Ridge and The LASSO

Note: The example contained herein was copied from the lab exercise in Chapter 6 of Introduction to Statistical Learning by Daniela Witten, Trevor Hastie, Gareth M. James, Robert Tibshirani.

For this exercise, we'll use some baseball data from 1986-1987. The Hitters dataset contains information about 322 baseball players and 20 attributes as follows:

1. AtBat: Number of times at bat in 1986

2. Hits: Number of hits in 1986

3. HmRun: Number of home runs in 1986

4. Runs: Number of runs in 1986

5. RBI: Number of runs batted in in 1986

6. Walks: Number of walks in 1986

7. Years: Number of years in the major leagues

8. CAtBat: Number of times at bat during his career

9. CHits: Number of hits during his career

10. CHmRun: Number of home runs during his career

11. CRuns: Number of runs during his career

12. CRBI: Number of runs batted in during his career

13. CWalks: Number of walks during his career

14. League: A factor with levels A and N indicating player's league at the end of 1986

15. Division: A factor with levels E and W indicating player's division at the end of 1986

16. PutOuts: Number of put outs in 1986

17. Assists: Number of assists in 1986

18. Errors: Number of errors in 1986

19. Salary: 1987 annual salary on opening day in thousands of dollars

20. NewLeague: A factor with levels A and N indicating player's league at the beginning of 1987

This dataset is available in the ISLR library. The target variable of interest here is the players' salaries. The target variable is missing for 59 of the players, thus we must omit those players from our analysis.

```
> library(leaps) #stepwise selection
> library(glmnet) #lasso/ridge/elastic net
> load("Hitters.Rdata")
> Hitters = na.omit(Hitters)   # omit rows with missing values
```

Stepwise Selection Methods

We'll start by looking at forward backward selection using the leaps library and the regsubsets command. The task is to pick the *size/complexity* of our model. The best way to do this is to use validation or cross-validation. Cross-validation gets a little confusing in this context, so we will keep the computation simpler and just use a single validation dataset. To create the index vectors for the training and test datasets, we'll create a random logical vector which selects true values (for the training set) 60% of the time.

```
> set.seed(7515)
> train=sample(c(T,F), nrow(Hitters), rep=TRUE, p=c(0.6,0.4))
> test=!train
> regfit.fwd = regsubsets(Salary ~ ., data=Hitters[train,], nvmax=19, method="forward")
> regfit.bwd = regsubsets(Salary ~ ., data=Hitters[train,], nvmax=19, method="backward")
> summary(regfit.fwd)
```

```
Subset selection object
Call: regsubsets.formula(Salary ~ ., data = Hitters[train, ], nvmax = 19,
    method = "forward")
19 Variables  (and intercept)
          Forced in Forced out
AtBat         FALSE      FALSE
Hits          FALSE      FALSE
HmRun         FALSE      FALSE
Runs          FALSE      FALSE
RBI           FALSE      FALSE
Walks         FALSE      FALSE
Years         FALSE      FALSE
CAtBat        FALSE      FALSE
CHits         FALSE      FALSE
CHmRun        FALSE      FALSE
CRuns         FALSE      FALSE
CRBI          FALSE      FALSE
CWalks        FALSE      FALSE
LeagueN       FALSE      FALSE
DivisionW     FALSE      FALSE
PutOuts       FALSE      FALSE
Assists       FALSE      FALSE
Errors        FALSE      FALSE
NewLeagueN    FALSE      FALSE
1 subsets of each size up to 19
```

```
Selection Algorithm: forward
          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    "*"   " "
2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    "*"   " "
3  ( 1 )  " "   "*"  " "   " "  " " " "   " "   "*"    " "   " "    "*"   " "
4  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   "*"    " "   " "    "*"   " "
5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   "*"    " "   " "    "*"   " "
6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   " "
7  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   " "
8  ( 1 )  "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   " "
9  ( 1 )  "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   " "
10 ( 1 )  "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   "*"    "*"   " "
11 ( 1 )  "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   "*"    "*"   "*"
12 ( 1 )  "*"   "*"  " "   "*"  " " "*"   "*"   "*"    " "   "*"    "*"   "*"
13 ( 1 )  "*"   "*"  " "   "*"  " " "*"   "*"   "*"    " "   "*"    "*"   "*"
14 ( 1 )  "*"   "*"  " "   "*"  " " "*"   "*"   "*"    " "   "*"    "*"   "*"
15 ( 1 )  "*"   "*"  " "   "*"  " " "*"   "*"   "*"    " "   "*"    "*"   "*"
16 ( 1 )  "*"   "*"  "*"   "*"  " " "*"   "*"   "*"    " "   "*"    "*"   "*"
17 ( 1 )  "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    " "   "*"    "*"   "*"
18 ( 1 )  "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    " "   "*"    "*"   "*"
19 ( 1 )  "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"
          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
1  ( 1 )  " "    " "     " "       " "     " "     " "    " "
2  ( 1 )  " "    " "     " "       " "     " "     " "    " "
3  ( 1 )  " "    " "     " "       " "     " "     " "    " "
4  ( 1 )  " "    " "     " "       " "     " "     " "    " "
5  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
6  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
7  ( 1 )  "*"    " "     " "       "*"     " "     " "    " "
8  ( 1 )  "*"    " "     " "       "*"     " "     " "    " "
9  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
10 ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
11 ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
12 ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
13 ( 1 )  "*"    " "     "*"       "*"     " "     "*"    " "
14 ( 1 )  "*"    " "     "*"       "*"     "*"     "*"    " "
15 ( 1 )  "*"    "*"     "*"       "*"     "*"     "*"    " "
16 ( 1 )  "*"    "*"     "*"       "*"     "*"     "*"    " "
17 ( 1 )  "*"    "*"     "*"       "*"     "*"     "*"    " "
18 ( 1 )  "*"    "*"     "*"       "*"     "*"     "*"    "*"
19 ( 1 )  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

> summary(regfit.bwd)

