# More Linear Regression Models

Subset Selection

Ridge Regression

Lasso

Principal Components Regression

Partial Least Squares Regression

# The Multiple Regression Model

The relationship between one dependent and two or more independent variables can be modeled using a linear function

**Y-intercept** **Slopes** **Random error**

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \ldots + \beta_k X_{ki} + \varepsilon_i$$

**Dependent (Response) variable** **Independent (Explanatory) variables**

# Why multiple regression is tricky

**Interactions (high correlations) among variables can mask true relationships (Multicollinearity)**

**We need a higher *n* to better estimate many coefficients**

**Slopes are estimates with high errors when multicollinear!**

**MANY organismal measurements are significantly correlated with each other!**

**Example:**
**Does it help to use tibia length and femur length in multiple regression to predict stature?**
**NOT USING OLS REGRESSION!**

# Multiple Linear Regression

**OLS - Ordinary Least Squares Regression**

Advantages

- Simpler models, even when multiple regression
- Low bias (**if *p* < *n***) (number of predictors < sample size)
- Fit can be acceptable even if less than ideal
- Predictive accuracy can be acceptable even if less than ideal

But there are other ways of fitting a straight line to data

**- necessary if *p* > *n* (OLS impossible) or *p* "near *n* (OLS has high variance)**

**- we can constrain or shrink coefficients to reduce variance (with some bias)**

# Multiple Linear Regression

**Other ways of fitting a linear model**

**Advantages**

Some predictors are not useful, so should be attenuated in model

- makes models easier to interpret
- in effect, feature selection/variable selection

**Subset selection**

Fit a model using the most valuable predictors

- similar to stepwise selection in classification

# Best Subset Selection

## Ideally, fit every combination of predictors

**Algorithm 6.1** *Best subset selection*

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.

2. For $k = 1, 2, \ldots p$:

   (a) Fit all $\binom{p}{k}$ models that contain exactly $k$ predictors.

   training sample (b) Pick the $\boxed{\text{best}}$ among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the $\boxed{\text{smallest RSS, or equivalently largest } R^2}$.

3. Select a $\boxed{\text{single best model}}$ from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, $\boxed{C_p \text{ (AIC), BIC, or adjusted } R^2}$.

**Problem:** Roughly $2^p$ models
Could be > 100,000 models!

**Problem**: RSS decreases, $R^2$ increases with more predictors

**Solution:** test samples

# The problems of RSS (MSE) and $R^2$: ML provides a solution

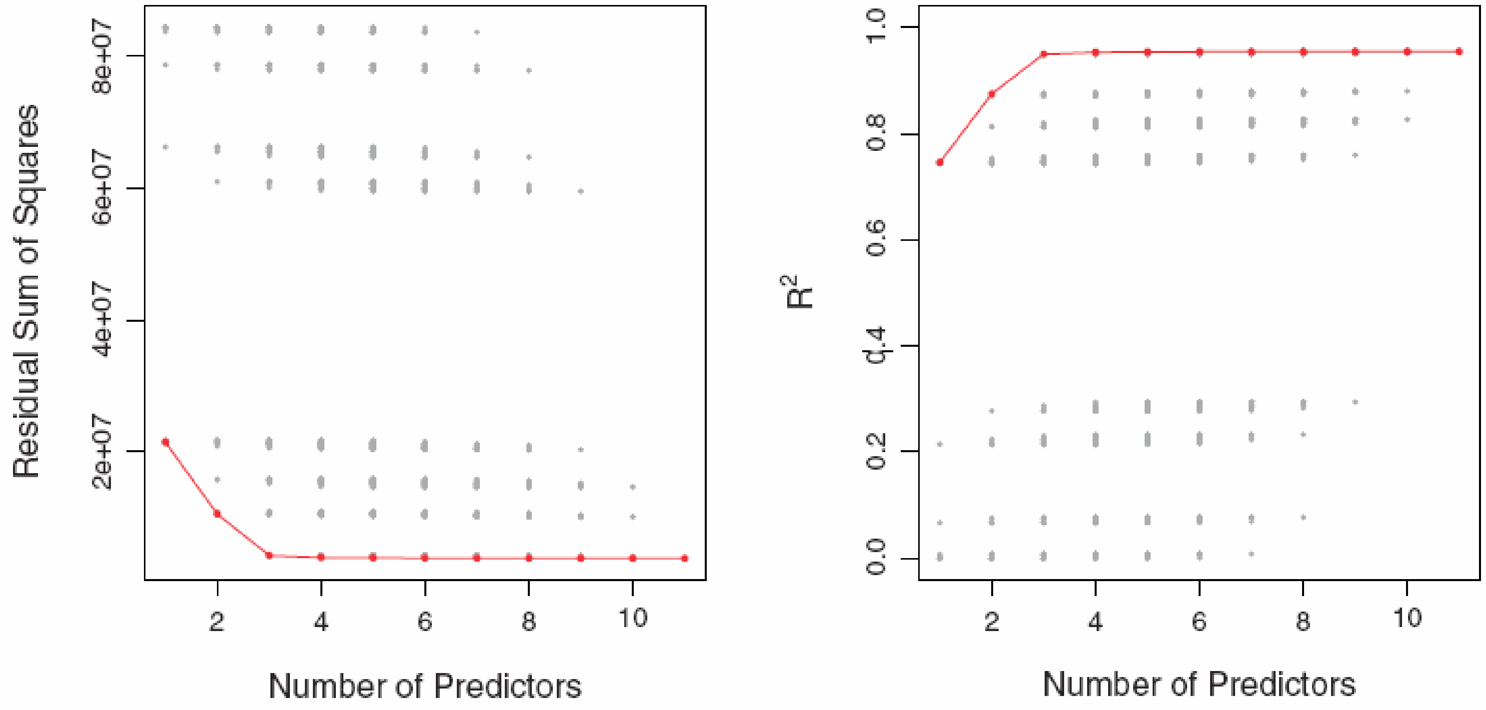



**FIGURE 6.1.** *For each possible model containing a subset of the ten predictors in the* Credit *data set, the RSS and $R^2$ are displayed. The red frontier tracks the* best *model for a given number of predictors, according to RSS and $R^2$. Though the data set contains only ten predictors, the $x$-axis ranges from 1 to 11, since one of the variables is categorical and takes on three values, leading to the creation of two dummy variables.*

# Which predictors estimate best?

# Stepwise Methods

## All possible subsets

find the best combination of *p* predictors that maximizes AIC/BIC/adusted $R^2$

8 of 15: 6,435 5 of 20: 15,504 10 of 20: 184,756

### Forward Selection

- add the best predictor one at a time (Maximum 240 models for 15 predictors)

### Backward Elimination

get rid of the worst predictors one at a time (smallest effect on RSS) **IFF** $n > p$

### Hybrid Stepwise (Combination of Forward and Backward)

# Stepwise Selection

**Results can be very similar**

- forward stepwise is ALWAYS doable and is ALWAYS faster

| # Variables | Best subset | Forward stepwise |
|---|---|---|
| One | rating | rating |
| Two | rating, income | rating, income |
| Three | rating, income, student | rating, income, student |
| Four | cards, income, student, limit | rating, income, student, limit |

**TABLE 6.1.** *The first four selected models for best subset selection and forward stepwise selection on the* Credit *data set. The first three models are identical but the fourth models differ.*

# Multiple Linear Regression

## We can use the training set to assess fit using $C_p$, AIC, BIC, or adjusted $R^2$

$$C_p = \frac{1}{n}\left(\text{RSS} + 2d\hat{\sigma}^2\right)$$

where $d$ = number of predictors and $\hat{\sigma}^2$ is the variance of the error estimate

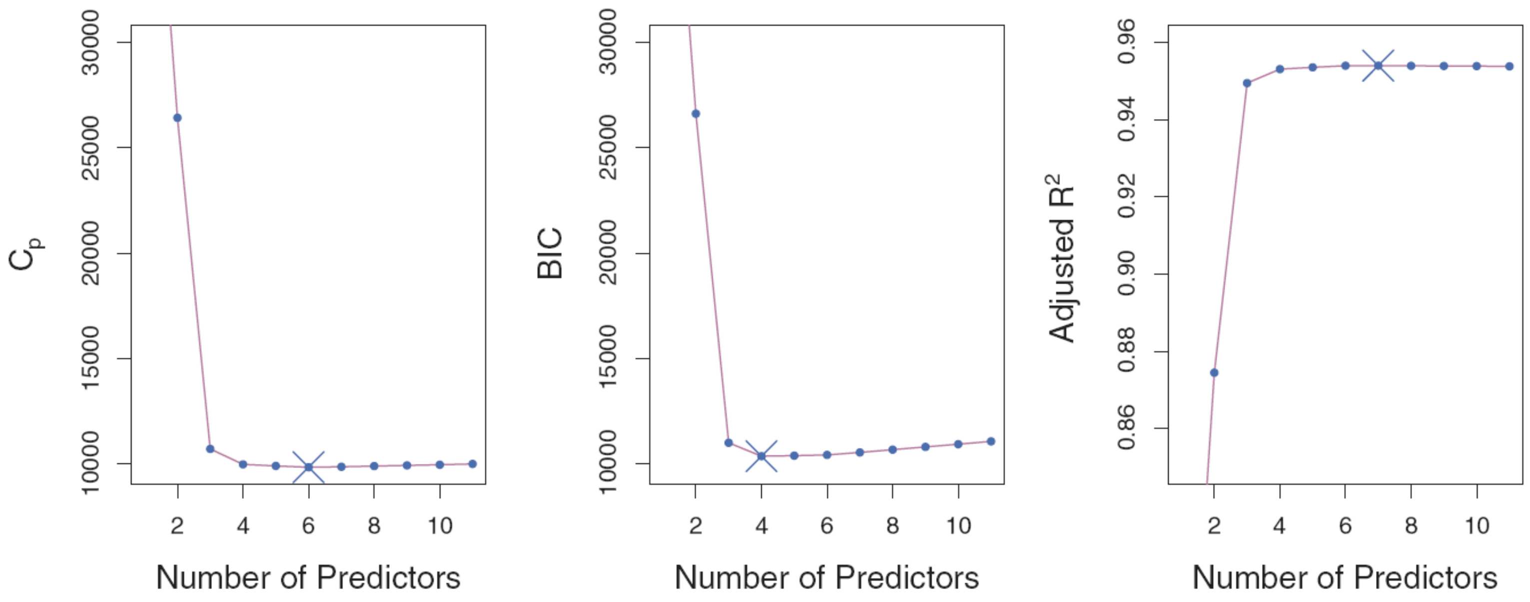



**FIGURE 6.2.** *$C_p$, BIC, and adjusted $R^2$ are shown for the best models of each size for the* Credit *data set (the lower frontier in Figure 6.1). $C_p$ and BIC are estimates of test MSE. In the middle plot we see that the BIC estimate of test error shows an increase after four variables are selected. The other two plots are rather flat after four variables are included.*

# Multiple Linear Regression

**We must use a test set or cross-validation to estimate MSE/RSS**

(not training $C_p$, AIC, BIC, or adjusted $R^2$)

**We can use training-test samples of 0.75, 0.25 or 10-fold cross-validation**

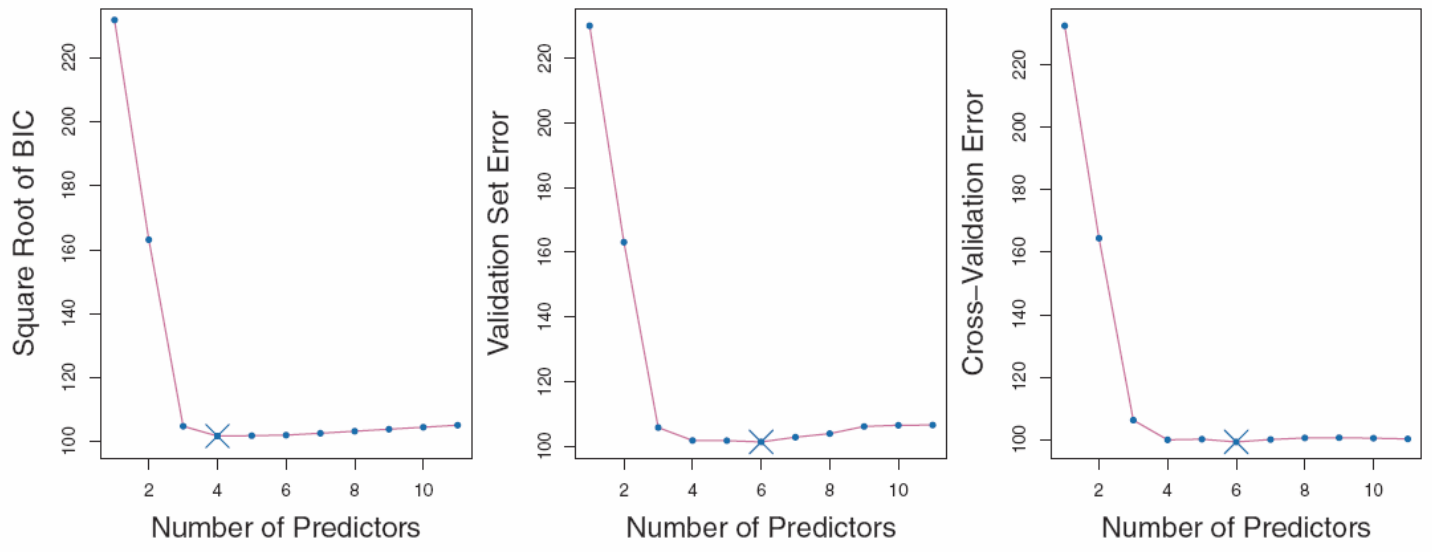



**FIGURE 6.3.** *For the* Credit *data set, three quantities are displayed for the best model containing d predictors, for d ranging from* 1 *to* 11. *The overall* best *model, based on each of these quantities, is shown as a blue cross.* Left: *Square root of BIC.* Center: *Validation set errors.* Right: *Cross-validation errors.*

# Multiple Linear Regression


**Models with 3 to 11 predictors look pretty flat!**

- if we ran again, best number of predictors could change

**How to choose?**

Calculate the standard error of the MSE for each number of predictors

Choose the smallest number of predictors within 1 s.e. of smallest value

(Occams' razor - simpler models are better!)

# Other ways of fitting a linear model

**Shrinkage**

- fit a model using all predictors, but *shrink* (regularize) all coefficients toward 0.
- if exactly zero, they have been removed
- reduces the variance (regularized) and adds small bias
- tradeoff is worth it!

