Hitters: ISLR Lab Example

Matt Richey

3/15/2020

Introduction

Let's recreate the lab from ISLR Chapter 6 analyzing the Hitters data set using penalized regression. There is nothing really different here than in the ISLR book, but we will do it all in a tidyverse context.

The Hitters Data

Load the ILSR library and include the Hitters data set.

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------------- tidyverse 1.3.0

## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------------------------------------------ tidyverse_conflicts()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

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

## Loaded glmnet 3.0-2
```

```r
library(ISLR)
```

Include the Hitters data from the ILSR package.

```r
data(Hitters)
head(Hitters)
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Andy Allanson      293   66     1   30  29    14     1    293    66      1
## -Alan Ashby         315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis        479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga   321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin    594  169     4   74  51    35    11   4408  1133     19
##                   CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Andy Allanson       30   29     14      A        E     446      33     20
## -Alan Ashby         321  414    375      N        W     632      43     10
## -Alvin Davis        224  266    263      A        W     880      82     14
## -Andre Dawson       828  838    354      N        E     200      11      3
## -Andres Galarraga    48   46     33      N        E     805      40      4
## -Alfredo Griffin    501  336    194      A        W     282     421     25
##                   Salary NewLeague
## -Andy Allanson        NA         A
## -Alan Ashby        475.0         N
## -Alvin Davis       480.0         A
## -Andre Dawson      500.0         N
## -Andres Galarraga   91.5         N
## -Alfredo Griffin   750.0         A
```

This is a data set representing the performance of Major League (US) baseball players in 1986. The goal is to predict their salary from the other variables.

We see there are 59 players for which there is no salary information.

```r
with(Hitters,sum(is.na(Salary)))
```

```
## [1] 59
```

Eliminate these from the data set.

```r
noSal <- with(Hitters,is.na(Salary))
Hitters <- Hitters[!noSal,]
```

What do we have now?

```r
dim(Hitters)
```

```
## [1] 263  20
```

Ok, down to 263 observations.

Ridge Regression to Predict Salary

The ridge regression coefficient estimates are determined by finding the $\hat{\beta}_1, ..., \hat{\beta}_p$ such that

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

is minimized, subject to the constraint

$$\sum_{j=1}^{p}\beta_j^2 \leq t.$$

Ridge regression only works with vector/matrix inputs

```
numCol <- ncol(Hitters)
names(Hitters)
```

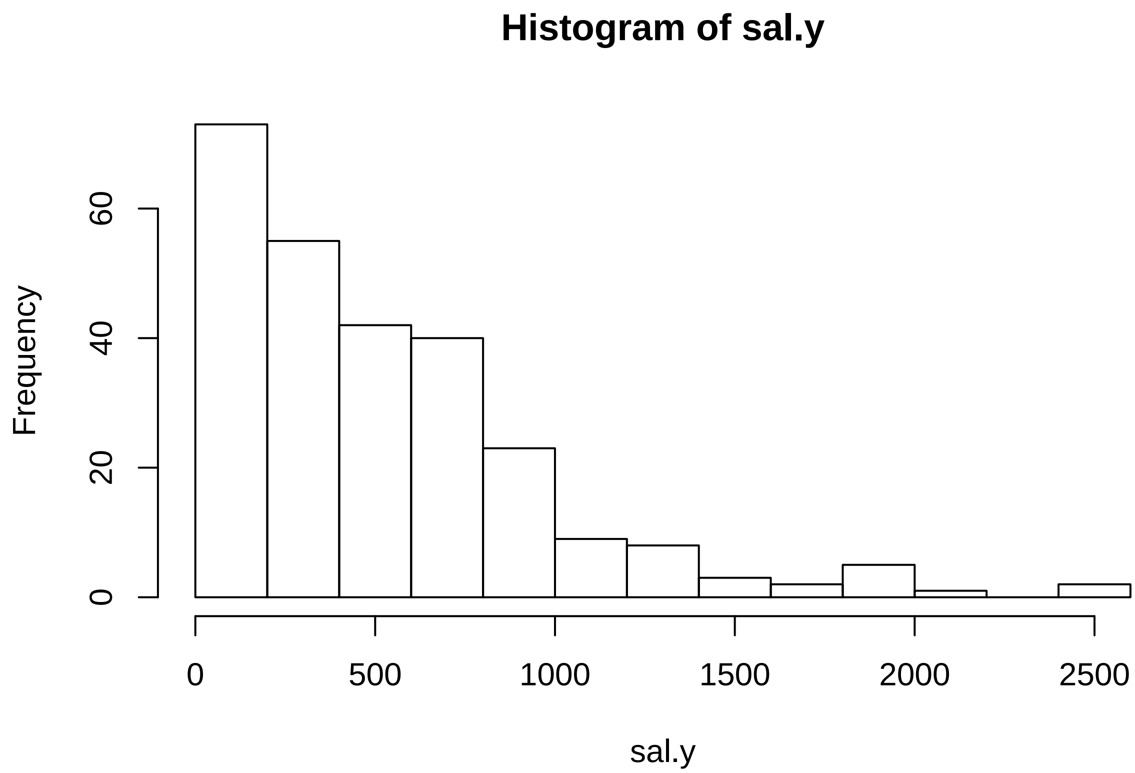
```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
##  [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
## [19] "Salary"    "NewLeague"
```