```
Subset selection object
Call: regsubsets.formula(Salary ~ ., data = Hitters[train, ], nvmax = 19,
    method = "backward")
19 Variables  (and intercept)
         Forced in Forced out
AtBat        FALSE      FALSE
Hits         FALSE      FALSE
HmRun        FALSE      FALSE
Runs         FALSE      FALSE
RBI          FALSE      FALSE
Walks        FALSE      FALSE
Years        FALSE      FALSE
CAtBat       FALSE      FALSE
CHits        FALSE      FALSE
```

```
CHmRun          FALSE           FALSE
CRuns           FALSE           FALSE
CRBI            FALSE           FALSE
CWalks          FALSE           FALSE
LeagueN         FALSE           FALSE
DivisionW       FALSE           FALSE
PutOuts         FALSE           FALSE
Assists         FALSE           FALSE
Errors          FALSE           FALSE
NewLeagueN      FALSE           FALSE
1 subsets of each size up to 19
Selection Algorithm: backward
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI
1 (1)	" "	" "	" "	" "	" "	" "	" "	" "	" "	" "	"*"	" "
2 (1)	" "	"*"	" "	" "	" "	" "	" "	" "	" "	" "	"*"	" "
3 (1)	"*"	"*"	" "	" "	" "	" "	" "	" "	" "	" "	"*"	" "
4 (1)	"*"	"*"	" "	" "	" "	"*"	" "	" "	" "	" "	"*"	" "
5 (1)	"*"	"*"	" "	" "	" "	"*"	" "	" "	" "	" "	"*"	" "
6 (1)	"*"	"*"	" "	" "	" "	"*"	" "	" "	" "	" "	"*"	" "
7 (1)	"*"	"*"	" "	" "	" "	"*"	" "	" "	" "	"*"	"*"	" "
8 (1)	"*"	"*"	" "	" "	" "	"*"	" "	" "	" "	"*"	"*"	"*"
9 (1)	"*"	"*"	" "	"*"	" "	"*"	" "	" "	" "	"*"	"*"	"*"
10 (1)	"*"	"*"	" "	"*"	" "	"*"	"*"	" "	" "	"*"	"*"	"*"
11 (1)	"*"	"*"	" "	"*"	" "	"*"	"*"	" "	" "	"*"	"*"	"*"
12 (1)	"*"	"*"	" "	"*"	" "	"*"	"*"	" "	" "	"*"	"*"	"*"
13 (1)	"*"	"*"	" "	"*"	" "	"*"	"*"	" "	" "	"*"	"*"	"*"
14 (1)	"*"	"*"	" "	"*"	" "	"*"	"*"	" "	" "	"*"	"*"	"*"
15 (1)	"*"	"*"	" "	"*"	" "	"*"	"*"	"*"	" "	"*"	"*"	"*"
16 (1)	"*"	"*"	"*"	"*"	" "	"*"	"*"	"*"	" "	"*"	"*"	"*"
17 (1)	"*"	"*"	"*"	"*"	"*"	"*"	"*"	"*"	" "	"*"	"*"	"*"
18 (1)	"*"	"*"	"*"	"*"	"*"	"*"	"*"	"*"	" "	"*"	"*"	"*"
19 (1)	"*"	"*"	"*"	"*"	"*"	"*"	"*"	"*"	"*"	"*"	"*"	"*"