**Be sure to standardize the predictors!**

# Shrinkage

The OLS model minimizes:

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

The shrinkage model minimizes

$$\text{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

The tuning parameter = lambda > 0

shrinkage penalty
(sum of squared coefficients)

# Shrinkage

## Coefficients with shrinkage

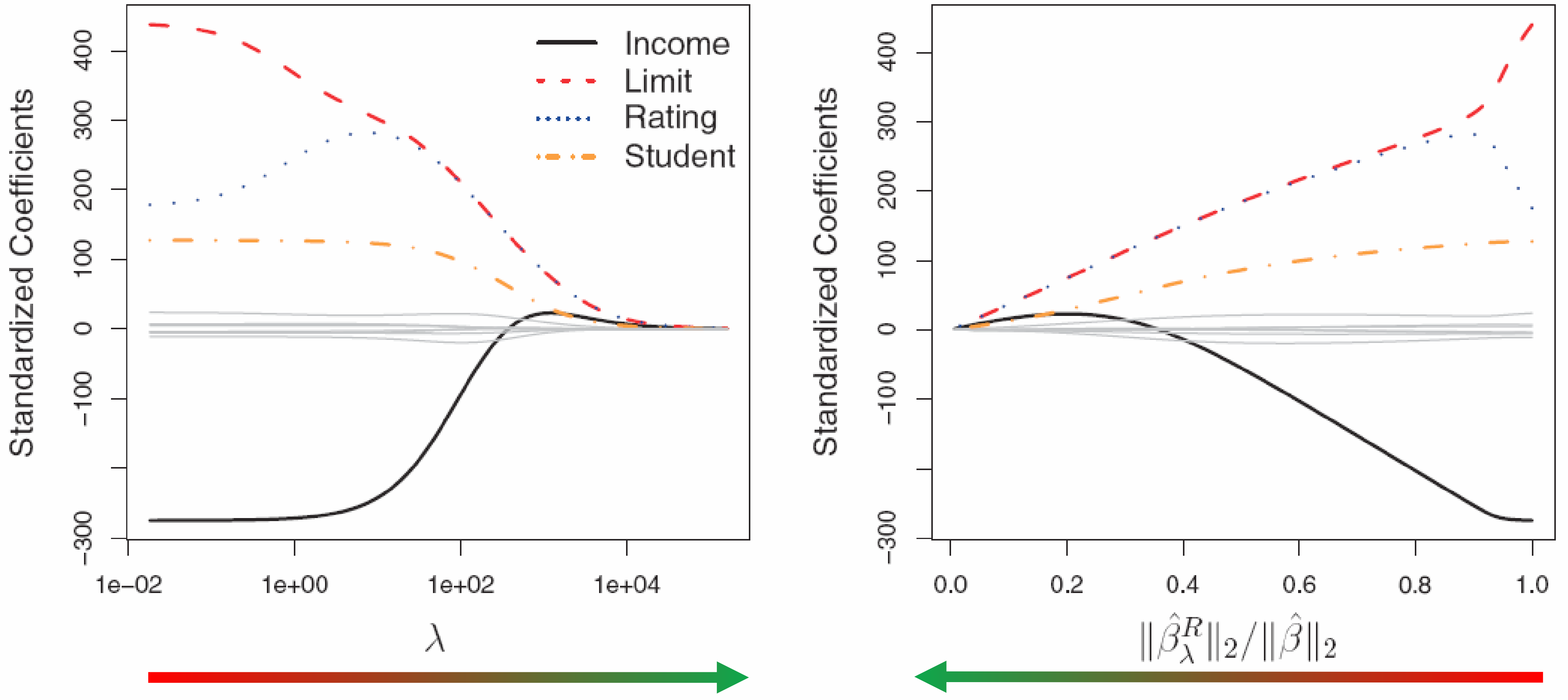



**FIGURE 6.4.** *The standardized ridge regression coefficients are displayed for the* Credit *data set, as a function of* $\lambda$ *and* $\|\hat{\beta}_\lambda^R\|_2/\|\hat{\beta}\|_2$.

# Shrinkage

## The sweet spot in the bias-variance tradeoff

45 predictors, n = 50

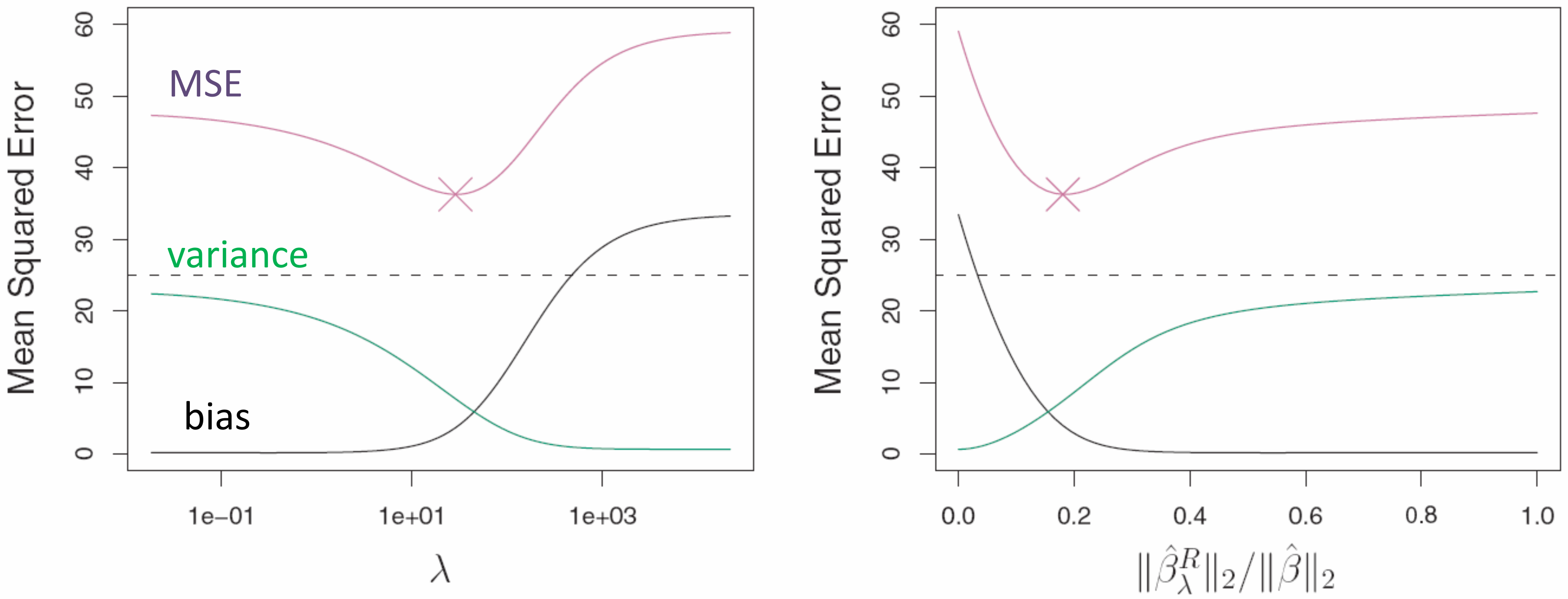



**FIGURE 6.5.** *Squared bias (black), variance (green), and test mean squared error (purple) for the ridge regression predictions on a simulated data set, as a function of* $\lambda$ *and* $\|\hat{\beta}_\lambda^R\|_2/\|\hat{\beta}\|_2$. *The horizontal dashed lines indicate the minimum possible MSE. The purple crosses indicate the ridge regression models for which the MSE is smallest.*

# The Lasso

Shrinkage uses all predictors, so understanding the model can be difficult

- some are not completely removed

The Lasso actually removes some predictors

- so it performs variable selection

The Lasso produces sparse models - with fewer predictors

- uses a tuning parameter $\lambda$, and we can use cross-validation to choose $\lambda$
- can also use a budget (constraint, *s*) on how many $\beta_{\lambda}^{R} > 0$ (Lasso coefficients)
- so we can perform selection on many more predictors

# The Lasso

## The shrinkage model minimizes

$$\mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

The tuning parameter = lambda > 0

shrinkage penalty
(sum of squared coefficients)

## The Lasso model minimizes

$$\mathrm{RSS} + \lambda \sum_{j=1}^{p} |\beta_j|$$

The tuning parameter = lambda > 0

**With high $\lambda$, some coefficients will be zero**

shrinkage penalty
(sum of absolute values of coefficients)

# The Lasso

## Coefficients with the lasso: some = 0 before others

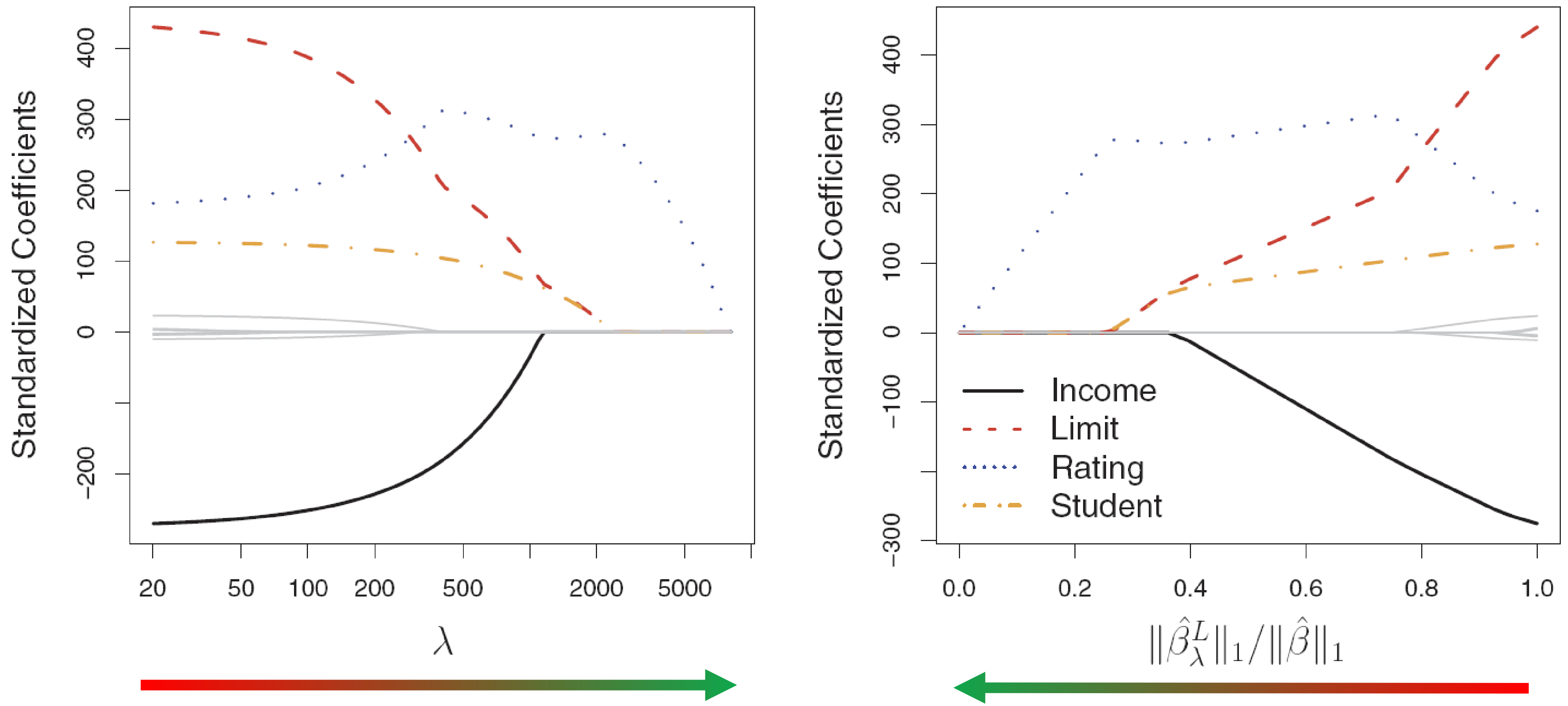



**FIGURE 6.6.** *The standardized lasso coefficients on the* `Credit` *data set are shown as a function of* $\lambda$ *and* $\|\hat{\beta}_\lambda^L\|_1/\|\hat{\beta}\|_1$.

# The Lasso

Minimum MSE areas include "corners"

- at corners, some coefficient = 0

- if $\lambda$ = 0, includes OLS model

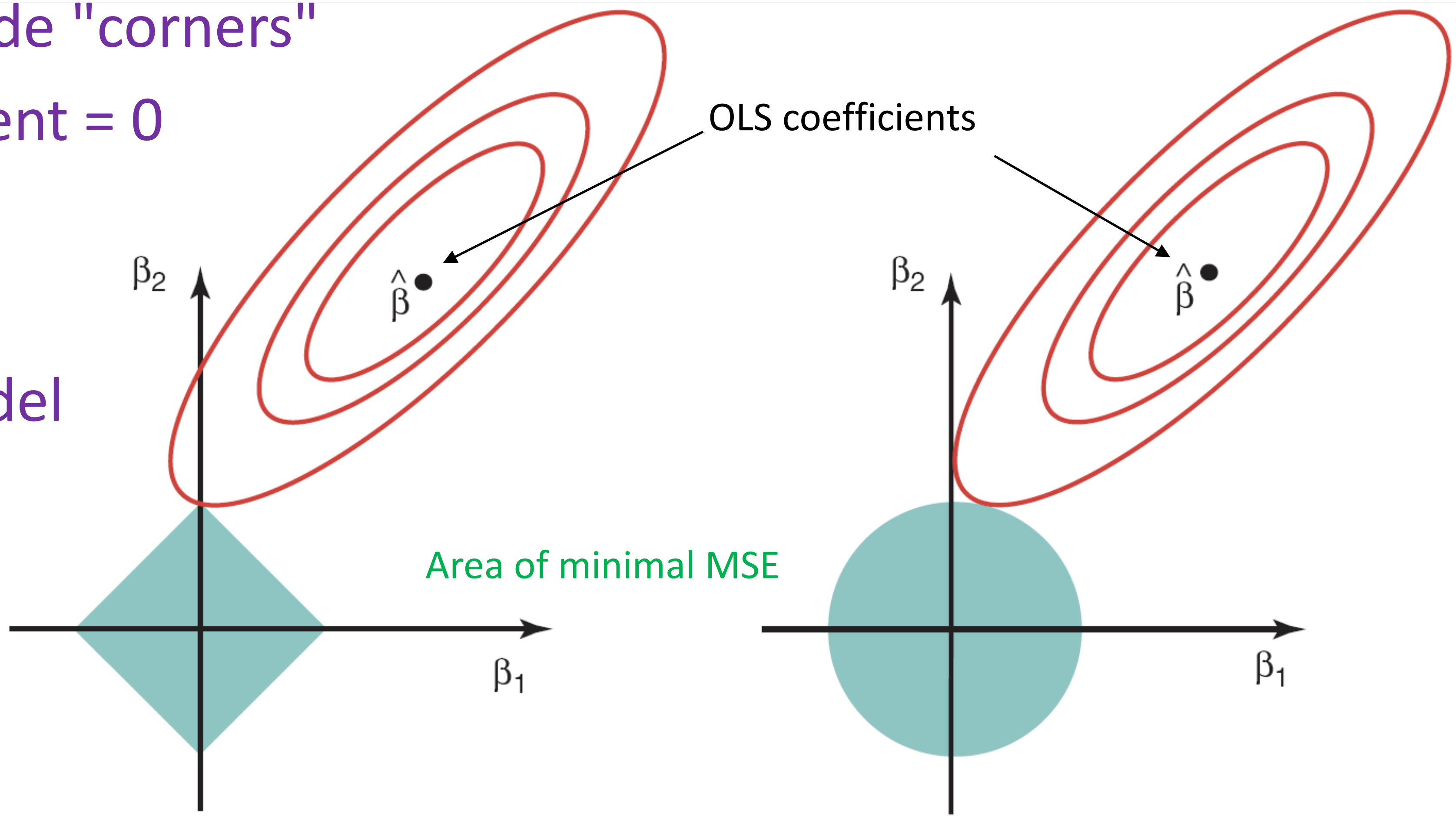



**FIGURE 6.7.** *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions,* $|\beta_1| + |\beta_2| \leq s$ *and* $\beta_1^2 + \beta_2^2 \leq s$, *while the red ellipses are the contours of the RSS.*