```
salCol <- 19 ## salary
```

```
vals.x <- data.matrix(Hitters[,-salCol])
sal.y <- data.matrix(Hitters[,salCol])
```

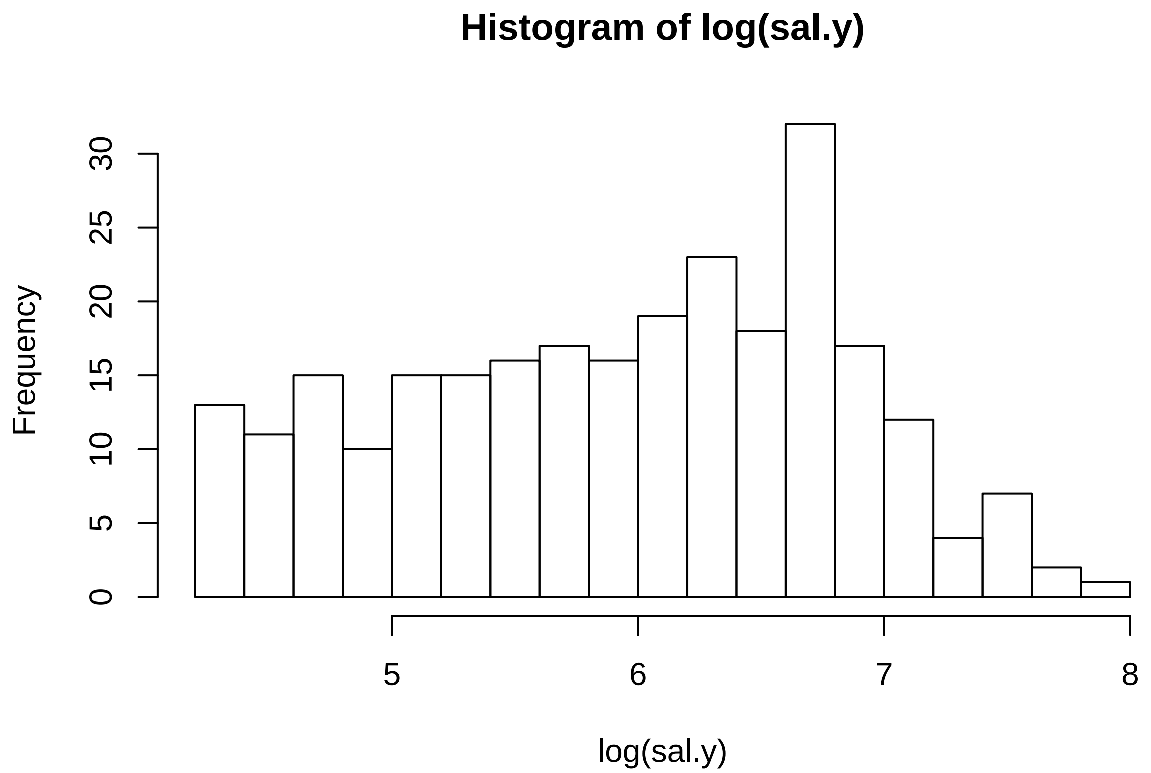
A quick glance at the salary data. Salary data is usually skewed right