	CWalks	LeagueN	DivisionW	PutOuts	Assists	Errors	NewLeagueN
1 (1)	" "	" "	" "	" "	" "	" "	" "
2 (1)	" "	" "	" "	" "	" "	" "	" "
3 (1)	" "	" "	" "	" "	" "	" "	" "
4 (1)	" "	" "	" "	" "	" "	" "	" "
5 (1)	"*"	" "	" "	" "	" "	" "	" "
6 (1)	"*"	" "	" "	"*"	" "	" "	" "
7 (1)	"*"	" "	" "	"*"	" "	" "	" "
8 (1)	"*"	" "	" "	"*"	" "	" "	" "
9 (1)	"*"	" "	" "	"*"	" "	" "	" "
10 (1)	"*"	" "	" "	"*"	" "	" "	" "
11 (1)	"*"	" "	"*"	"*"	" "	" "	" "
12 (1)	"*"	" "	"*"	"*"	" "	"*"	" "
13 (1)	"*"	" "	"*"	"*"	"*"	"*"	" "
14 (1)	"*"	"*"	"*"	"*"	"*"	"*"	" "
15 (1)	"*"	"*"	"*"	"*"	"*"	"*"	" "
16 (1)	"*"	"*"	"*"	"*"	"*"	"*"	" "
17 (1)	"*"	"*"	"*"	"*"	"*"	"*"	" "
18 (1)	"*"	"*"	"*"	"*"	"*"	"*"	"*"
19 (1)	"*"	"*"	"*"	"*"	"*"	"*"	"*"

The summaries tell us which variables show up in the best p-variable models where p ranges from 1 input to all 19 inputs. This spacious output can be hard to read, but we can use the *vorder* output vector to tell us in what order the variables the model should contain if we allow p variables into the model. You can easily see that each selection method provided us with different models, no matter what number of variables we choose to allow.

```
> regfit.fwd$vorder
```

```
[1]  1 12  3  9  2 17  7 14  5 16 11 13  8 19 18 15  4  6 20 10
```

```
> regfit.bwd$vorder
```

```
[1]  1 12  3  2  7 14 17 11 13  5  8 16 19 18 15  9  4  6 20 10
```

We can get the actual coefficients from the chosen models using the coef() function and specifying the number of variables we want:

```
> coef(regfit.fwd, 7)
```

```
(Intercept)        AtBat         Hits        Walks        CAtBat         CRuns
 49.1354867   -2.5488907    8.8952359    6.5829602   -0.1290065     2.1072678
     CWalks      PutOuts
 -0.7728877    0.2119603
```

```
> coef(regfit.bwd, 7)
```

```
(Intercept)        AtBat         Hits        Walks        CHmRun         CRuns
 17.9102884   -2.8964270    9.9139855    8.4739365    0.8614003     1.0895680
     CWalks      PutOuts
 -0.9142449    0.1631992
```

Since the regsubsets function only outputs a set of coefficients for each value of p, we need a model matrix to compute the predictions, $\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$, on the test data. The model.matrix command makes this easy for us. Keep in mind that matrix multiplication in R is done using the %*% command, and that we only want to score the test data.

```
> X = model.matrix(Salary~., data=Hitters[test, ])
> fwd.val.MSE=vector()
> bwd.val.MSE=vector()
> for (i in 1:19) {
+    beta.fwd = coef(regfit.fwd, i)
+    beta.bwd = coef(regfit.bwd, i)
+    pred.fwd = X[,names(beta.fwd)] %*% beta.fwd
+    pred.bwd = X[,names(beta.bwd)] %*% beta.bwd
+    fwd.val.MSE[i] = mean((Hitters$Salary[test] - pred.fwd)^2)
+    bwd.val.MSE[i] = mean((Hitters$Salary[test] - pred.bwd)^2)
+ }
> min(fwd.val.MSE)
```

```
[1] 137203.1
```

```
> min(bwd.val.MSE)
```

```
[1] 137510.2
```