# The Lasso

## The "sweet spot" using the lasso

Uses only 2 of 45 predictors **correlated with response**, n = 50

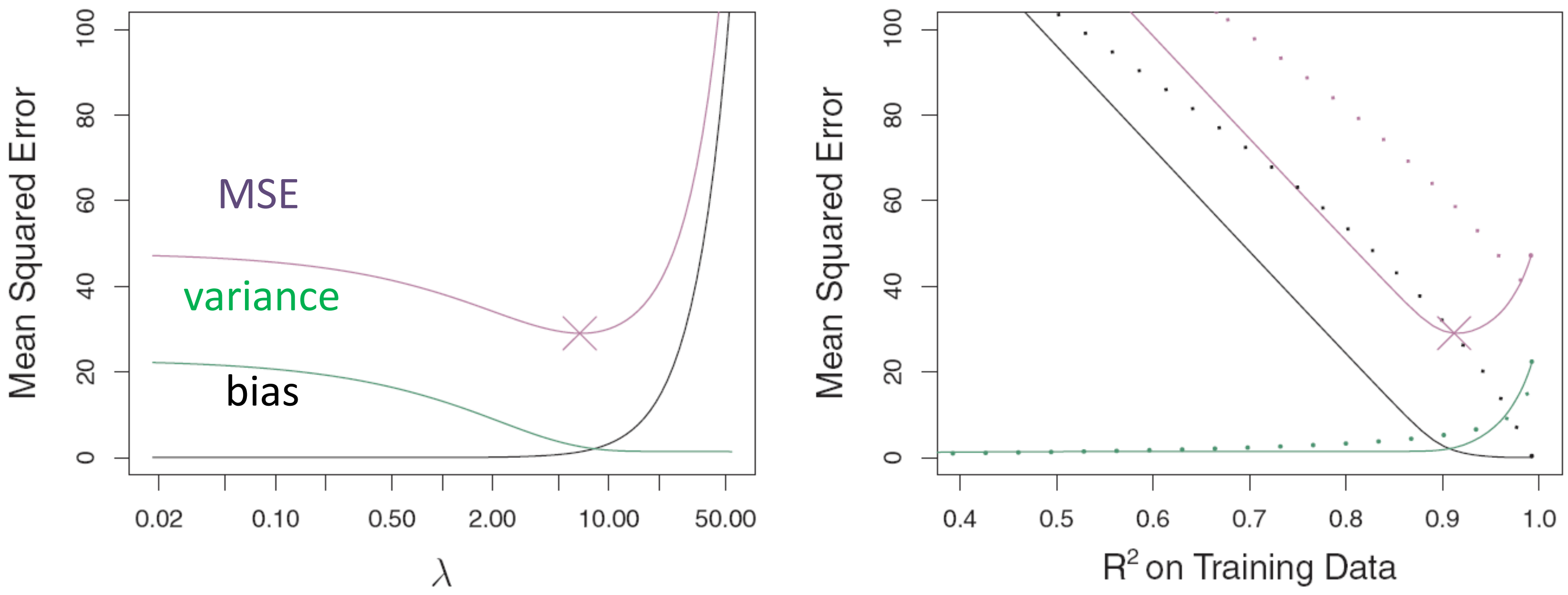



**FIGURE 6.9.** Left: *Plots of squared bias (black), variance (green), and test MSE (purple) for the lasso. The simulated data is similar to that in Figure 6.8, except that now only two predictors are related to the response.* Right: *Comparison of squared bias, variance and test MSE between lasso (solid) and ridge (dotted). Both are plotted against their $R^2$ on the training data, as a common form of indexing. The crosses in both plots indicate the lasso model for which the MSE is smallest.*

# Tuning

## The "sweet spot" for λ using ridge regression

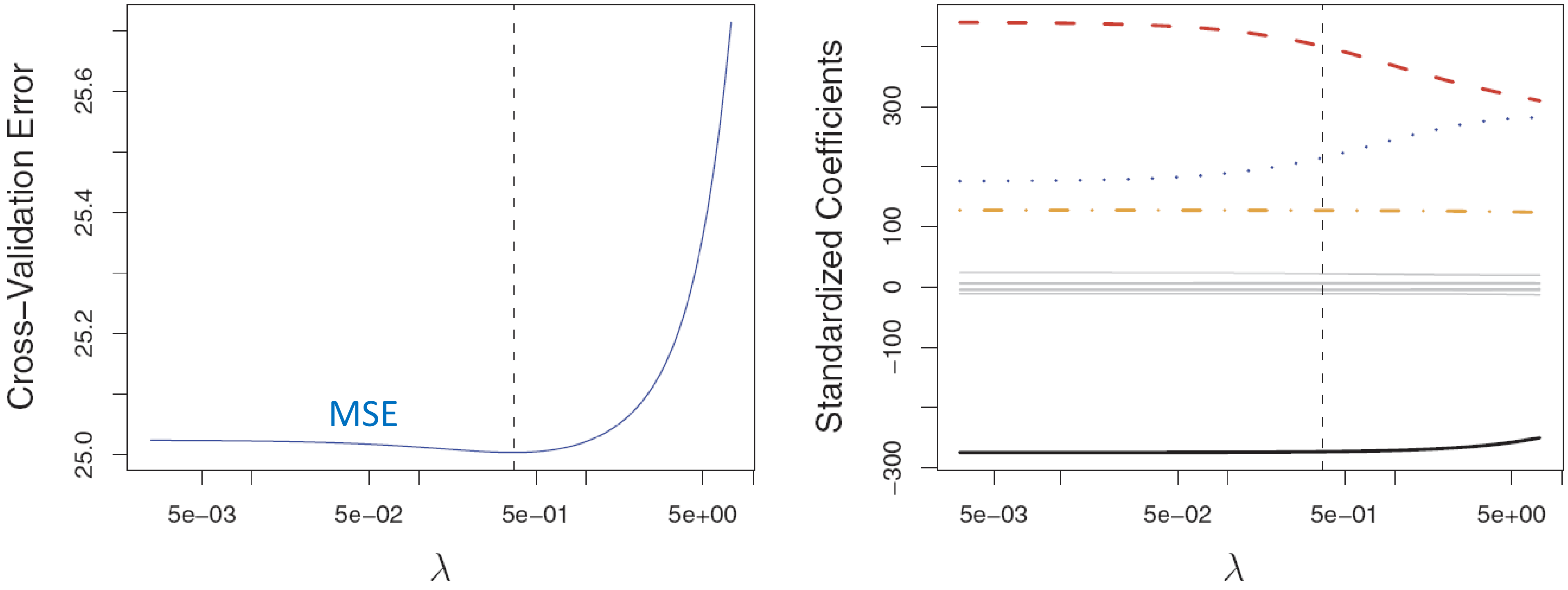



**FIGURE 6.12.** Left: *Cross-validation errors that result from applying ridge regression to the* `Credit` *data set with various value of* $\lambda$. Right: *The coefficient estimates as a function of* $\lambda$. *The vertical dashed lines indicate the value of* $\lambda$ *selected by cross-validation.*

# Tuning

## The "sweet spot" for λ using the lasso

Uses only 2 of 45 predictors **correlated with response**, n = 50

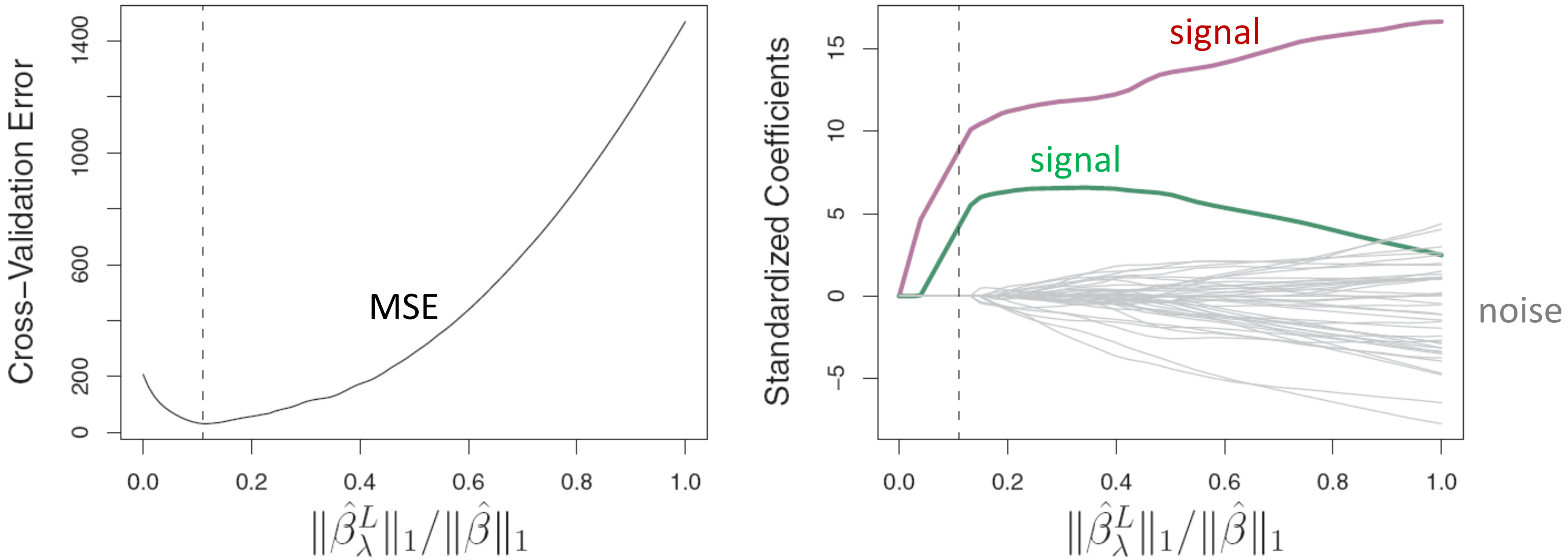



**FIGURE 6.13.** Left: *Ten-fold cross-validation MSE for the lasso, applied to the sparse simulated data set from Figure 6.9.* Right: *The corresponding lasso coefficient estimates are displayed. The vertical dashed lines indicate the lasso fit for which the cross-validation error is smallest.*

# Ridge Regression and the Lasso

**Ridge Regression** will perform better when the outcome is a function of many predictors with roughly equal coefficients

**The Lasso** will perform better when the outcome is a function of a small number of predictors and the rest are very small or zero

We do not know these things in advance!

