STA 4320 CHAP 6.2

Prof. He Jiang

```
require(ISLR2) # Hitters dataset
```

```
## Loading required package: ISLR2
```

```
require(leaps) # subset selection
```

```
## Loading required package: leaps
```

```
require(glmnet) # ridge, lasso
```

```
## Loading required package: glmnet
```

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

```
## Loaded glmnet 4.1-8
```

Hitters dataset and NA terms

The Hitters dataset consists of Major League Baseball data from the 1986 and 1987 seasons.

We remove the NA terms.

```
dat = na.omit(Hitters)
any(is.na(dat))
```

```
## [1] FALSE
```

6.2

We build the x matrix and the y vector.

```
# glmnet is from the glmnet package
# the -1 removes the column of 1s
x = model.matrix(Salary ~ ., dat)[, -1]
y = dat$Salary
```

Note that the model.matrix() function produces a matrix corresponding to the 19 predictors (where Salary is excluded). It also automatically transforms any qualitative variables into indicator variables.

Ridge regression

When alpha = 0, we fit a ridge regression.

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

If not specified, the grid will be determined automatically by the software.

Supply a decreasing sequence of lambda values will help with the computation.

Notice also the absence of the ~ sign.

Standardizing has been set to the default here. We can set standardize = FALSE if we do not want to standardize.

To acquire the coefficients, we use the coef command

```
dim( coef(ridge_mod) )
```

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

Here the 20 rows correspond to the 20 variables. The 100 columns correspond to the 100 lambda values.

Looking at lambda = 11497.57 (50th location).

```
ridge_mod$lambda[50]
```

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

```
# The coefficient should have small absolute values.
coef(ridge_mod)[, 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
```

```
# In fact, we can compute the L2 norm for the coefficients,
# with the intercept excluded
sqrt( sum( ( coef(ridge_mod)[-1, 50] )^2 ) )
```

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

Looking at lambda = 705.4802 (60th location).

```
ridge_mod$lambda[60]
```

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

```
# The coefficient should have larger than before absolute values.
coef(ridge_mod)[, 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
```

```
# In fact, we can compute the L2 norm for the coefficients,
# with the intercept excluded
sqrt( sum( ( coef(ridge_mod)[-1, 60] )^2 ) )
```

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

For predicting regression coefficients for a new lambda, we use predict() like in the lm command.

```
# prediction with a new lambda = 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
```

```
# the 1:20 here is to make the output look nicer in a row format
```

Training validation split and cross validation

We first split the data into two parts of roughly equal sizes. This can be done using a sampling of TRUE/FALSE (like in the previous code), or can be done by directly sampling the row indices.

```
set.seed(1)
train = sample(1:nrow(x), nrow(x) / 2)
# the following removes training row indices to form the validation vector
test = (-train)
y_test = y[test]
```

For example, when lambda = 4.

```
# thresh is a threshold values used to determine when to stop the algorithm
ridge_mod = glmnet(x[train, ], y[train], alpha = 0, lambda = grid, thresh = 1e-12)
# s is the new lambda value
ridge_pred = predict(ridge_mod, s = 4, newx = x[test, ])
mean( (ridge_pred - y_test)^2 )
```

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

As another example, when lambda = 1e10 (very large value), we are equivalently fitting a model with only the intercept (as the intercept is not in the constraint).

```
ridge_pred = predict(ridge_mod, s = 1e10, newx = x[test, ])
mean( (ridge_pred - y_test)^2 )
```

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

```
# compare to a model with only the intercept
mean( (mean(y[train]) - y_test)^2 )
```

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

When lambda = 0, we should have the regular least squares regression.

```
ridge_pred = predict(ridge_mod, s = 0, newx = x[test, ],
                     exact = TRUE,
                     x = x[train, ], y = y[train])
# the exact = TRUE here yields least squares coefficients for lambda = 0
mean( (ridge_pred - y_test)^2 )
```

```
## [1] 168588.6
```

```
# compare to least squares
lm(y ~ x, subset = train)
```