Now that we have the validation MSE for each number of parameters and each selection technique, we simply choose the number of parameters that minimizes the validation MSE for each selection method.

```
> plot(1:19,fwd.val.MSE,
+       type='b',
+       pch=16,
+       col='magenta',
+       xlab="Number of Inputs",
+       ylab = "Validation MSE",
+       main = "Forward Selection Results"
+ )
>       abline(v = which.min(fwd.val.MSE), col='blue')
```

Forward Selection Results
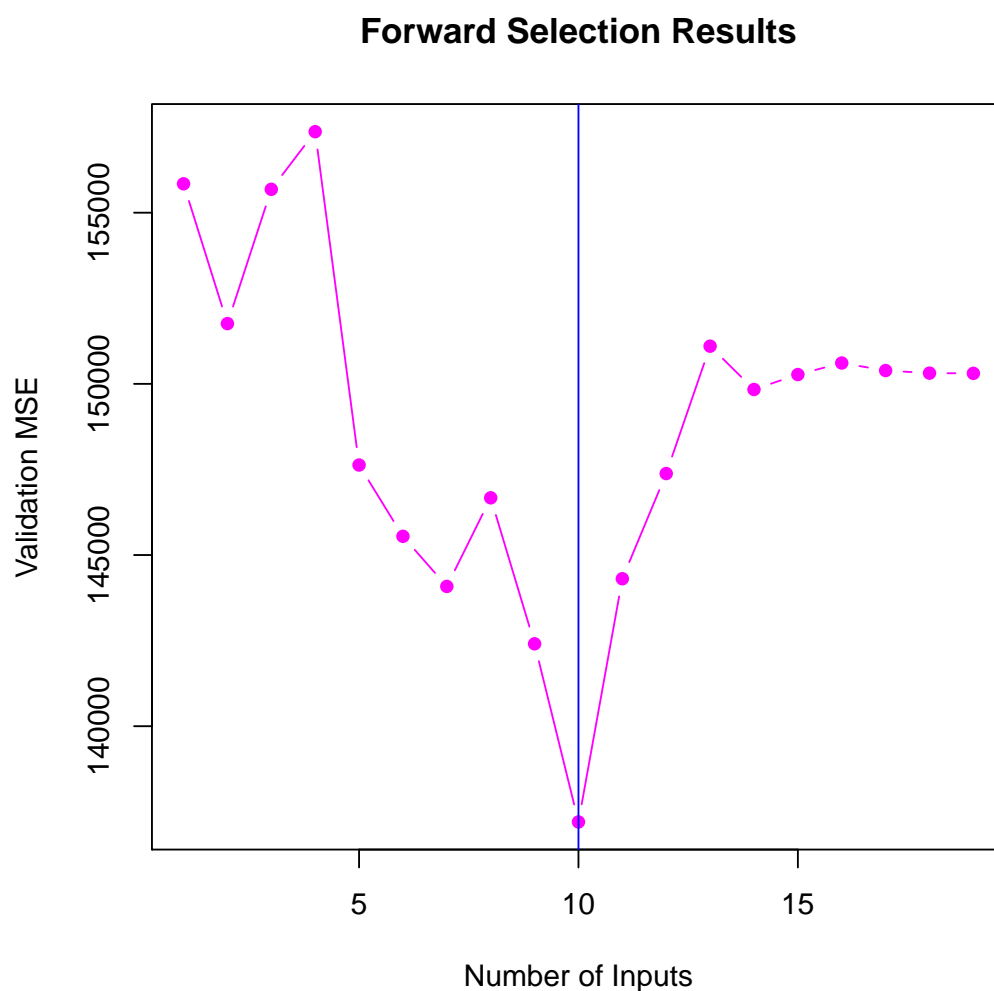

```
> plot(1:19,bwd.val.MSE,
+       type='b',
+       pch=16,
+       col='magenta',
+       xlab="Number of Inputs",
+       ylab = "Validation MSE",
+       main = "Backward Selection Results"
+ )
>       abline(v = which.min(bwd.val.MSE), col='blue')
```