So we use cross-validation to see which is best for the data at hand

Use the lasso for interpretations using simpler models

We will use cross-validation to select values for $\lambda$ and $s$

# R, Lasso, Ridge Regression

```
# install.packages("glmnet")
library(glmnet) # lasso and ridge regression

#install.packages("ISLR")
library(ISLR)
#library(caret) #tune hyper-parameters
library(leaps) # subset selection

Hitters <- na.omit(Hitters)
```

# Subset Selection

```
str(Hitters) # in ISLR
'data.frame':  322 obs. of  20 variables:
 $ AtBat    : num  293 315 479 496 321 594 185 298 323 401 ...
 $ Hits     : num  66 81 130 141 87 169 37 73 81 92 ...
 $ HmRun    : num  1 7 18 20 10 4 1 0 6 17 ...
 $ Runs     : num  30 24 66 65 39 74 23 24 26 49 ...
 $ RBI      : num  29 38 72 78 42 51 8 24 32 66 ...
 $ Walks    : num  14 39 76 37 30 35 21 7 8 65 ...
 $ Years    : num  1 14 3 11 2 11 2 3 2 13 ...
 $ CAtBat   : num  293 3449 1624 5628 396 ...
 $ CHits    : num  66 835 457 1575 101 ...
 $ CHmRun   : num  1 69 63 225 12 19 1 0 6 253 ...
 $ CRuns    : num  30 321 224 828 48 501 30 41 32 784 ...
 $ CRBI     : num  29 414 266 838 46 336 9 37 34 890 ...
 $ CWalks   : num  14 375 263 354 33 194 24 12 8 866 ...
 $ League   : Factor w/ 2 levels "A","N": 1 2 1 2 2 1 2 1 2 1 ...
 $ Division : Factor w/ 2 levels "E","W": 1 2 2 1 1 2 1 2 2 1 ...
 $ PutOuts  : num  446 632 880 200 805 282 76 121 143 0 ...
 $ Assists  : num  33 43 82 11 40 421 127 283 290 0 ...
 $ Errors   : num  20 10 14 3 4 25 7 9 19 0 ...
 $ Salary   : num  NA 475 480 500 91.5 750 70 100 75 1100 ...
 $ NewLeague: Factor w/ 2 levels "A","N": 1 2 1 2 2 1 1 1 2 1 ...
```

# Subset Selection

**Hitters**

A data frame with 322 observations of major league players on the following 20 variables.

| | |
|---|---|
| AtBat | Number of times at bat in 1986 |
| Hits | Number of hits in 1986 |
| HmRun | Number of home runs in 1986 |
| Runs | Number of runs in 1986 |
| RBI | Number of runs batted in in 1986 |
| Walks | Number of walks in 1986 |
| Years | Number of years in the major leagues |
| CAtBat | Number of times at bat during his career |
| CHits | Number of hits during his career |
| CHmRun | Number of home runs during his career |
| CRuns | Number of runs during his career |
| CRBI | Number of runs batted in during his career |
| CWalks | Number of walks during his career |
| League | A factor with levels A and N indicating player's league at the end of 1986 |
| Division | A factor with levels E and W indicating player's division at the end of 1986 |
| PutOuts | Number of put outs in 1986 |
| Assists | Number of assists in 1986 |
| Errors | Number of errors in 1986 |
| Salary | 1987 annual salary on opening day in thousands of dollars |
| NewLeague | A factor with levels A and N indicating player's league at the beginning of 1987 |

# Subset Selection

```
# We want to predict salary using ALL SUBSETS through regsubsets
reg.ss <- regsubsets(Salary~., Hitters)
summary (reg.ss)
Subset selection object
Call: regsubsets.formula(Salary ~ ., Hitters)
19 Variables  (and intercept)
           Forced in Forced out
AtBat          FALSE      FALSE
Hits           FALSE      FALSE
HmRun          FALSE      FALSE
Runs           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 8
Selection Algorithm: exhaustive
         AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     " "       " "     " "     " "    " "
2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     " "       " "     " "     " "    " "
3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     " "       "*"     " "     " "    " "
4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     "*"       "*"     " "     " "    " "
5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     "*"       "*"     " "     " "    " "
6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"  " "    " "     "*"       "*"     " "     " "    " "
7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "  " "    " "     "*"       "*"     " "     " "    " "
8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "  "*"    " "     "*"       "*"     " "     " "    " "
```

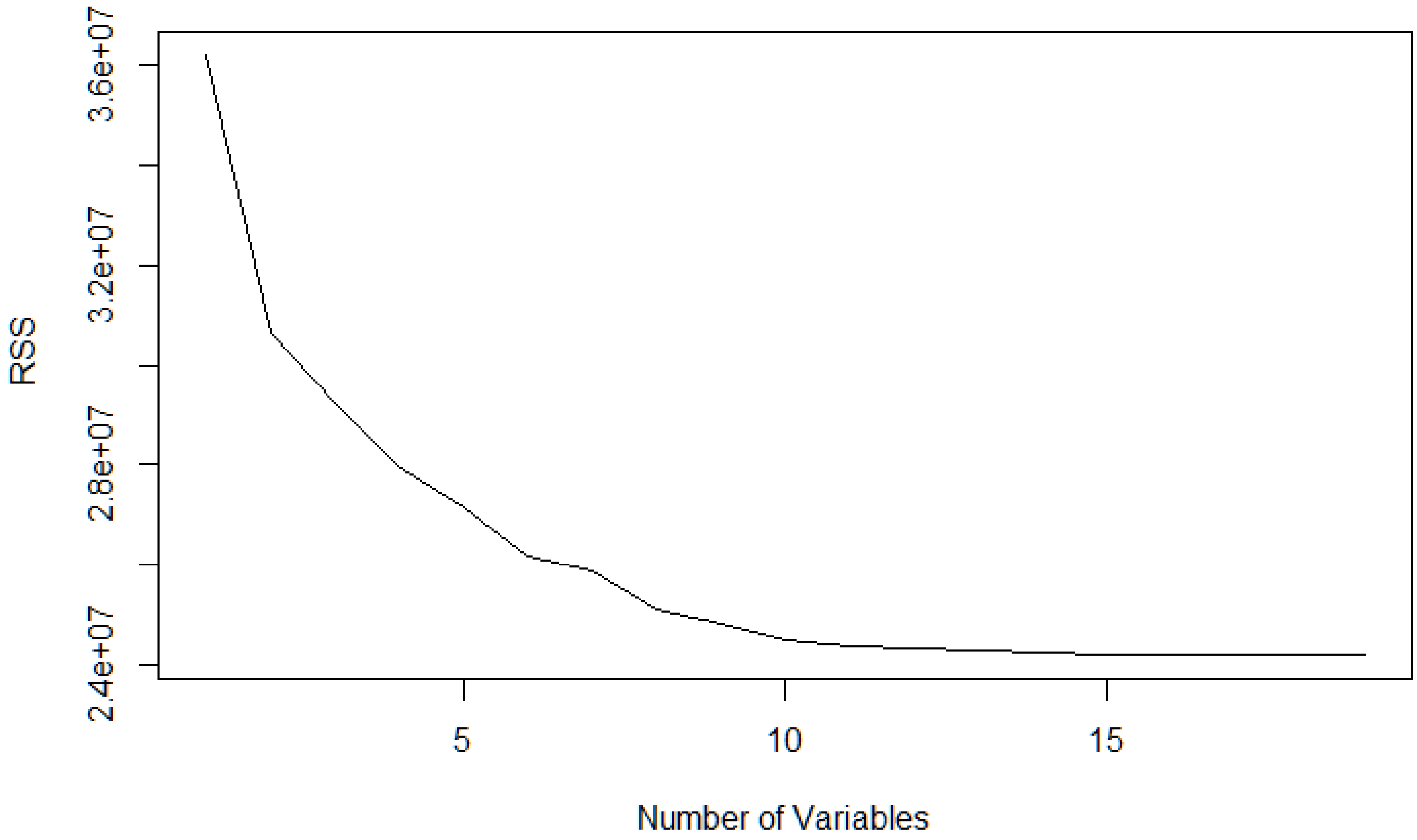
# Subset Selection

```
# We want ALL 19 vars
reg.ss <- regsubsets(Salary~., Hitters, nvmax =19)
res.reg.ss <- summary(reg.ss)
Subset selection object
Call: regsubsets.formula(Salary ~ ., Hitters, nvmax = 19)
19 Variables  (and intercept)
           Forced in Forced out
AtBat          FALSE      FALSE
...
NewLeagueN     FALSE      FALSE
1 subsets of each size up to 19
Selection Algorithm: exhaustive
          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI 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 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

# Subset Selection

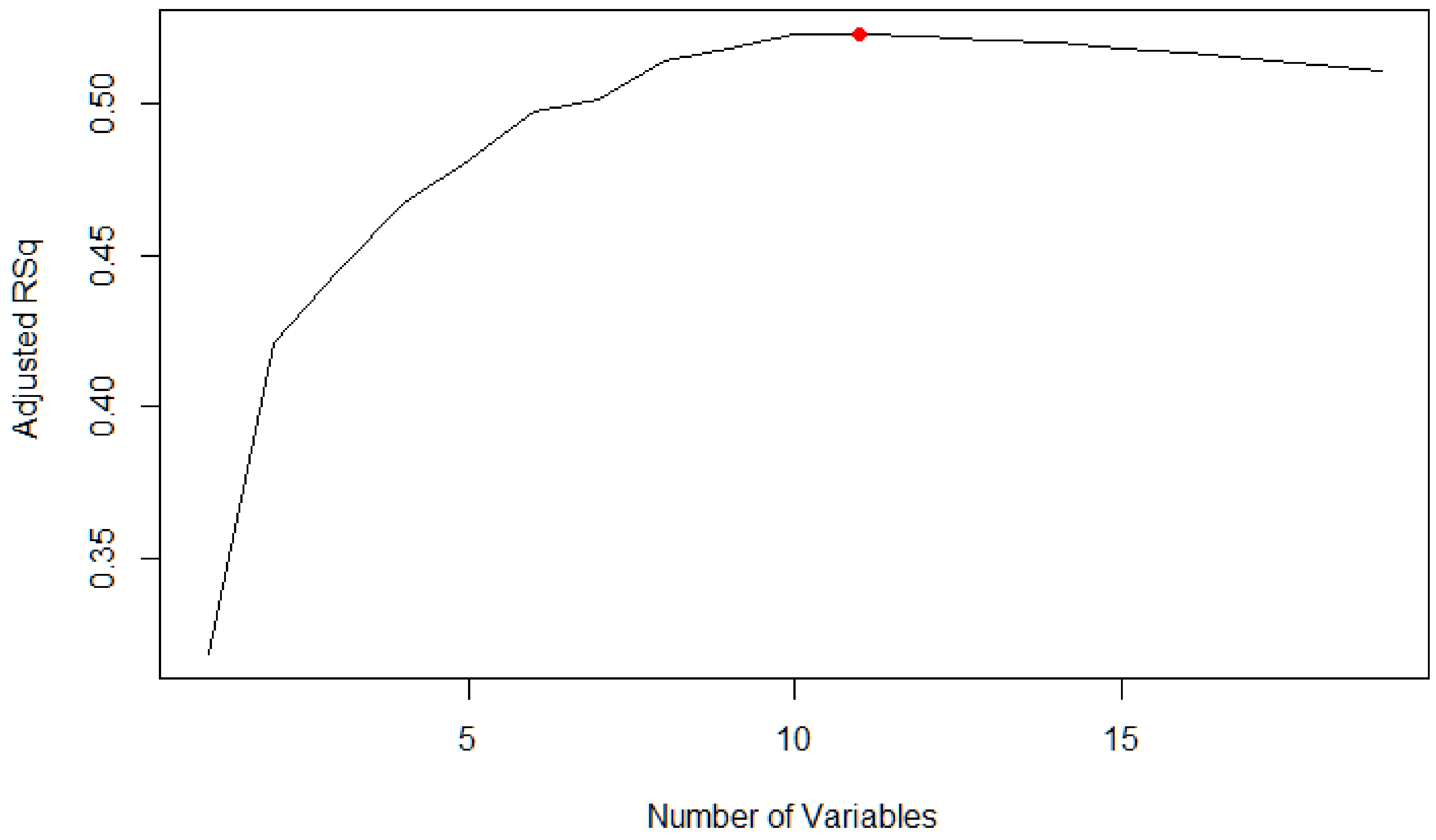
```
# We want information on each model
res.reg.ss$rsq
 [1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227 0.5285569 0.5346124 0.5404950
[11] 0.5426153 0.5436302 0.5444570 0.5452164 0.5454692 0.5457656 0.5459518 0.5460945 0.5461159

# let's plot some results
#par(mfrow =c(2,2))
plot(res.reg.ss$rss ,xlab=" Number of Variables ",ylab=" RSS", type = 'l')
plot(res.reg.ss$adjr2 ,xlab =" Number of Variables ", ylab=" Adjusted RSq",type = 'l')

bestset <- which.max(res.reg.ss$adjr2)
bestset
points (bestset, res.reg.ss$adjr2[bestset], col = "red",cex = 1.5, pch = 20)


plot(res.reg.ss$cp ,xlab =" Number of Variables ", ylab = "Cp", type = 'l')
cpmin <- which.min (res.reg.ss$cp )
cpmin
points (cpmin, res.reg.ss$cp [cpmin], col ="red",cex = 1.5, pch = 20)
bicmin <- which.min(res.reg.ss$bic)
bicmin

plot(res.reg.ss$bic ,xlab=" Number of Variables ", ylab = " BIC",type = 'l')
points (bicmin, res.reg.ss$bic [6], col =" red", cex = 2, pch = 20)

par(mfrow =c(1,1))
```

# Baseball Salaries



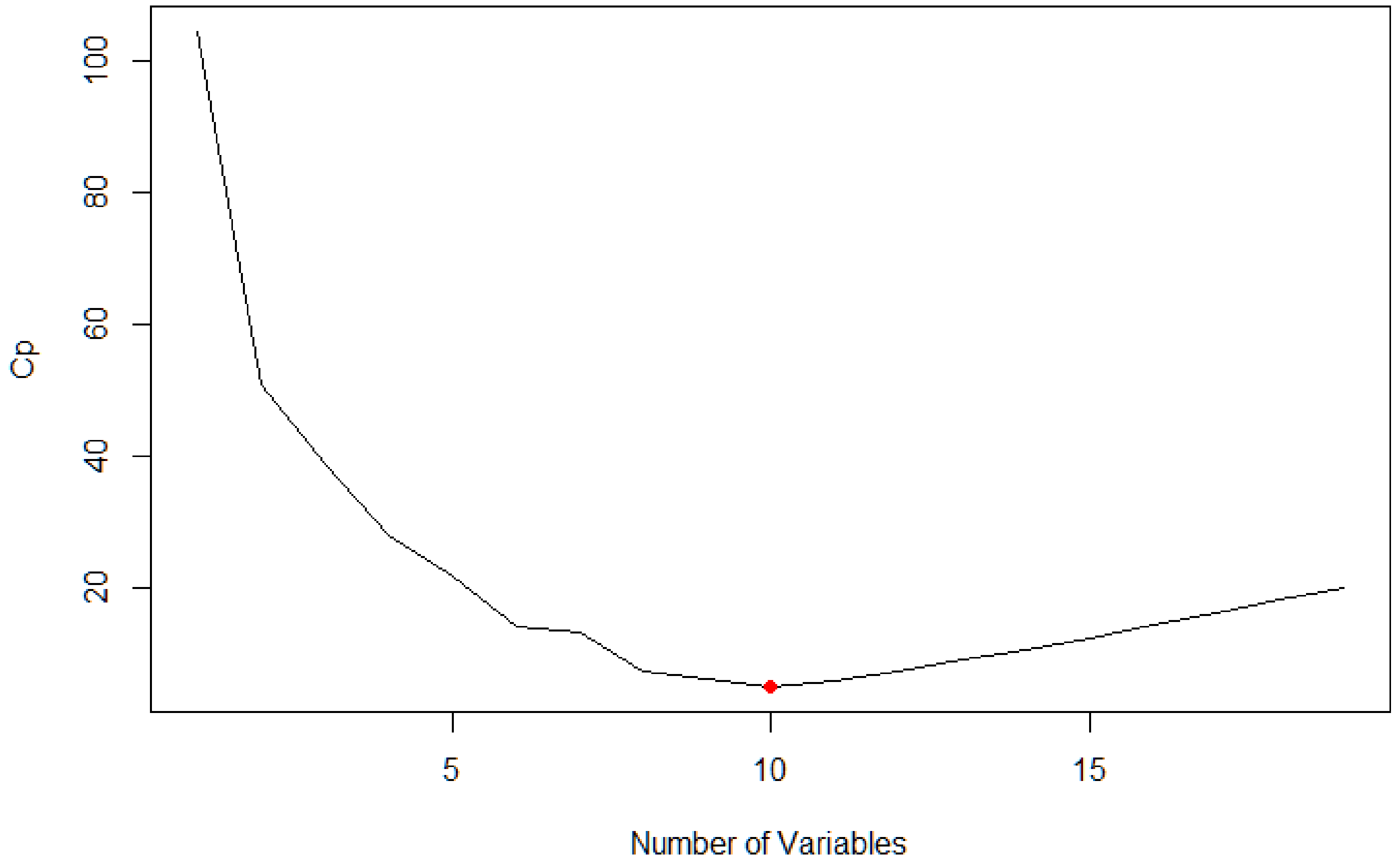