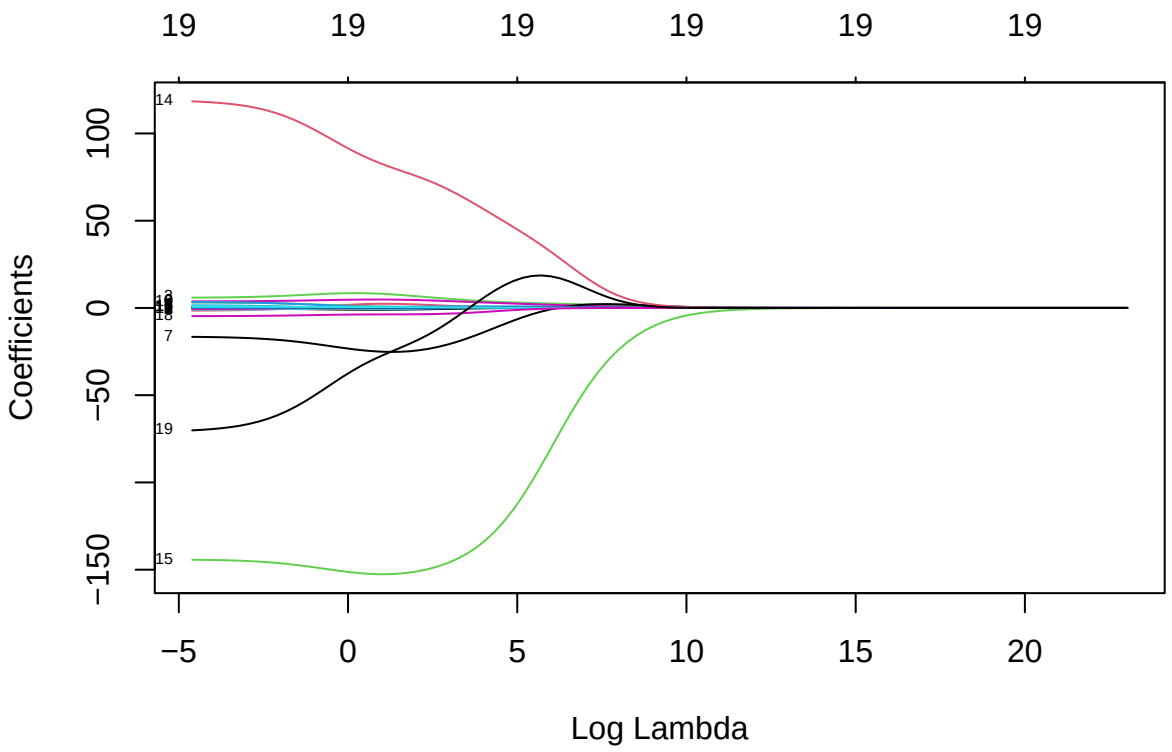
```
##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
## (Intercept)       xAtBat        xHits       xHmRun        xRuns         xRBI
##    274.0145      -0.3521      -1.6377       5.8145       1.5424       1.1243
##      xWalks       xYears      xCAtBat       xCHits      xCHmRun       xCRuns
##      3.7287     -16.3773      -0.6412       3.1632       3.4008      -0.9739
##       xCRBI      xCWalks     xLeagueN   xDivisionW     xPutOuts     xAssists
##     -0.6005       0.3379     119.1486    -144.0831       0.1976       0.6804
##     xErrors  xNewLeagueN
##     -4.7128     -71.0951
```

```
# ridge regression with lambda = 0
# note that the 1:20 is for formatting
predict(ridge_mod, s = 0, exact = TRUE, type = "coefficients",
        x = x[train, ], y = y[train])[1:20, ]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  274.0200994   -0.3521900   -1.6371383    5.8146692    1.5423361    1.1241837
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##    3.7288406  -16.3795195   -0.6411235    3.1629444    3.4005281   -0.9739405
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   -0.6003976    0.3378422 119.1434637 -144.0853061    0.1976300    0.6804200
##       Errors   NewLeagueN
##   -4.7127879  -71.0898914
```

Ridge regression plot vs lambda.