Backward Selection Results
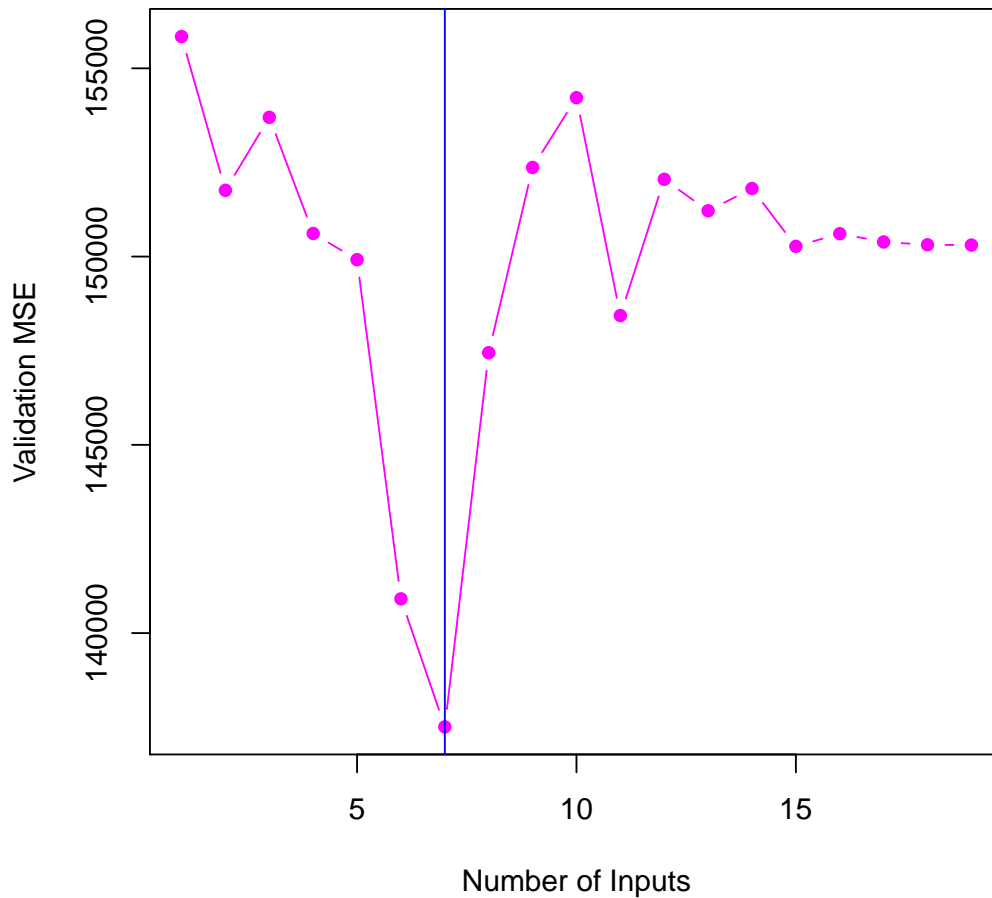
The two graphs are quite different. Forward selection chooses a 10 variable model whereas backward selection chooses a 7 variable model. We can look at *which* variables were chosen with the coef() function:

```
> coef(regfit.fwd,10)
```

(Intercept)	AtBat	Hits	Runs	Walks	CAtBat
93.9840930	-2.5569014	11.1792265	-5.4583086	8.0172922	-0.1594122
CHmRun	CRuns	CWalks	DivisionW	PutOuts	
0.7415922	2.2277955	-0.8530738	-64.3153622	0.1866306	

```
> coef(regfit.bwd,7)
```

(Intercept)	AtBat	Hits	Walks	CHmRun	CRuns
17.9102884	-2.8964270	9.9139855	8.4739365	0.8614003	1.0895680
CWalks	PutOuts				
-0.9142449	0.1631992				

In order to finalize the coefficients for the chosen models, we'd re-run the variable selection process on the entire dataset using only the chosen number of variables.

```
> regfit.fwd.best = regsubsets(Salary~., data=Hitters, method="forward", nvmax=10)
> regfit.bwd.best = regsubsets(Salary~., data=Hitters, method="backward", nvmax=7)
> coef(regfit.fwd.best,10)
```