```
hist(sal.y)
```



It might be better to use the log of the salaries to make this look more normal and hence conform better with assumptions of regression.

```
hist(log(sal.y),breaks=25)
```

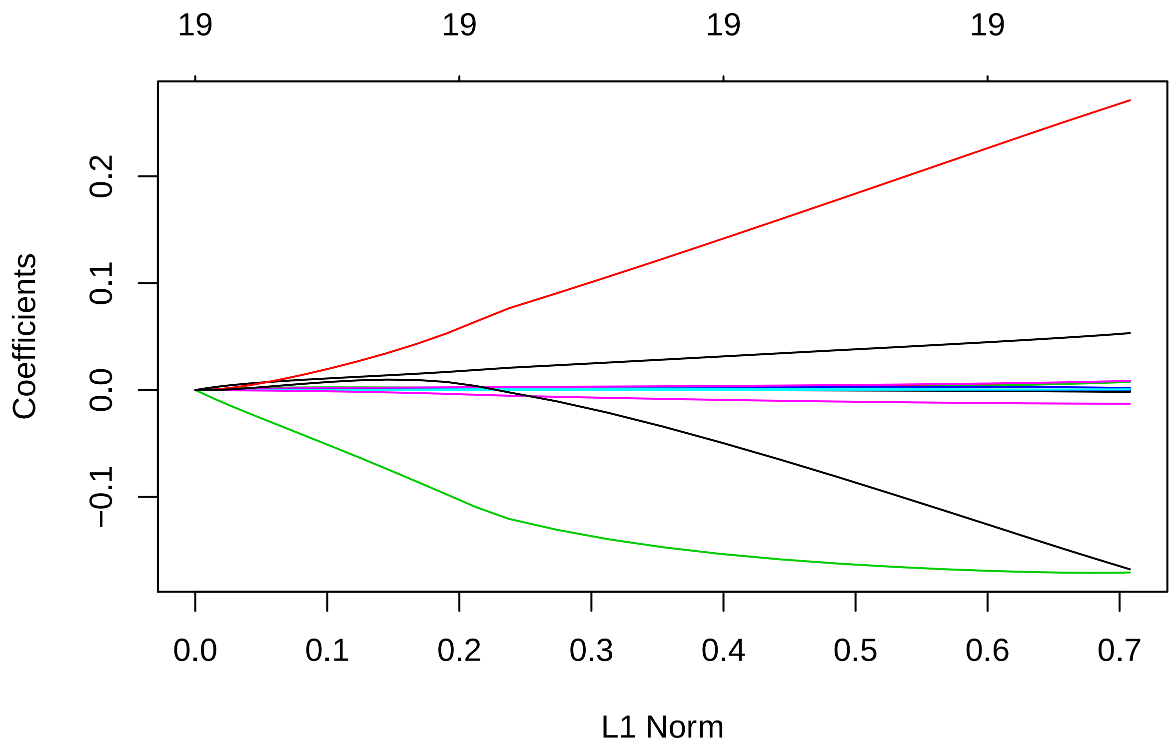



```
sal.y <- log(sal.y) ## Use log salary (different from ILSR lab)
```

Let's start with just the salary. Use a lambda grid with 100 values

```
lambda.grid <- 10^seq(10,-2,length=100)
mod.ridge <- mod.ridge <- glmnet(vals.x,
                                 sal.y,
                                 alpha=0,
                                 lambda=lambda.grid )
```

```
plot(mod.ridge)
```



The coefficients are computed for each of the 100 lambda values.