```
plot(res.reg.ss$rss ,xlab=" Number of Variables ",ylab=" RSS", type = 'l')
```

# Baseball Salaries



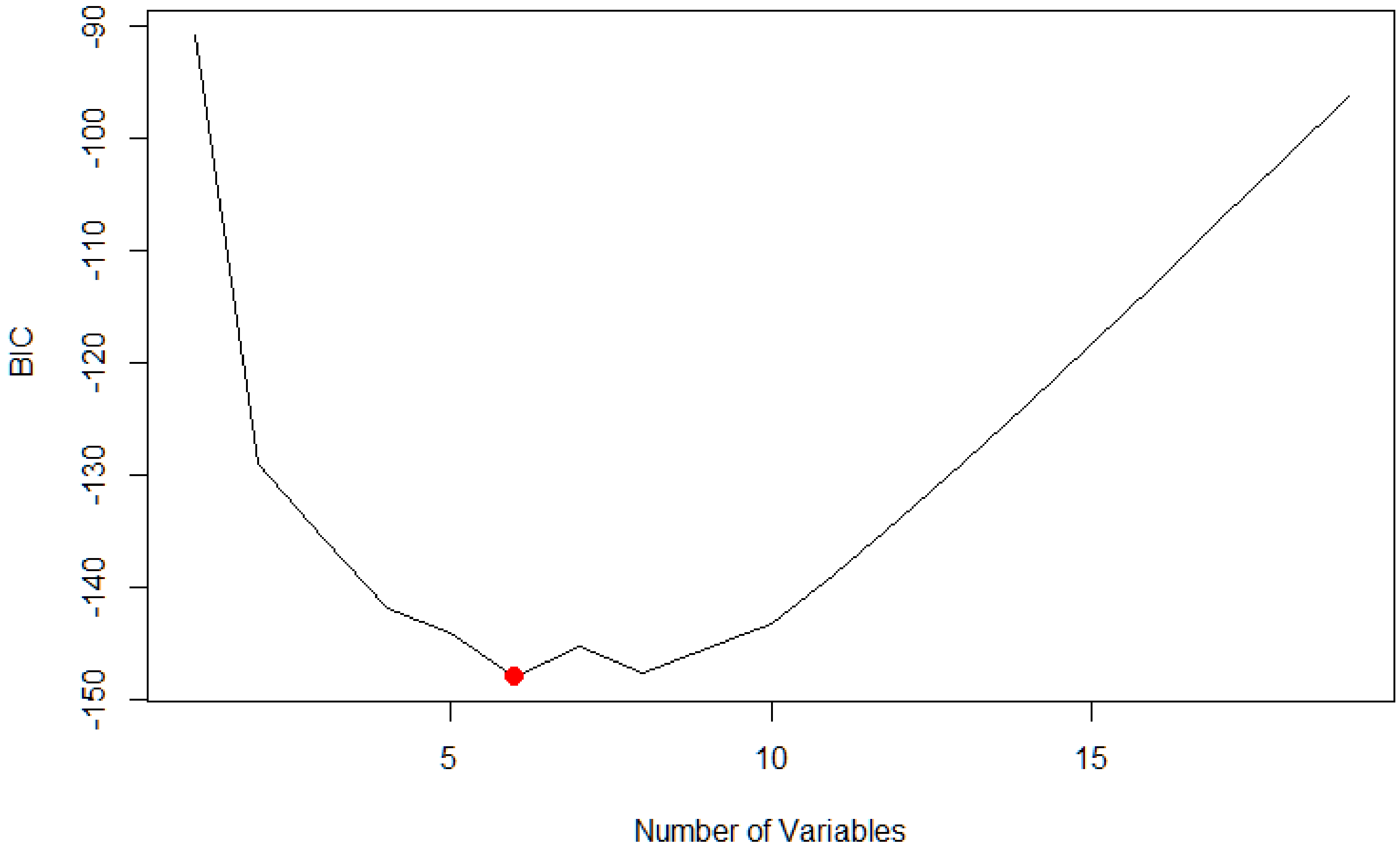

```
plot(res.reg.ss$adjr2 ,xlab =" Number of Variables ", ylab=" Adjusted RSq",type = 'l')
bestset <- which.max(res.reg.ss$adjr2)
points (bestset, res.reg.ss$adjr2[bestset], col = "red",cex = 1.5, pch = 20)
```

# Baseball Salaries



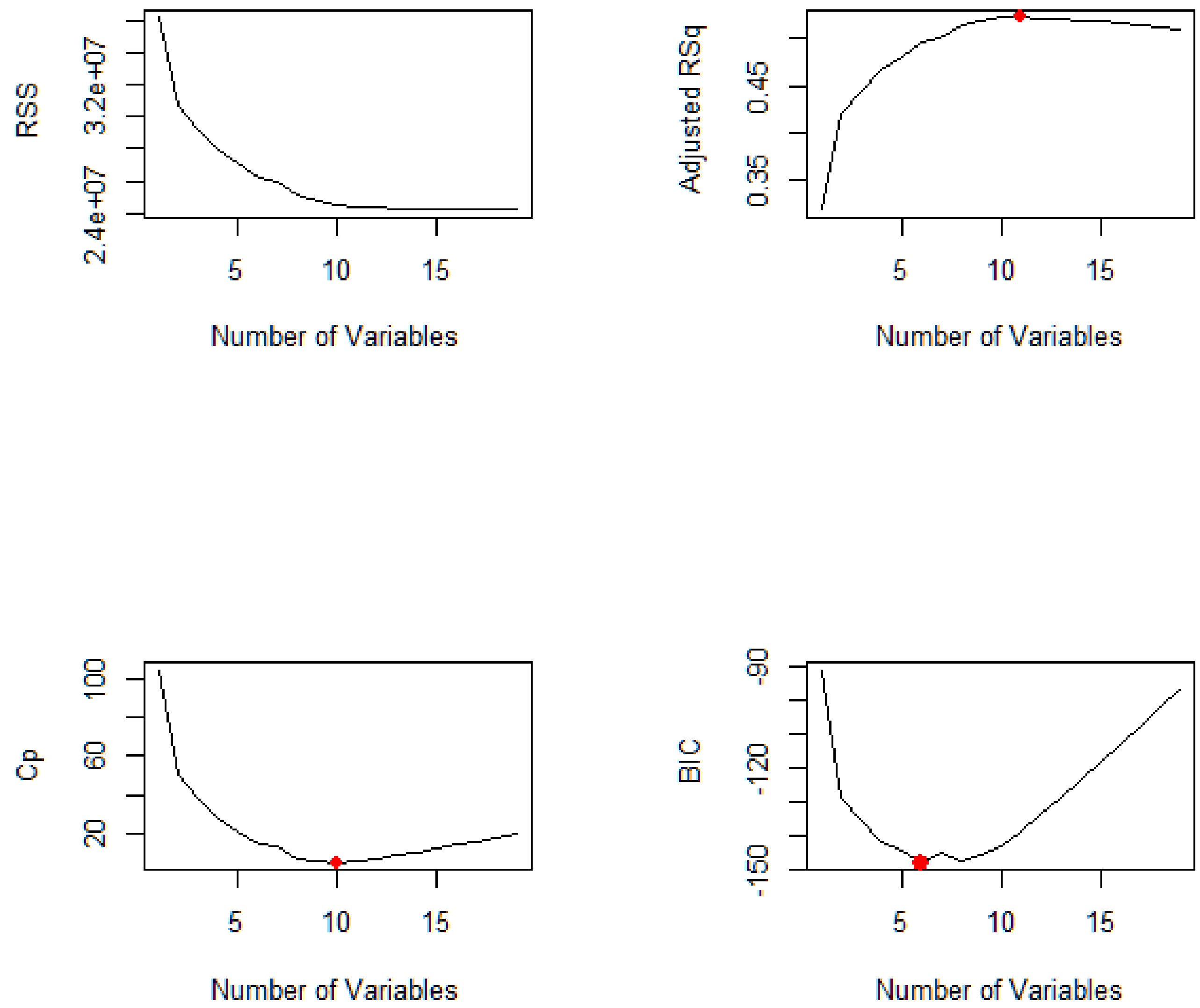

```
plot(res.reg.ss$cp ,xlab =" Number of Variables ", ylab = "Cp", type = 'l')
cpmin <- which.min (res.reg.ss$cp )
points (cpmin, res.reg.ss$cp [cpmin], col ="red",cex = 1.5, pch = 20)
```

# Baseball Salaries




```
bicmin <- which.min(res.reg.ss$bic) # bicmin = 6
plot(res.reg.ss$bic ,xlab=" Number of Variables ", ylab = " BIC",type = 'l')
points (bicmin, res.reg.ss$bic [6], col =" red", cex = 2, pch = 20)
```

# Baseball Salaries




```
par(mfrow =c(2,2))
```

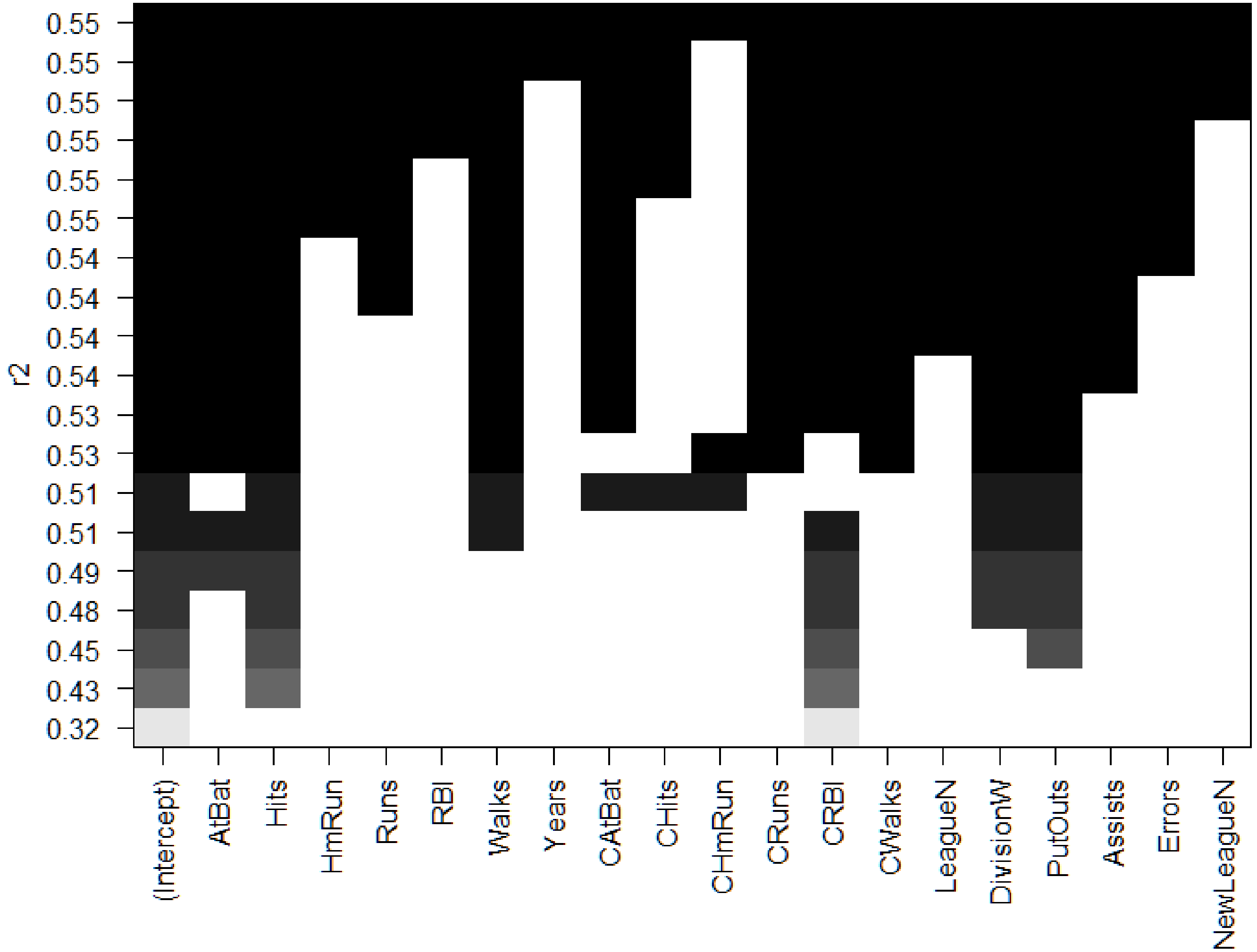
# Baseball Salaries

## We can also do other diagnostic graphs related to variable importance

```
plot(reg.ss,scale ="r2")
plot(reg.ss,scale ="adjr2")
plot(reg.ss,scale ="Cp")
plot(reg.ss,scale ="bic")
```