(Intercept)	AtBat	Hits	Walks	CAtBat	CRuns
162.5354420	-2.1686501	6.9180175	5.7732246	-0.1300798	1.4082490
CRBI	CWalks	DivisionW	PutOuts	Assists	
0.7743122	-0.8308264	-112.3800575	0.2973726	0.2831680	

```
> coef(regfit.bwd.best,7)
```

(Intercept)	AtBat	Hits	Walks	CRuns	CWalks
105.6487488	-1.9762838	6.7574914	6.0558691	1.1293095	-0.7163346
DivisionW	PutOuts				
-116.1692169	0.3028847				

Ridge Regression

For the sake of exploration, let's see what type of results Ridge Regression will provide on our validation data. We'll first tune the shrinkage parameter, λ, using cross-validation because this is standard practice. This will be done using the glmnet package and glmnet() function. This package has built in capabilities for doing a cross-validation loop which makes it convenient for our purposes. The glmnet() function wants the user to specify a model matrix **X** (design matrix) and a vector containing the target variable **y**, it does not use the y~x syntax that many other modeling packages use.

```
> X=model.matrix(Salary~. ,data=Hitters)[,-1]
> y = Hitters$Salary
```

We'll now perform cross-validation to determine an optimal value of λ. We'll only use the training data for the cross-validation step. We can then test the ridge regression model with this lambda on our validation dataset from the previous section. The option alpha=0 specifies ridge regression. We will change this to alpha=1 to implement the LASSO.

```
> set.seed(1)
> cv.out = cv.glmnet(X[train,], y[train], alpha=0)
> plot(cv.out)
```
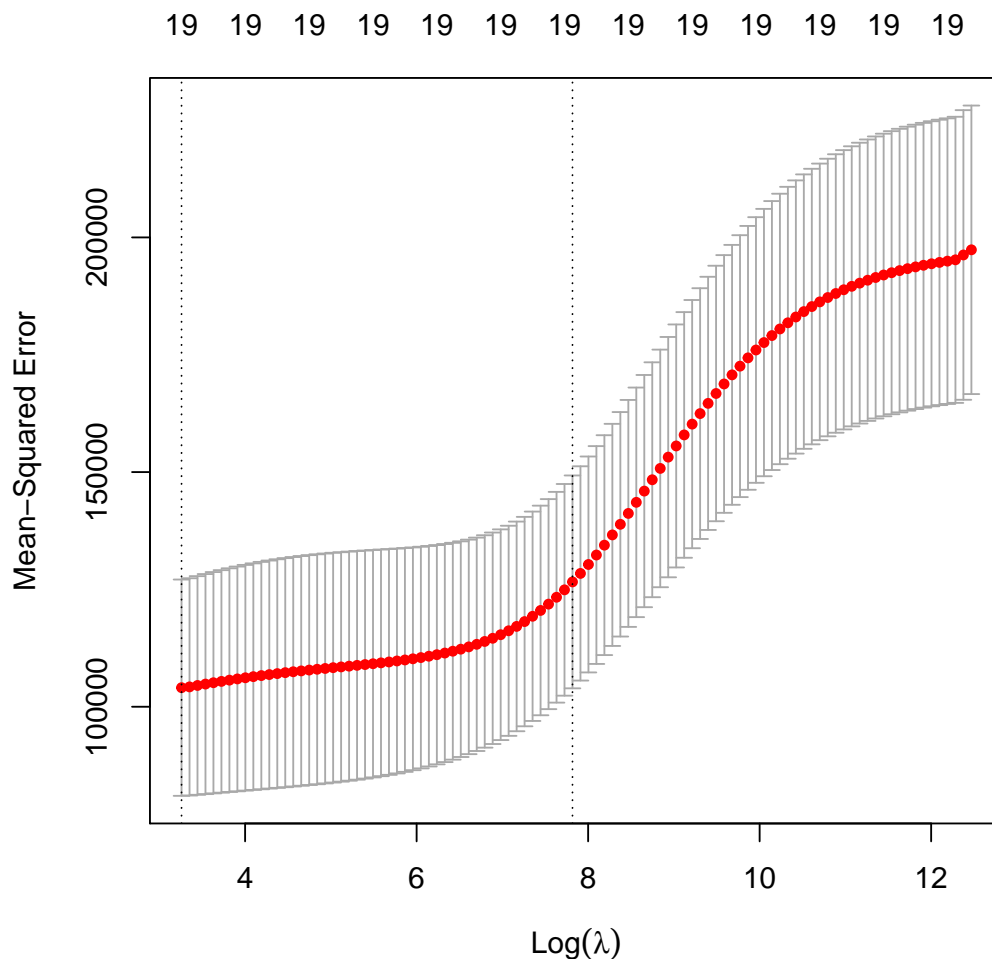
The method will choose the value of λ that has the best performance on cross-validation. We can grab that value from the output, create a ridge model using it and then check the performance of the ridge regression model on our out-of-sample test data used previously.

```
> bestlambda=cv.out$lambda.min
> bestlambda
```

```
[1] 26.01949
```

```
> ridge.mod = glmnet(X[train,], y[train], alpha=0, lambda=bestlambda)
> pred.ridge = predict(ridge.mod, newx=X[test,])
> val.MSE.ridge = mean((pred.ridge - y[test])^2)
> val.MSE.ridge
```

```
[1] 131187.9
```