```
coef1 <- coef(mod.ridge)
dim(coef1)
```

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

Exam these coefficients at one lambda value.

```
(lambda.val <- lambda.grid[50])
```

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

```
coef1[,50]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  5.926569e+00  1.932600e-07  6.835543e-07  2.662337e-06  1.143248e-06
##           RBI         Walks         Years        CAtBat         CHits
##  1.177222e-06  1.366023e-06  7.691013e-06  1.832590e-08  6.561286e-08
##        CHmRun         CRuns          CRBI        CWalks        League
##  4.372577e-07  1.286456e-07  1.278491e-07  1.419904e-07 -8.779229e-07
##      Division       PutOuts       Assists        Errors     NewLeague
## -2.056417e-05  5.501829e-08  2.363554e-08 -2.157416e-07 -1.424732e-06
```

We could get these a different way via predict…

```r
predict(mod.ridge,s=lambda.val,type="coefficient")
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  5.926569e+00
## AtBat        1.932600e-07
## Hits         6.835543e-07
## HmRun        2.662337e-06
## Runs         1.143248e-06
## RBI          1.177222e-06
## Walks        1.366023e-06
## Years        7.691013e-06
## CAtBat       1.832590e-08
## CHits        6.561286e-08
## CHmRun       4.372577e-07
## CRuns        1.286456e-07
## CRBI         1.278491e-07
## CWalks       1.419904e-07
## League      -8.779229e-07
## Division    -2.056417e-05
## PutOuts      5.501829e-08
## Assists      2.363554e-08
## Errors      -2.157416e-07
## NewLeague   -1.424732e-06
```

We are interested in the predictive power of ridge regression.

Create a train/test combination.

```r
N <- nrow(Hitters)
train <- sample(1:N,N/2,rep=F)
train.x <- vals.x[train,]
test.x <- vals.x[-train,]
train.y <- sal.y[train]
test.y <- sal.y[-train]
```

As always, build on train, evaluate on test.

```r
mod.ridge <- glmnet(train.x,
                    train.y,
                    alpha=0,
                    lambda=lambda.grid )
```

glmnet is nice in how it handles grids of lambdas. We can now predict at one particular lambda value, say lambda=4.

```r
lambda.val <- 4
sal.pred <- predict(mod.ridge,
                newx = test.x,
                s=lambda.val)
```

Now we have one prediction for each observation (player). Let's compute the MSE.

```
mean((sal.pred-test.y)^2)
```

```
## [1] 0.4636148
```

Ridge regression with lambda=0 is is simply linear regression.

```
sal.pred.0 <- predict(mod.ridge,
                newx = test.x,
                s=0)
(mse0 <- mean((sal.pred.0-test.y)^2))
```