```
bicmin
[1] 6
```

# Baseball Salaries



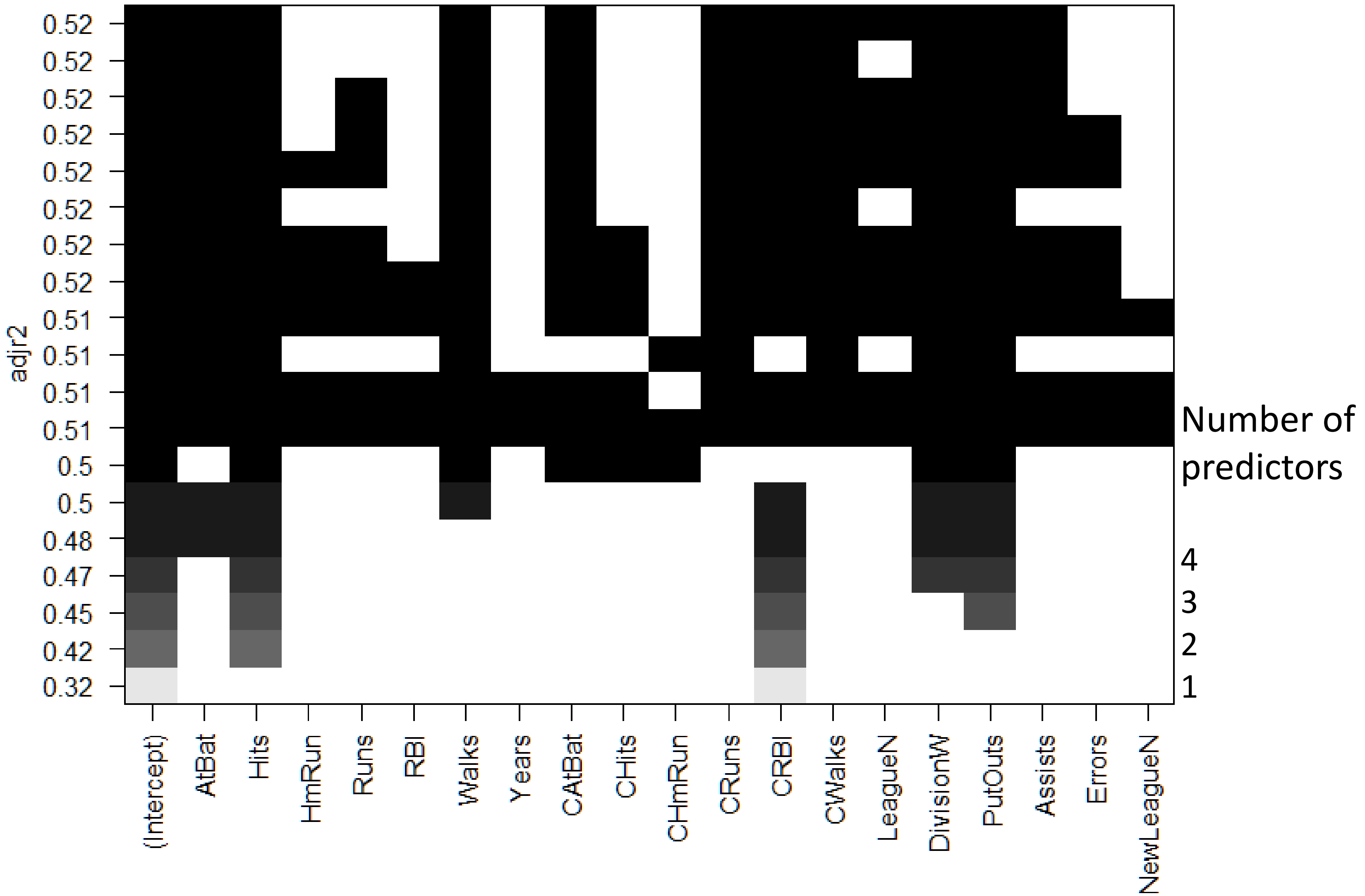

Dark blocks: Clearly better combinations

Number of predictors

4
3
2
1

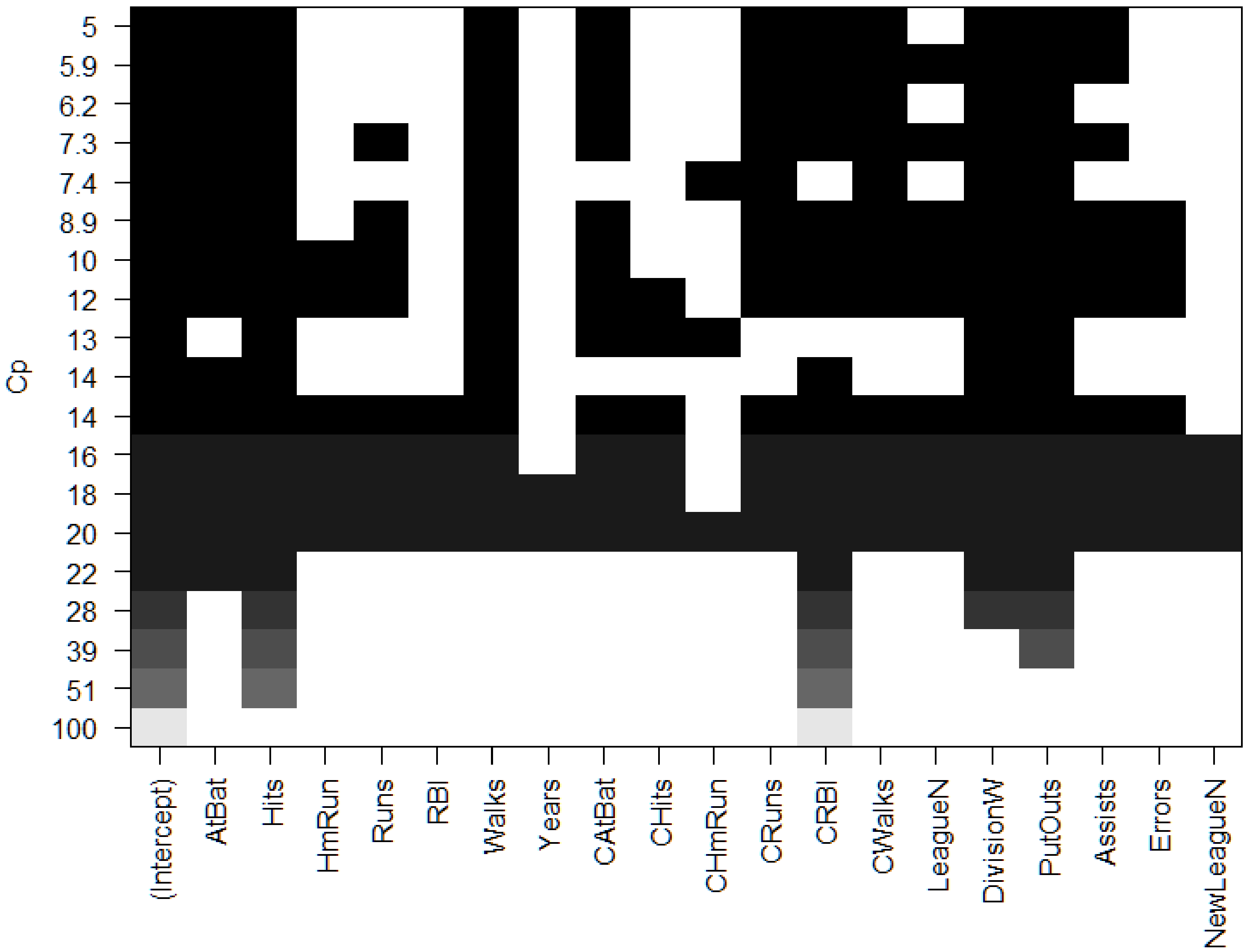
`plot(reg.ss,scale ="r2")`

# Baseball Salaries




`plot(reg.ss,scale ="adjr2")`

# Baseball Salaries

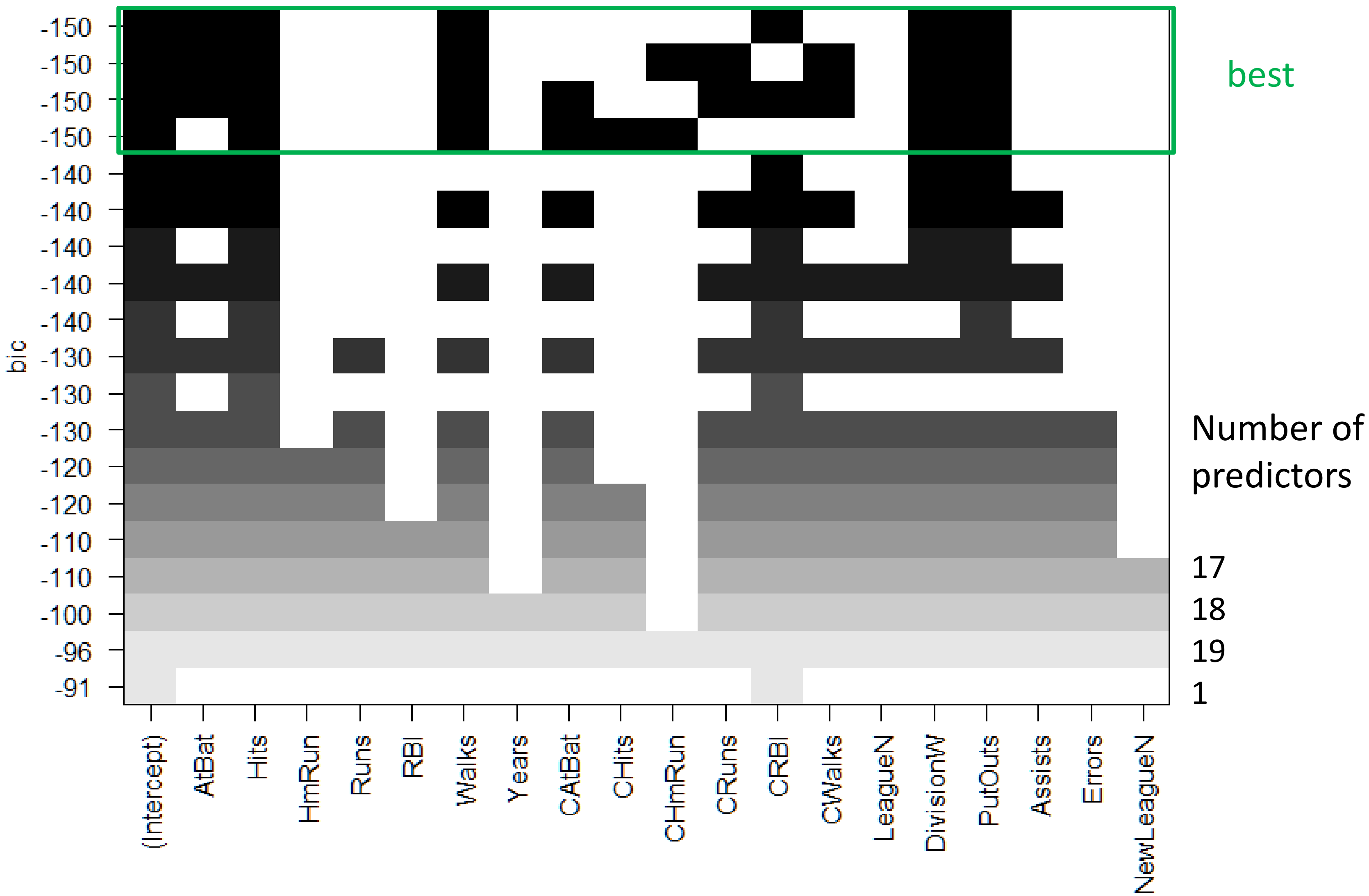



Number of predictors

13
14
5
3
2
1

`plot(reg.ss,scale ="Cp")`

# Baseball Salaries




`plot(reg.ss,scale ="bic")`

# Baseball Salaries

## Look at the coefficients for the best-fit model using BIC

```
bicmin
[1] 6
coef(reg.ss, bicmin)
 (Intercept)        AtBat         Hits        Walks         CRBI    DivisionW      PutOuts
  91.5117981   -1.8685892    7.6043976    3.6976468    0.6430169 -122.9515338    0.2643076
```

**These are the OLS coefficients and can be used directly to calculate salary**

## We can also use forward stepwise selection of predictors using `regsubsets` and `forward`

```
reg.fwd <- regsubsets (Salary~., data= Hitters ,nvmax = 19, method = "forward")
```

# Forward Selection

```
summary(reg.fwd)
Subset selection object
Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "forward")
19 Variables  (and intercept)
           Forced in Forced out
AtBat          FALSE      FALSE
Hits           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 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 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

```
which.min(summary(reg.fwd)$bic)
[1] 6
which.min(summary(reg.fwd)$adjr2)
[1] 1
which.min(summary(reg.fwd)$cp)
[1] 10
```

# Baseball Salaries

## Let's compare best-subsets and forward-stepwise models

```
# best subsets using bicmin (6)
coef(reg.ss, 6)
 (Intercept)       AtBat        Hits       Walks        CRBI    DivisionW      PutOuts
  91.5117981  -1.8685892   7.6043976   3.6976468   0.6430169 -122.9515338    0.2643076

# best 6 using forward stepwise
coef(reg.fwd, 6)
 (Intercept)       AtBat        Hits       Walks        CRBI    DivisionW      PutOuts
  91.5117981  -1.8685892   7.6043976   3.6976468   0.6430169 -122.9515338    0.2643076

# best subsets with 7
coef(reg.ss, 7)
 (Intercept)        Hits       Walks      CAtBat       CHits      CHmRun    DivisionW      PutOuts
  79.4509472   1.2833513   3.2274264  -0.3752350   1.4957073   1.4420538 -129.9866432    0.2366813

# best forward with 7
coef(reg.fwd, 7)
 (Intercept)       AtBat        Hits       Walks        CRBI      CWalks    DivisionW      PutOuts
 109.7873062  -1.9588851   7.4498772   4.9131401   0.8537622  -0.3053070 -127.1223928    0.2533404
```

# Choosing the best prediction models

## Model-fitting (number of predictors) will be done using training set

## Test error will be estimated using test set

```
set.seed (7)
# not "stratified" but does not make sense here, simply random 50/50, T/F for every record
train <- sample(c(TRUE ,FALSE), nrow(Hitters),rep = TRUE)
test <- (!train)

# use best subset selection on training set
regfit.ss = regsubsets(Salary~., data = Hitters[train,], nvmax =19)

# set up "x" model matrix
test.mat <- model.matrix(Salary~., data = Hitters[test,])
```