Notice that the validation MSE for this ridge regression is a dramatic reduction from the MSE of the full model in the original least-squares context. Just take a look at the validation MSE graphs from forward/backward selection when the number of inputs is p>9. This ridge regression model does contain

all 19 predictor variables, but still performs well on out-of-sample data. That's quite a feat! In certain situations, it is not reasonable to keep all of the predictor variables in the model, but when it is feasible, ridge regression is an excellent alternative to classical least-squares.

To get the final parameter estimates for the ridge regression model, we'd use the full data:

```
> out=glmnet(X,y,alpha=0, lambda=bestlambda)
> ridge.coef = predict(out, type="coefficients")
> ridge.coef
```

```
20 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)   79.33662716
AtBat         -0.67098359
Hits           2.74575818
HmRun         -1.34701139
Runs           1.03258681
RBI            0.71845486
Walks          3.35113396
Years         -8.88376724
CAtBat        -0.00024648
CHits          0.13341047
CHmRun         0.68080619
CRuns          0.28687848
CRBI           0.25962713
CWalks        -0.27127440
LeagueN       52.98475229
DivisionW   -122.77762704
PutOuts        0.26343208
Assists        0.16767503
Errors        -3.67865103
NewLeagueN   -17.86087278
```

The LASSO

If we wanted to use the LASSO in a context similar to forward selection, we'd probably start by looking at the following plot, which shows how the coefficients in the model change as the shrinkage parameter λ declines. When λ is very large, we are prohibited from adding any parameters to the model. We start with the intercept-only null model. As λ gets smaller (moving to the right of the plot) we see that we can start allowing variables to enter the model if they will drive down the sum of squared error. In this way, LASSO acts as a selection technique, with one variable entering the model at a time.