```
## [1] 0.4083832
```

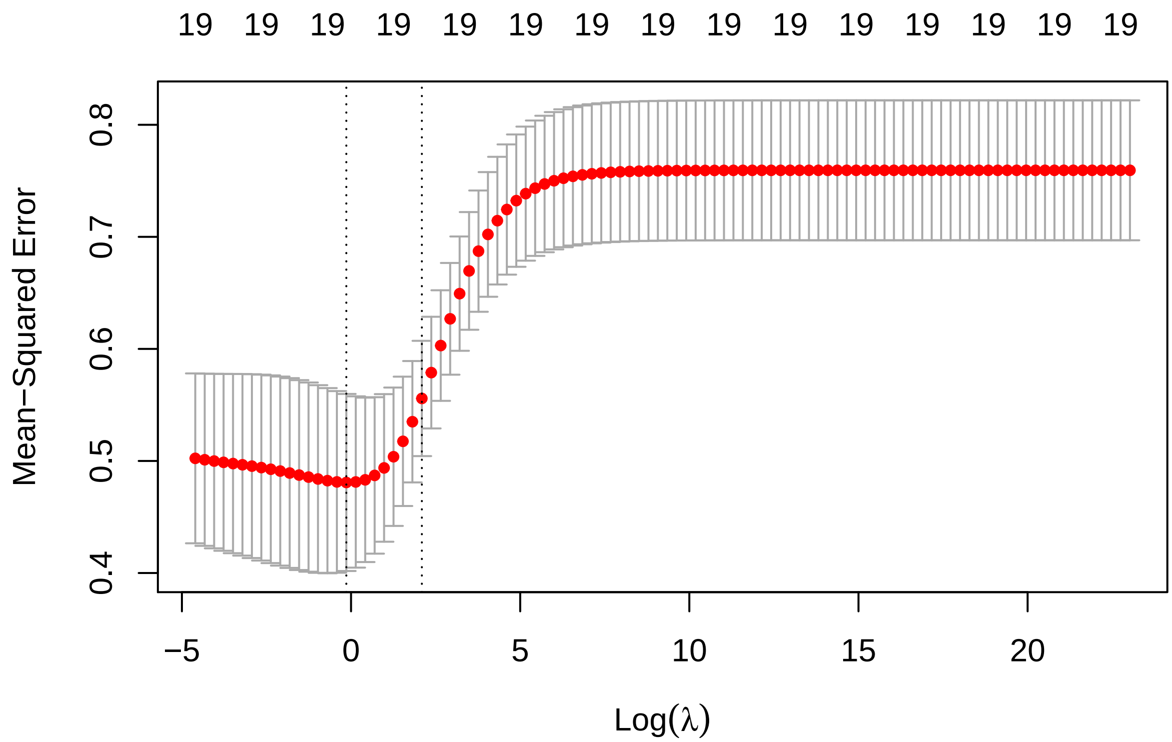
Looks like linear regression beats ridge regression with lambda=4.

Now let's cross validate over the range of lambda values.

```
mod.ridge.cv <- cv.glmnet(
  train.x,
  train.y,
  alpha=0,
  lambda=lambda.grid)
```

And the plot. . .

```
plot(mod.ridge.cv)
```



Looks like a clear choice for the optimal lambda near log(lambda) = 6

```
(lambda.opt <- mod.ridge.cv$lambda.min)
```

```
## [1] 0.869749
```

```
log(lambda.opt)
```

```
## [1] -0.1395506
```

Use this in glmnet to predict on the test data again.

```
sal.pred <- predict(mod.ridge,
                    newx = test.x,
                    s=lambda.opt)
(mse.ridge <- mean((sal.pred-test.y)^2))
```

```
## [1] 0.3918562
```

How does this compare to the linear regression mse?