# Choosing the best prediction models

```
# The "x" model matrix
test.mat
```

| | (Intercept) | AtBat | Hits | HmRun | Runs | RBI | Walks | Years | CAtBat | CHits | CHmRun | CRuns | CRBI | CWalks | LeagueN | DivisionW | PutOuts | Assists | Errors | NewLeagueN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -Alan Ashby | 1 | 315 | 81 | 7 | 24 | 38 | 39 | 14 | 3449 | 835 | 69 | 321 | 414 | 375 | 1 | 1 | 632 | 43 | 10 | 1 |
| -Alvin Davis | 1 | 479 | 130 | 18 | 66 | 72 | 76 | 3 | 1624 | 457 | 63 | 224 | 266 | 263 | 0 | 1 | 880 | 82 | 14 | 0 |
| -Andre Dawson | 1 | 496 | 141 | 20 | 65 | 78 | 37 | 11 | 5628 | 1575 | 225 | 828 | 838 | 354 | 1 | 0 | 200 | 11 | 3 | 1 |
| -Andres Galarraga | 1 | 321 | 87 | 10 | 39 | 42 | 30 | 2 | 396 | 101 | 12 | 48 | 46 | 33 | 1 | 0 | 805 | 40 | 4 | 1 |
| -Alfredo Griffin | 1 | 594 | 169 | 4 | 74 | 51 | 35 | 11 | 4408 | 1133 | 19 | 501 | 336 | 194 | 0 | 1 | 282 | 421 | 25 | 0 |
| -Al Newman | 1 | 185 | 37 | 1 | 23 | 8 | 21 | 2 | 214 | 42 | 1 | 30 | 9 | 24 | 1 | 0 | 76 | 127 | 7 | 0 |
| -Argenis Salazar | 1 | 298 | 73 | 0 | 24 | 24 | 7 | 3 | 509 | 108 | 0 | 41 | 37 | 12 | 0 | 1 | 121 | 283 | 9 | 0 |
| -Andres Thomas | 1 | 323 | 81 | 6 | 26 | 32 | 8 | 2 | 341 | 86 | 6 | 32 | 34 | 8 | 1 | 1 | 143 | 290 | 19 | 1 |
| -Andre Thornton | 1 | 401 | 92 | 17 | 49 | 66 | 65 | 13 | 5206 | 1332 | 253 | 784 | 890 | 866 | 0 | 0 | 0 | 0 | 0 | 0 |
| -Buddy Bell | 1 | 568 | 158 | 20 | 89 | 75 | 73 | 15 | 8068 | 2273 | 177 | 1045 | 993 | 732 | 1 | 1 | 105 | 290 | 10 | 1 |
| -Barry Bonds | 1 | 413 | 92 | 16 | 72 | 48 | 65 | 1 | 413 | 92 | 16 | 72 | 48 | 65 | 1 | 0 | 280 | 9 | 5 | 1 |
| -Bobby Bonilla | 1 | 426 | 109 | 3 | 55 | 43 | 62 | 1 | 426 | 109 | 3 | 55 | 43 | 62 | 0 | 1 | 361 | 22 | 2 | 1 |
| -Brett Butler | 1 | 587 | 163 | 4 | 92 | 51 | 70 | 6 | 2695 | 747 | 17 | 442 | 198 | 317 | 0 | 0 | 434 | 9 | 3 | 0 |
| -Bob Dernier | 1 | 324 | 73 | 4 | 32 | 18 | 22 | 7 | 1931 | 491 | 13 | 291 | 108 | 180 | 1 | 0 | 222 | 3 | 3 | 1 |

```
^^^ notice that the column values are all 0,1 for factors
-  and the column names are different - League --> LeagueN, reflecting factor levels.
- Alan Ashby is in the National League, (League = 'N', not 'A', so LeagueN = 1)
```

# Baseball Salaries

```
# get MSE using test data
val.errors =rep(NA ,19)
for(i in 1:19)
{
  coefi <- coef(regfit.ss, id=i)
  pred <- test.mat[,names(coefi)]%*% coefi  # matrix multiplication
  val.errors[i] <- mean(( Hitters$Salary[test]-pred)^2)
}
val.errors
 [1] 136066.8 143503.4 134202.7 106604.0 111873.1 113970.1 105141.6 109710.0 109247.8 113079.4
[11] 114646.0 113463.2 116690.8 120297.5 119492.7 118188.6 113200.9 112782.0 112333.4

which.min(val.errors) # different from book; different from you?
[1] 7

coef(regfit.ss,7)
 (Intercept)        AtBat         Hits        Walks        Years        CHits    DivisionW      PutOuts
 401.9777394   -2.4522168    7.5045989    4.4089801  -48.2143365    0.6684161 -125.0446386    0.2259642 )
```

# Baseball Salaries

```
# That was tedious, so make it into a function we can use later
predict.regsubsets = function(object, newdata, id,...)
{
  form = as.formula(object$call[[2]]) # extract model ("call")
  mat = model.matrix (form, newdata)
  coefi = coef(object, id=id)
  xvars = names(coefi)
  mat[,xvars] %*% coefi
}
```

```
And now do best subset selection on the FULL dataset, choose the best 7-variable model
regfit.ss = regsubsets(Salary~., data=Hitters, nvmax =19)
```

```
# these are different from before
coef(regfit.ss,7)
 (Intercept)         Hits        Walks       CAtBat        CHits       CHmRun    DivisionW      PutOuts
  79.4509472    1.2833513    3.2274264   -0.3752350    1.4957073    1.4420538 -129.9866432    0.2366813
```

**So we need to choose a model using some kind of cross-validation or holdout sample!**

# Model Selection using CV

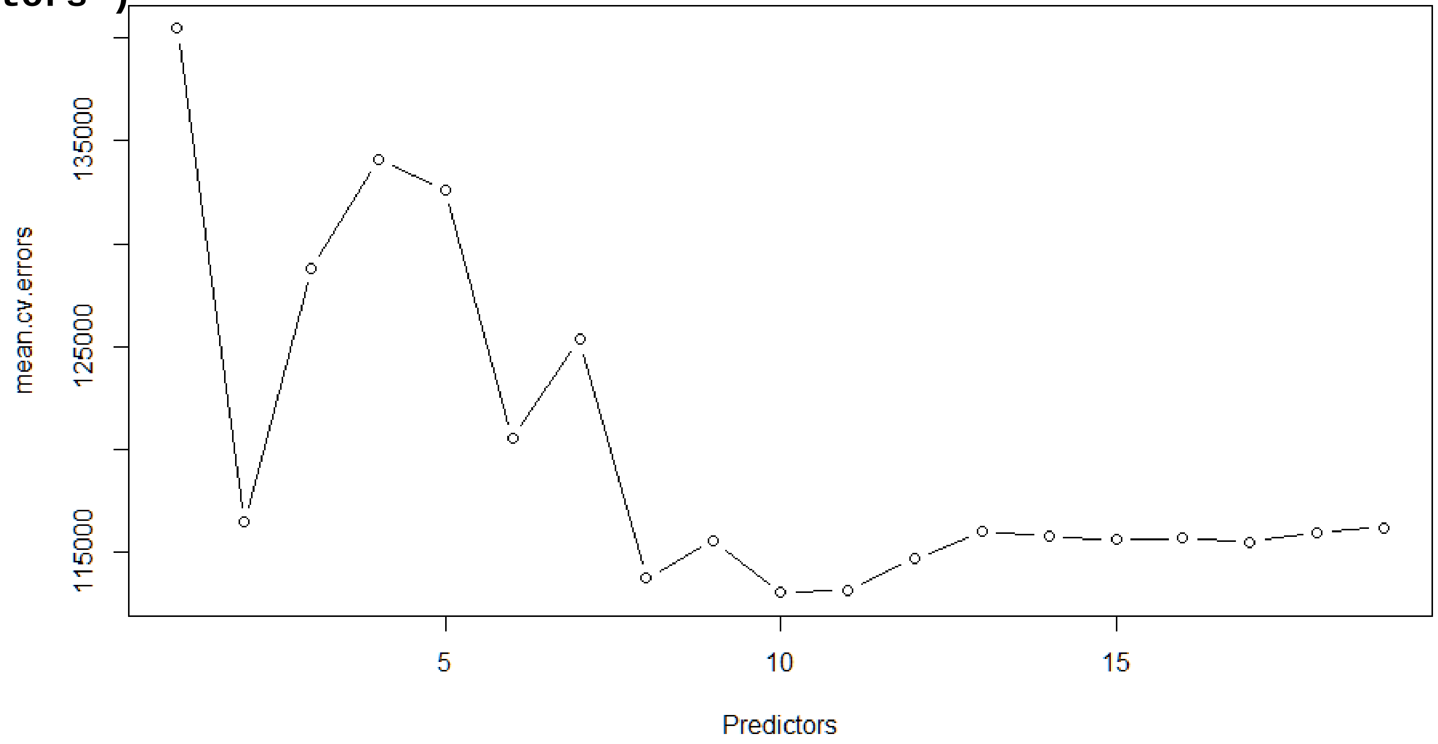
```
# set up k-fold cross-validation of MSE
# set up k folds
k = 10
set.seed(7)
folds = sample(1:k, nrow(Hitters), replace = TRUE)

# set up cv MSE table
cv.mse = matrix (NA ,k,19, dimnames =list(NULL , paste (1:19) ))

# loop through all folds
for(j in 1:k)
{
  best.ss = regsubsets (Salary ~., data = Hitters [folds != j,], nvmax =19)

  # compile mean MSE of all folds for each number of predictors
  for(i in 1:19)
  {
    pred = predict(best.ss, Hitters[folds == j,], id=i)
    cv.mse[j,i] = mean((Hitters$Salary[folds == j] - pred)^2)
  }
}
```

# Model Selection using CV

```
cv.mse  # the MSEs for each fold and number of predictors
              1         2         3         4         5         6         7         8         9        10
 [1,] 145608.84 131762.73 140073.23 132248.48 142804.42 128394.39 148369.98 117656.34 138676.15 133762.79
 [2,]  63622.35  36320.26  43528.06  35784.03  67546.33  81480.04  80170.10  79475.24  77891.09  86409.62
 [3,] 128478.90 116278.29 141513.75 234357.39 198646.23 185804.96 167033.74 154538.53 151378.01 147499.39
 [4,] 223570.13 236372.82 246601.89 230305.38 223016.63 210604.53 211196.15 202203.66 206103.83 202211.08
 [5,] 172322.29 139474.55 127043.54 110167.90 106235.44  96087.32 100545.39  91107.10  92735.19  84730.93
 [6,] 116075.54  69920.61  71025.79  89086.82  88879.24  64408.89  67426.49  51126.83  48231.82  47502.24
 [7,] 153182.53 131082.43 135296.19 127391.45 139288.29 133766.89 156842.06 150425.78 151271.81 140891.29
 [8,] 160762.16 124524.01 127529.10 121597.00 117507.07 115600.69 125954.18 126804.69 120849.30 109752.49
 [9,] 139890.39  85665.74 139480.88 141344.23 127102.27 100684.38  96255.76  75131.95  76115.17  88279.40
[10,] 101075.00  93634.05 115987.71 118739.58 115093.18  88762.29  99741.85  89276.23  92860.47  89500.83
             11        12        13        14        15        16        17        18        19
 [1,] 144181.75 135771.72 135419.66 135059.83 132586.51 132398.57 131207.58 130975.75 131304.17
 [2,]  82677.71  87478.21  87486.59  89330.13  87220.62  87009.28  86606.26  87889.09  87617.35
 [3,] 144496.02 148836.88 149032.48 144943.91 144883.43 142465.51 143618.46 147276.90 148109.69
 [4,] 197894.80 196801.62 196839.07 196803.42 196065.19 196210.47 196024.16 195892.06 196253.71
 [5,]  86572.38  84791.78  87977.59  89495.22  91199.72  89160.49  89072.61  88996.76  88941.38
 [6,]  50748.42  53466.71  54803.84  53730.70  54001.73  54011.71  53897.82  53620.95  53591.54
 [7,] 142970.72 148645.01 149378.21 147191.63 148025.88 148332.05 148756.35 148362.74 149007.97
 [8,] 116903.14 123422.39 122602.59 126075.47 126011.80 126046.54 124923.79 125631.71 125199.61
 [9,]  73492.93  78449.44  85158.42  85040.60  86586.44  91701.31  91651.47  91514.78  92895.69
[10,]  91627.29  89333.78  91634.03  90359.54  89654.58  89800.63  89628.77  89511.37  89455.14
```

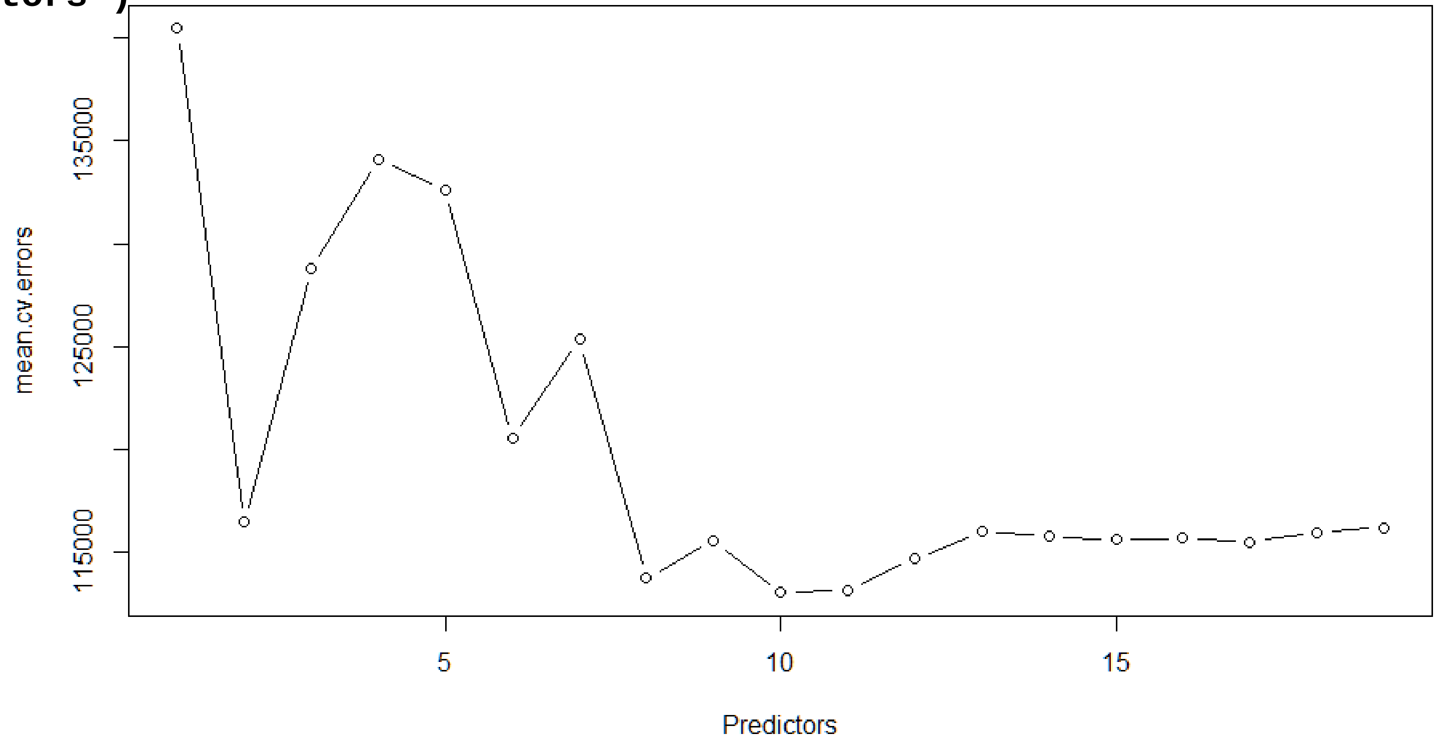
# Model Selection using CV

```
# get the mean of the means after run
mean.cv.mse = apply(cv.errors, 2, mean)
mean.cv.mse
 1        2        3        4        5        6        7        8        9        10       11
140458.8 116503.5 128808.0 134102.2 132611.9 120559.4 125353.6 113774.6 115611.3 113054.0 113156.5
      12       13       14       15       16       17       18       19
114699.8 116033.2 115803.0 115623.6 115713.7 115538.7 115967.2 116237.6

par(mfrow = c(1,1))
plot(mean.cv.errors, type = 'b', xlab = 'Predictors')


# what about the "NULL" model?
# What IS the NULL model?
```