Again, to run LASSO, we use the glmnet() function with the option alpha=1. We can tune the shrinkage parameter λ using cross-validation on the training data, and then compare the resulting model on our validation data.

```
> set.seed(1)
> cv.out=cv.glmnet(X[train,],y[train],alpha=1)
> plot(cv.out)
```
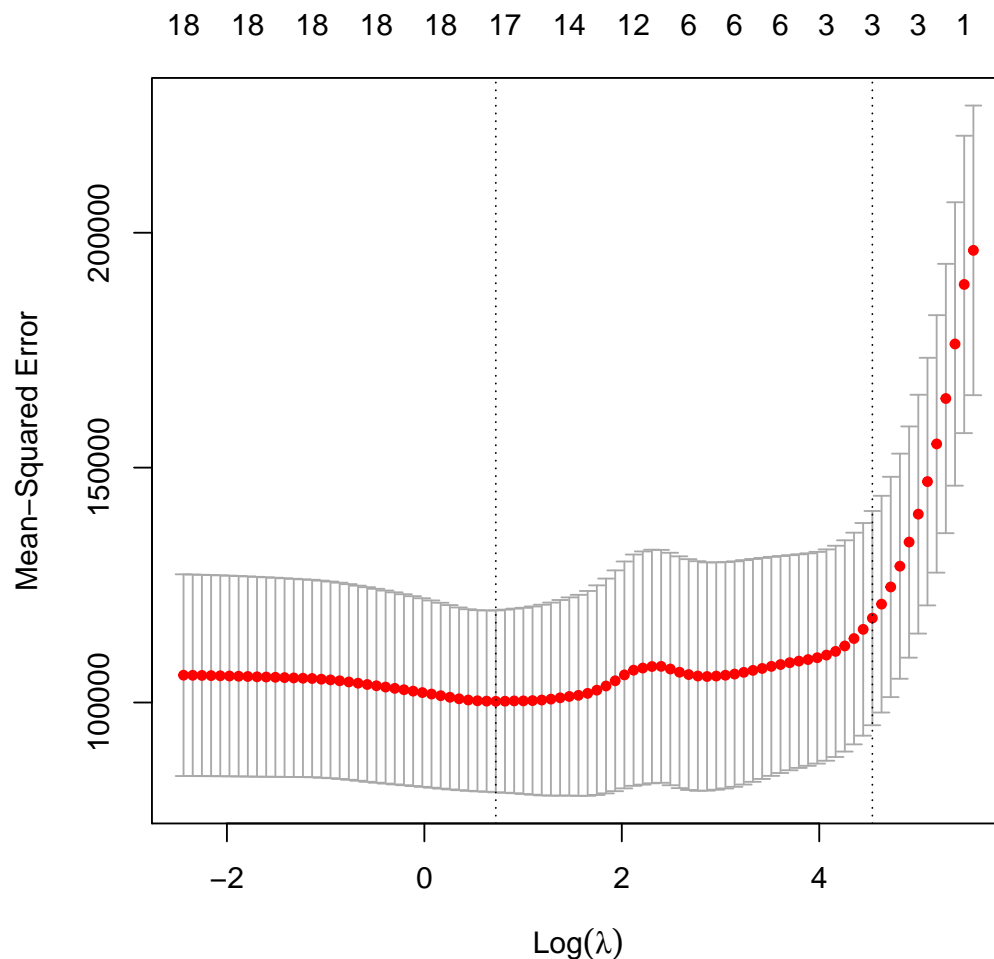
We can then use the information from the output dataset to obtain the best lambda and predict the salaries of hitters from the out-of-sample validation data, computing the MSE as before.

```
> bestlambda=cv.out$lambda.min
> pred.lasso = predict(cv.out, s=bestlambda, newx=X[test,])
> val.MSE.lasso = mean((pred.lasso-y[test])^2)
> val.MSE.lasso
```

```
[1] 137038
```

The advantage of the LASSO over ridge regression is that the resulting coefficient estimates are sparse. Below we see that 12 of the 19 coefficient estimates are exactly zero. So the LASSO model contains only 7 variables.

```
> out=glmnet(X,y,alpha=1,lambda=bestlambda)
> lasso.coef=predict(out, type="coefficients")
> lasso.coef
```

```
20 x 1 sparse Matrix of class "dgCMatrix"
                    s0
```

```
(Intercept)   1.379844e+02
AtBat        -1.729667e+00
Hits          6.053342e+00
HmRun         1.586088e-01
Runs              .
RBI               .
Walks         5.054177e+00
Years        -1.033180e+01
CAtBat       -7.831918e-04
CHits             .
CHmRun        5.607890e-01
CRuns         7.199190e-01
CRBI          3.905694e-01
CWalks       -6.019104e-01
LeagueN       3.330146e+01
DivisionW    -1.193222e+02
PutOuts       2.769324e-01
Assists       2.084560e-01
Errors       -2.336031e+00
NewLeagueN        .
```

You'll notice if you play with the random seed in the cross validation step, you may get results that look quite different. In fact, they are not as different as they seem - while so many coefficients may not be exactly zero, they are close enough to zero that omitting those variables from the model will not have such a drastic effect on the validation MSE. Once you get the final list of parameters, you may have to make a judgement call on which parameter estimates are large enough to include. An alternative approach is to use bestlambda=lambda.1se. The lambda.1se is the value of lambda that provided the simplest model but which has cross-validation error within 1 standard deviation of the minimum error. This value of lambda tends to provide results that are more stable across validation datasets. In our example, lambda.1se tends to produce models with only 5 variables.