```
c(mse0,mse.ridge)
```

```
## [1] 0.4083832 0.3918562
```

Ok. . . so we get a slight improvement in the prediction of the (log) salary with Ridge Regression.

Lasso

Lasso works about the same way. The big difference is that we are now use.

The lasso coefficient estimates are determined by finding the $\hat{\beta}_1, ..., \hat{\beta}_p$ such that

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

is minimized, subject to the constraint

$$\sum_{j=1}^{p} |\beta_j| \leq t$$

Note the absolute value!

Computationall, almost everything is the same. The difference is at the end we do variable selection.

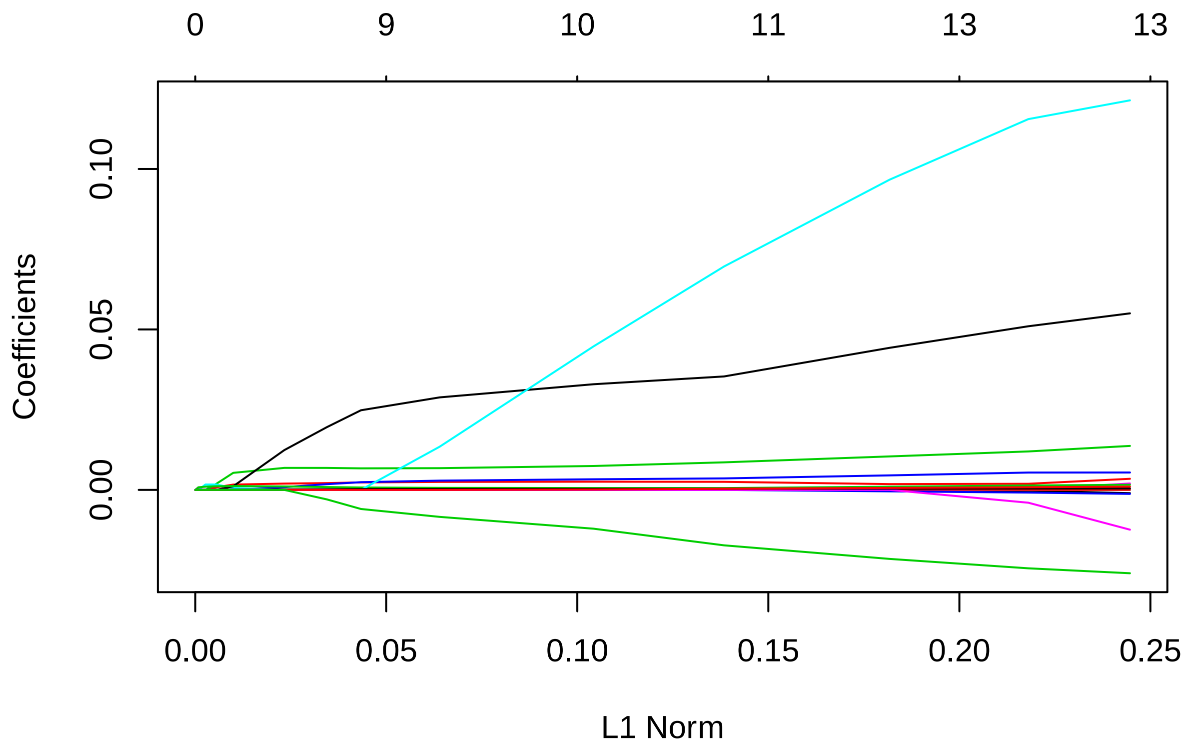
Now let's cross validate over the range of lambda values.

```
mod.lasso <- glmnet(
  train.x,
  train.y,
  alpha=1, ## for lasso
  lambda=lambda.grid)
```

And the plot…

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

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



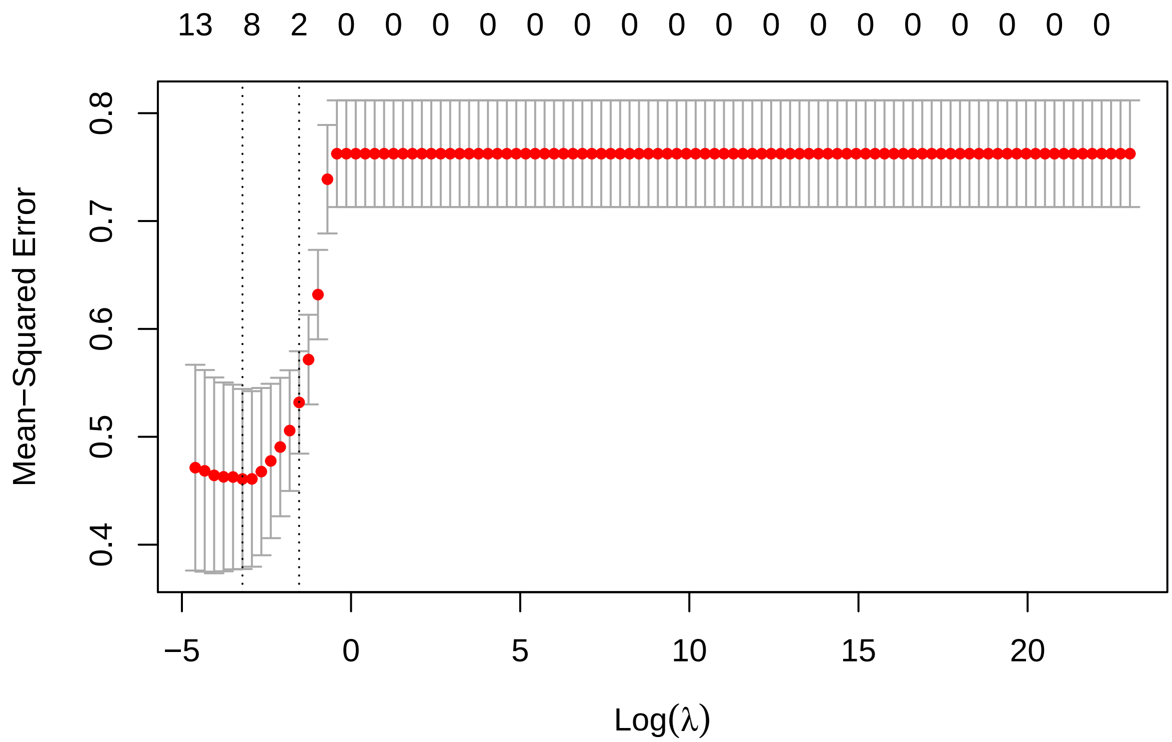
Interesting,this already shows the potential for some variable elimination. It looks as if several of the variables are hitting zero.

Now let's cross validate over the range of lambda values.