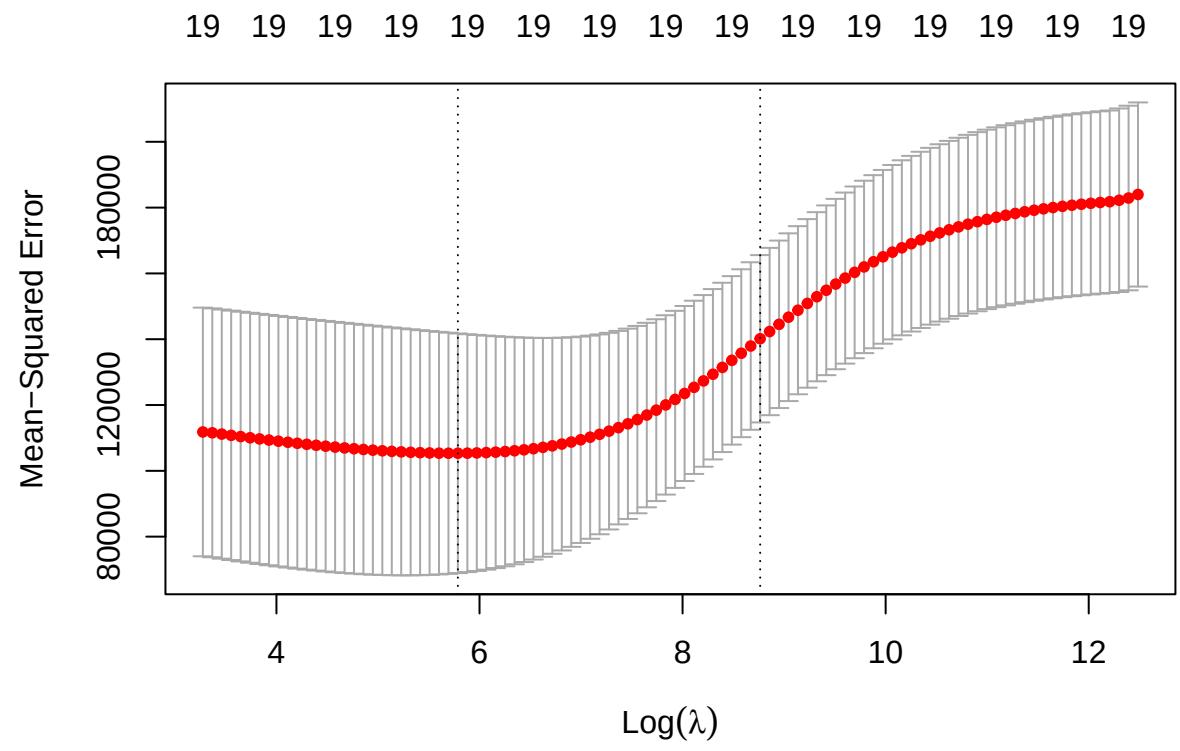
```
plot(ridge_mod, "lambda", label = TRUE)
```



We then use cross validation to examining the choice of lambda.

The cv.glmnet performs (10 fold) cross validation.

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



```
best_lam = cv_out$lambda.min
best_lam
```

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

Using the previous training validation split, we can find the MSE for this best lambda.

```
ridge_pred = predict(ridge_mod, s = best_lam, newx = x[test, ])
mean( (ridge_pred - y_test)^2 )
```

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

Once the best lambda has been determined (by cross validation), we use it to fit the model on the entire dataset.

