantage.
- The lasso coefficients minimize the quantity –

$$\Sigma_{i=1}^{n}(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\Sigma_{i=1}^{p}|\beta_j| = RSS + \lambda\Sigma_{i=1}^{p}|\beta_j|$$

- As with ridge regression, the lasso shrinks the coefficient estimates towards zero.

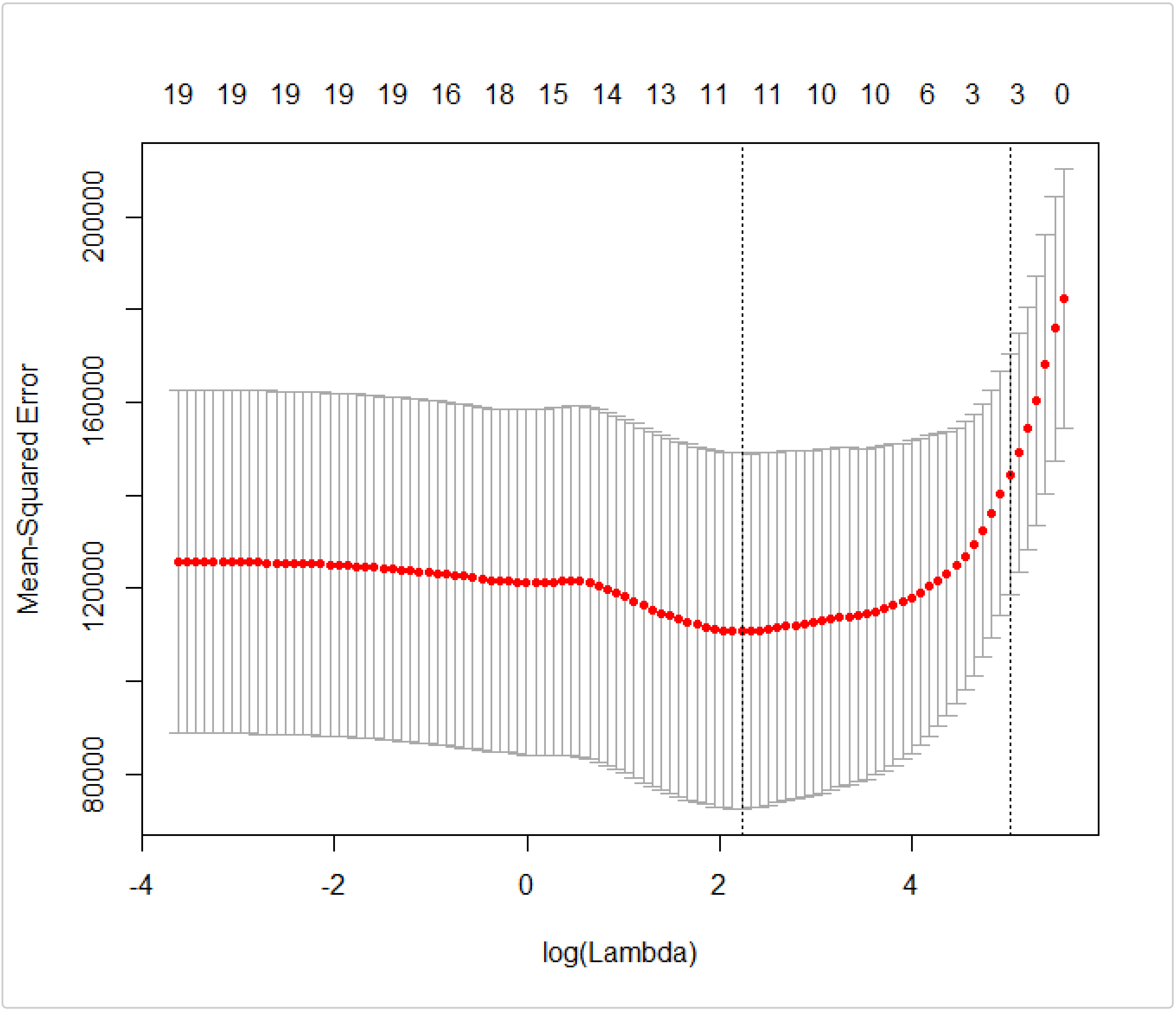
- However, in the case of the lasso, the penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large.
- Like best subset selection, the lasso performs variable selection. As a result, models generated from the lasso are generally much easier to interpret than those produced by ridge regression.
- We say that the lasso yields sparse models, that involve only a subset of the variables.
- As in ridge regression, selecting a good value of λ for the lasso is critical.

```
lasso.mod = glmnet(x[train,],y[train],alpha = 1,lambda = grid)
plot(lasso.mod)
```



- 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 = cv.glmnet(x[train,],y[train],alpha = 1)
plot(cv.out)
```



```
bestlam = cv.out$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 regression with λ chosen by cross-validation.

- Lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse.

```
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
```

-
  - 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.

```
lasso.coef[lasso.coef!=0]
```

```
##   (Intercept)         AtBat          Hits         Walks         Years
##    1.27479059   -0.05497143    2.18034583    2.29192406   -0.33806109
##        CHmRun         CRuns          CRBI       LeagueN     DivisionW
##    0.02825013    0.21628385    0.41712537   20.28615023 -116.16755870
##       PutOuts        Errors
##    0.23752385   -0.85629148
```