```
mod.lasso.cv <- cv.glmnet(
  train.x,
  train.y,
  alpha=1, ##for  lasso
  lambda=lambda.grid)
```

And the plot…

```
plot(mod.lasso.cv)
```



Looks like a clear choice for the optimal lambda near log(lambda) = 6

```
(lambda.opt <- mod.lasso.cv$lambda.min)
```

```
## [1] 0.04037017
```

```
log(lambda.opt)
```

```
## [1] -3.209664
```

Use this in glmnet to predict on the test data again.

```
sal.pred <- predict(mod.lasso,
                    newx = test.x,
                    s=lambda.opt)
(mse.lasso <- mean((sal.pred-test.y)^2))
```

```
## [1] 0.3897474
```

How does this compare to the linear regression and ridge mse?

```
c(mse0,mse.ridge,mse.lasso)
```

```
## [1] 0.4083832 0.3918562 0.3897474
```

Not much different, in the same ballpark (argh).

The important message here is how we can potentially select a smaller set of predictors.

Use the mod.lasso (from above) in predict, specifying that lambda is lambda.opt.

here's one way to get to the non-zero coefficients.

First, get all the coefficients

```
(coefs <- predict(mod.lasso,type="coefficients",s=lambda.opt))
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  4.7910046022
## AtBat        .
## Hits         0.0024864156
## HmRun        0.0067768571
## Runs         0.0028884016
## RBI          .
## Walks        .
## Years        0.0288289761
## CAtBat       .
## CHits        0.0001971247
## CHmRun       .
## CRuns        0.0006334115
## CRBI         .
## CWalks       .
## League       0.0134641522
## Division     .
## PutOuts      0.0002929733
## Assists      .
## Errors      -0.0084074739
## NewLeague    .
```

Grab the field names and select the indices that are not zero. These are held in the value "coefs@i"

```
hitters.names <- names(Hitters)
lasso.indices <- coefs@i
hitters.names[coefs@i]
```

```
## [1] "Hits"    "HmRun"   "Runs"    "Years"   "CHits"   "CRuns"   "League"
## [8] "PutOuts" "Errors"
```

Here are the predictors Lasso doesn't use.

```
hitters.names[-coefs@i]
```

```
##  [1] "AtBat"     "RBI"       "Walks"     "CAtBat"    "CHmRun"    "CRBI"
##  [7] "CWalks"    "Division"  "Assists"   "Salary"    "NewLeague"
```