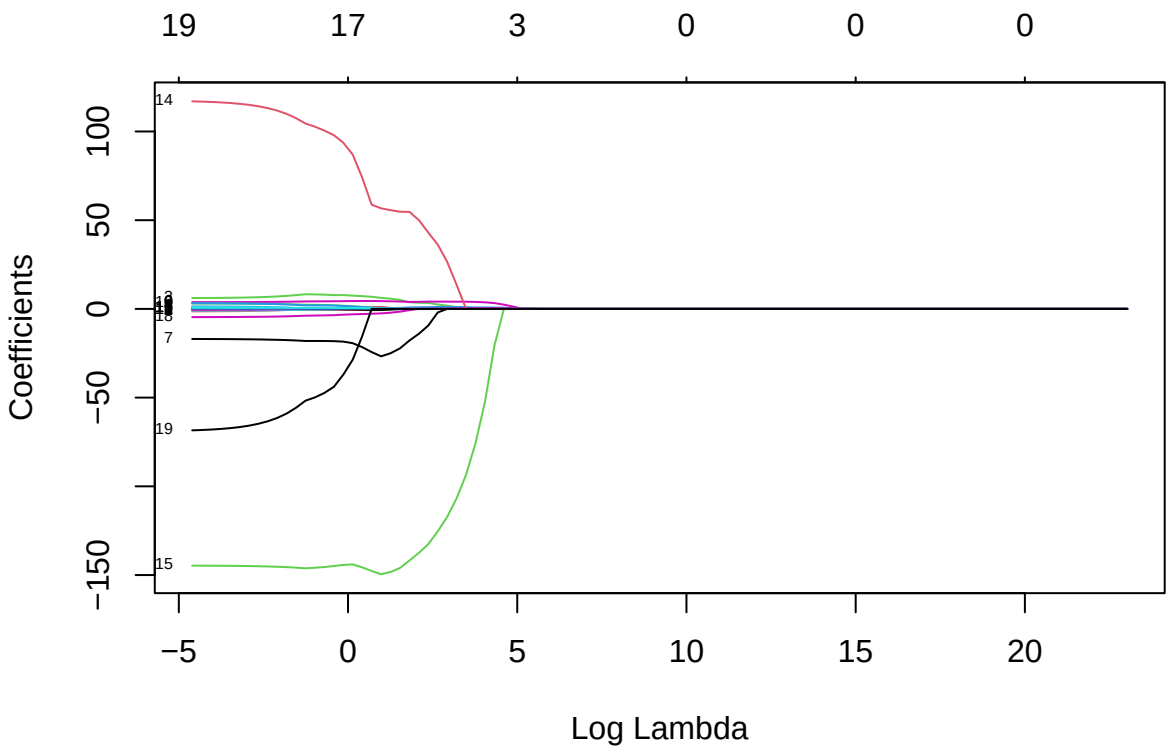
```
out = glmnet(x, y, alpha = 0)
predict(out, type = "coefficients", s = best_lam)[1:20, ]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  15.44383120   0.07715547   0.85911582   0.60103106   1.06369007   0.87936105
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##   1.62444617   1.35254778   0.01134999   0.05746654   0.40680157   0.11456224
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   0.12116504   0.05299202  22.09143197 -79.04032656   0.16619903   0.02941950
##       Errors   NewLeagueN
##  -1.36092945   9.12487765
```

Lasso

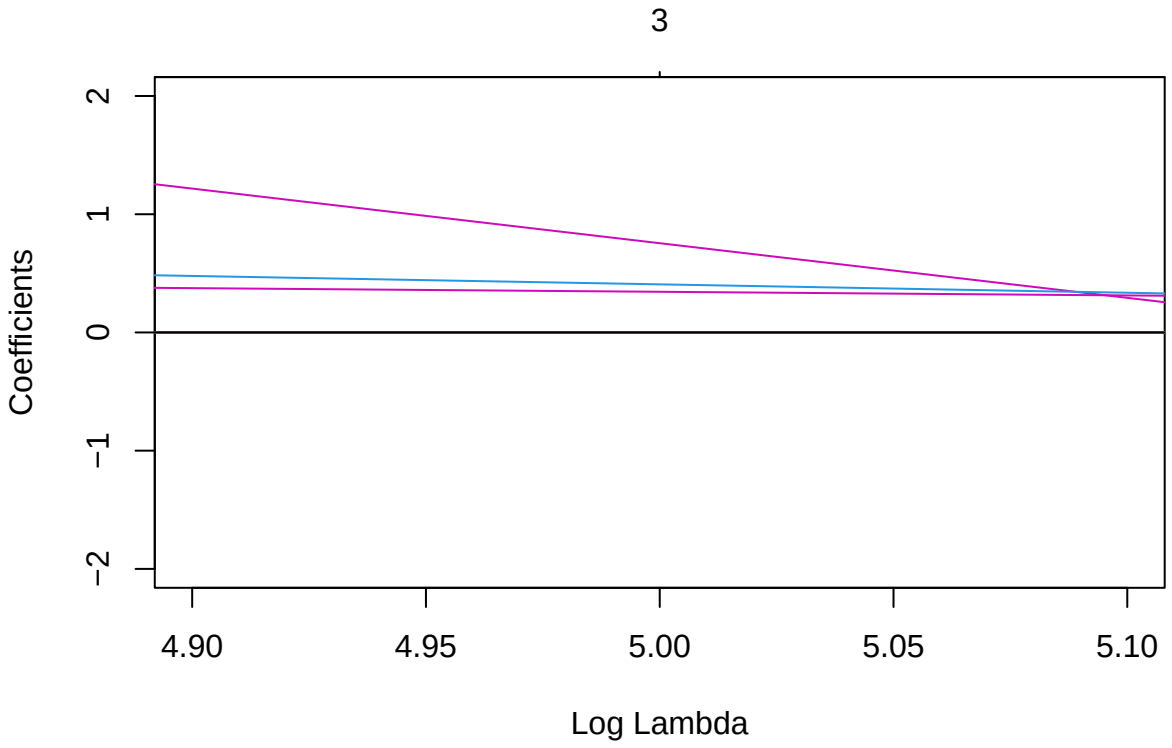
The lasso uses the same glmnet() function, with the change that here alpha = 1.

```
lasso_mod = glmnet(x[train, ], y[train], alpha = 1, lambda = grid)
plot(lasso_mod, "lambda", label = TRUE)
```



We can also Zoom in, for example to the part where log(lambda) = 5

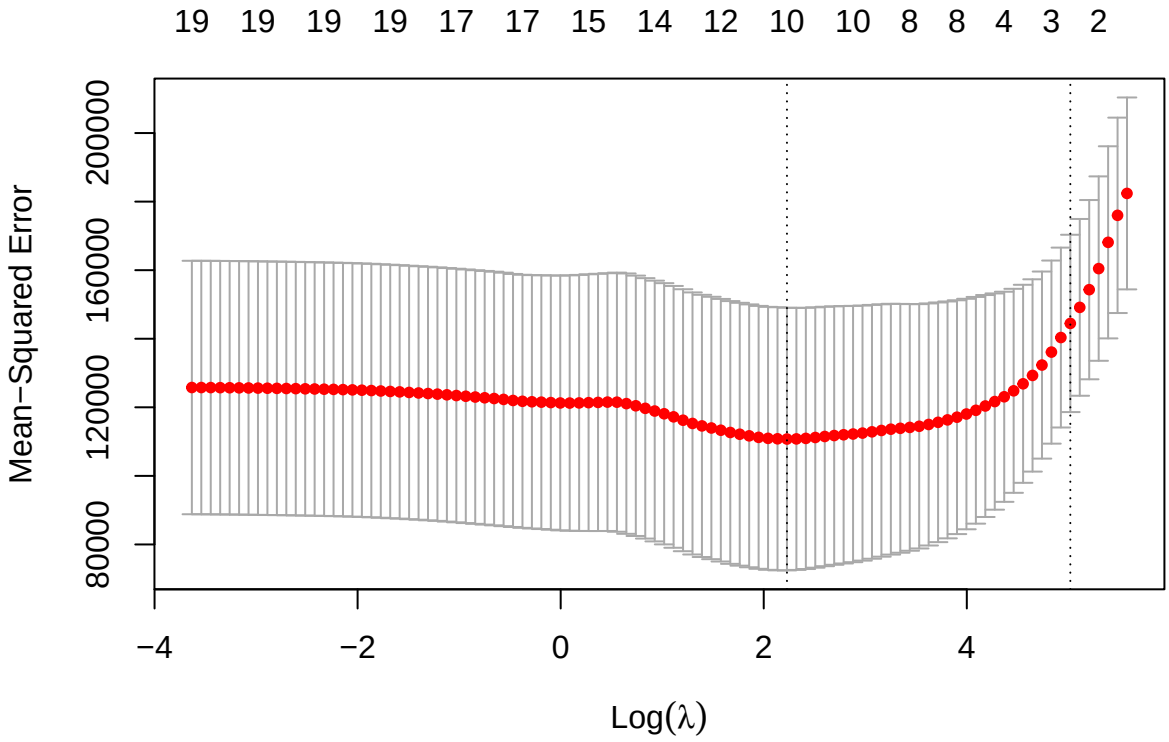
```
plot(lasso_mod, "lambda", label=TRUE, xlim = c(4.9, 5.1), ylim = c(-2, 2))
```



Doing cross-validation for lasso.

Note that in the plot, the first dotted line is the lambda that minimizes the test MSE.

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



The cross-validation and computed test error.

Note this error is significantly lower than that of regular least squares regression (of 168588.6).

```
best_lam = cv_out$lambda.min
lasso_pred = predict(lasso_mod, s = best_lam, newx = x[test, ])
mean((lasso_pred - y_test)^2)
```

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

Finally, we see the lasso coefficients computed on the entire data.

```
out = glmnet(x, y, alpha = 1, lambda = grid)
lasso_coef = predict(out, type = "coefficients", s = best_lam)[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
```

Note that the lasso sets many coefficients to 0. We can see only the remaining non-zero coefficients.

```
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
```