# Model Selection using CV

```
# get the mean of the means after run
mean.cv.mse = apply(cv.errors, 2, mean)
mean.cv.mse
 1        2        3        4        5        6        7        8        9        10       11
140458.8 116503.5 128808.0 134102.2 132611.9 120559.4 125353.6 113774.6 115611.3 113054.0 113156.5
      12       13       14       15       16       17       18       19
114699.8 116033.2 115803.0 115623.6 115713.7 115538.7 115967.2 116237.6

par(mfrow = c(1,1))
plot(mean.cv.errors, type = 'b', xlab = 'Predictors')


# what about the "NULL" model?
# What IS the NULL model?
# Using just the mean!

mean((Hitters$Salary - mean(Hitters$Salary))^2)
[1] 202734.3
```



# Model Selection using CV

```
# Let's use 10 predictors and get "final" model, we can save time by limiting nvmax
reg.final = regsubsets(Salary~., data = Hitters, nvmax = 11)

coef(reg.final, 10)
 (Intercept)        AtBat         Hits        Walks       CAtBat        CRuns         CRBI       CWalks
 162.5354420   -2.1686501    6.9180175    5.7732246   -0.1300798    1.4082490    0.7743122   -0.8308264
   DivisionW      PutOuts      Assists
-112.3800575    0.2973726    0.2831680

# 11 produces the same as on page 250-251
coef(reg.final, 11)
 (Intercept)        AtBat         Hits        Walks       CAtBat        CRuns         CRBI       CWalks
 135.7512195   -2.1277482    6.9236994    5.6202755   -0.1389914    1.4553310    0.7852528   -0.8228559
     LeagueN    DivisionW      PutOuts      Assists
  43.1116152 -111.1460252    0.2894087    0.2688277
```

# Ridge Regression

```
# glmnet uses an x matrix and y, not a model ~ statement
# model.matrix changes factors into numbers
x <- model.matrix (Salary~.,Hitters )[,-1]
y <- Hitters$Salary

require(glmnet)

# set up a grid of values for lambda; the higher the lambda, the greater the shrinkage penalty
grid <- 10^seq (10,-2, length =100)
grid
  [1] 1.000000e+10 7.564633e+09 5.722368e+09 4.328761e+09 3.274549e+09 2.477076e+09 1.873817e+09 1.417474e+09
  [9] 1.072267e+09 8.111308e+08 6.135907e+08 4.641589e+08 3.511192e+08 2.656088e+08 2.009233e+08 1.519911e+08
 [17] 1.149757e+08 8.697490e+07 6.579332e+07 4.977024e+07 3.764936e+07 2.848036e+07 2.154435e+07 1.629751e+07
 [25] 1.232847e+07 9.326033e+06 7.054802e+06 5.336699e+06 4.037017e+06 3.053856e+06 2.310130e+06 1.747528e+06
 [33] 1.321941e+06 1.000000e+06 7.564633e+05 5.722368e+05 4.328761e+05 3.274549e+05 2.477076e+05 1.873817e+05
 [41] 1.417474e+05 1.072267e+05 8.111308e+04 6.135907e+04 4.641589e+04 3.511192e+04 2.656088e+04 2.009233e+04
 [49] 1.519911e+04 1.149757e+04 8.697490e+03 6.579332e+03 4.977024e+03 3.764936e+03 2.848036e+03 2.154435e+03
 [57] 1.629751e+03 1.232847e+03 9.326033e+02 7.054802e+02 5.336699e+02 4.037017e+02 3.053856e+02 2.310130e+02
 [65] 1.747528e+02 1.321941e+02 1.000000e+02 7.564633e+01 5.722368e+01 4.328761e+01 3.274549e+01 2.477076e+01
 [73] 1.873817e+01 1.417474e+01 1.072267e+01 8.111308e+00 6.135907e+00 4.641589e+00 3.511192e+00 2.656088e+00
 [81] 2.009233e+00 1.519911e+00 1.149757e+00 8.697490e-01 6.579332e-01 4.977024e-01 3.764936e-01 2.848036e-01
 [89] 2.154435e-01 1.629751e-01 1.232847e-01 9.326033e-02 7.054802e-02 5.336699e-02 4.037017e-02 3.053856e-02
 [97] 2.310130e-02 1.747528e-02 1.321941e-02 1.000000e-02
```

# Ridge Regression

```
#### glmnet automatically scales predictors #### (unless: ,standardize = F)
# if alpha = 0, perform ridge regression
ridge.mod <- glmnet(x, y, alpha = 0, lambda = grid)

coef(ridge.mod) # 20 predictors, 100 lambdas
20 x 100 sparse Matrix of class "dgCMatrix"
   [[ suppressing 81 column names ‘s0’, ‘s1’, ‘s2’ ... ]]
   [[ suppressing 81 column names ‘s0’, ‘s1’, ‘s2’ ... ]]

(Intercept) 5.359257e+02 5.359256e+02 5.359256e+02 5.359254e+02 5.359253e+02 5.359251e+02 5.359249e+02 5.359246e+02 5.359241e+02 5.359236e+02 5.359228e+02 5.359218e+02
AtBat       5.443467e-08 7.195940e-08 9.512609e-08 1.257511e-07 1.662355e-07 2.197535e-07 2.905011e-07 3.840251e-07 5.076583e-07 6.710939e-07 8.871458e-07 1.172753e-06
Hits        1.974589e-07 2.610289e-07 3.450649e-07 4.561554e-07 6.030105e-07 7.971441e-07 1.053777e-06 1.393031e-06 1.841504e-06 2.434358e-06 3.218075e-06 4.254101e-06
HmRun       7.956523e-07 1.051805e-06 1.390424e-06 1.838059e-06 2.429805e-06 3.212059e-06 4.246151e-06 5.613159e-06 7.420260e-06 9.809139e-06 1.296709e-05 1.714170e-05
Runs        3.339178e-07 4.414196e-07 5.835307e-07 7.713931e-07 1.019736e-06 1.348031e-06 1.782017e-06 2.355720e-06 3.114121e-06 4.116682e-06 5.442006e-06 7.194001e-06
RBI         3.527222e-07 4.662778e-07 6.163918e-07 8.148335e-07 1.077162e-06 1.423944e-06 1.882370e-06 2.488380e-06 3.289490e-06 4.348509e-06 5.748467e-06 7.599123e-06

ridge.mod$lambda [50]
[1] 11497.57
coef(ridge.mod)[,50]
coef(ridge.mod)[,50]
 (Intercept)        AtBat         Hits        HmRun         Runs          RBI        Walks        Years       CAtBat        CHits
407.356050200  0.036957182  0.138180344  0.524629976  0.230701523  0.239841459  0.289618741  1.107702929  0.003131815  0.011653637
      CHmRun        CRuns         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists       Errors   NewLeagueN
  0.087545670  0.023379882  0.024138320  0.025015421  0.085028114 -6.215440973  0.016482577  0.002612988 -0.020502690  0.301433531
```

# Ridge Regression

```
ridge.mod$lambda [50] # a "large" value, should shrink coefficients
[1] 11497.57
coef(ridge.mod)[,50]
 (Intercept)        AtBat         Hits        HmRun         Runs          RBI        Walks        Years       CAtBat        CHits 
407.356050200  0.036957182  0.138180344  0.524629976  0.230701523  0.239841459  0.289618741  1.107702929  0.003131815  0.011653637 
      CHmRun        CRuns         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists       Errors   NewLeagueN 
  0.087545670  0.023379882  0.024138320  0.025015421  0.085028114 -6.215440973  0.016482577  0.002612988 -0.020502690  0.301433531 

ridge.mod$lambda [60] # a smaller value, shrinks coefficients less
[1] 705.4802
coef(ridge.mod)[,60]
 (Intercept)        AtBat         Hits        HmRun         Runs          RBI        Walks        Years       CAtBat        CHits 
 54.32519950   0.11211115   0.65622409   1.17980910   0.93769713   0.84718546   1.31987948   2.59640425   0.01083413   0.04674557 
      CHmRun        CRuns         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists       Errors   NewLeagueN 
  0.33777318   0.09355528   0.09780402   0.07189612  13.68370191 -54.65877750   0.11852289   0.01606037  -0.70358655   8.61181213 

# We can get coefficients for a specific lambda(s) - round for looking at
round(predict(ridge.mod, s=50, type = "coefficients")[1:20,],3)
(Intercept)       AtBat        Hits       HmRun        Runs         RBI       Walks       Years      CAtBat       CHits 
     48.766      -0.358       1.969      -1.278       1.146       0.804       2.716      -6.218       0.005       0.106 
     CHmRun       CRuns        CRBI      CWalks     LeagueN   DivisionW     PutOuts     Assists      Errors  NewLeagueN 
      0.624       0.221       0.219      -0.150      45.926    -118.201       0.250       0.122      -3.279      -9.497 
```

# Ridge Regression

```
# let's get the mean MSE in a test sample
set.seed(7)
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train )
y.test <- y[test]

# use the grid again
ridge.mod <- glmnet(x[train,], y[train], alpha =0, lambda =grid, thresh = 1e-12)

# predict using glmnet : note the use of newx
ridge.pred <- predict(ridge.mod, s = 4, newx = x[test ,])
mean((ridge.pred - y.test)^2)
# [1] 142668
ridge.pred
                            1
-Alan Ashby        379.96271
-Andres Galarraga  443.53309
-Alfredo Griffin   542.27969
-Al Newman         271.22654
-Andres Thomas     105.28723
-Alan Trammell     788.37334
-Alex Trevino      221.63960
-Andy VanSlyke     406.59938
```

```
# NULL model
mean((mean(y[train])- y.test)^2)
[1] 257648.4

# hmmmm
ridge.pred <- predict(ridge.mod, s=1e10, newx=x[test,])
mean((ridge.pred - y.test)^2)
[1] 257648.4
```

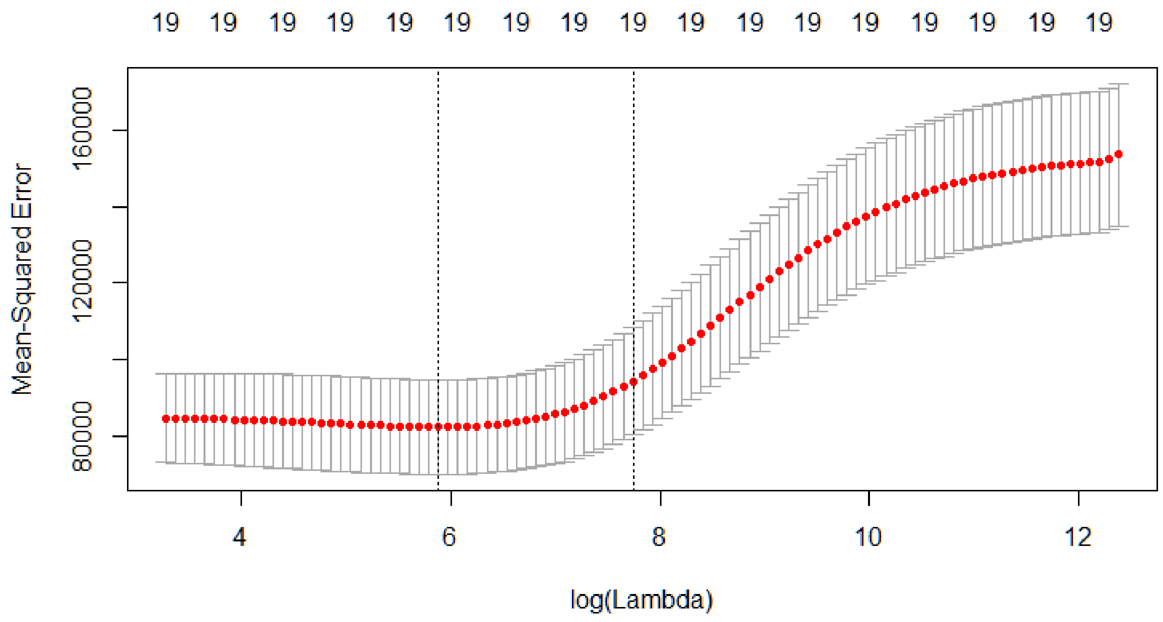
# Ridge Regression

```
# test linear regression (set lambda = 0)
ridge.pred <- predict(ridge.mod, s = 0, newx = x[test,])
mean((ridge.pred - y.test)^2)
[1] 141620.5
lm(y~x, subset =train)
Call:
lm(formula = y ~ x, subset = train)

Coefficients:
(Intercept)       xAtBat        xHits       xHmRun        xRuns         xRBI       xWalks       xYears      xCAtBat       xCHits
   90.46667     -1.93630      6.92120     -2.29256      0.75298      0.74394      3.33849      5.85338     -0.16762      0.50393
    xCHmRun       xCRuns        xCRBI      xCWalks     xLeagueN   xDivisionW     xPutOuts     xAssists      xErrors  xNewLeagueN
    2.41127      0.72515      0.01203     -0.73113    191.15886    -67.36335      0.21135      0.47537     -7.03317   -147.83212

round(predict(ridge.mod, s = 0,type = "coefficients") [1:20,],5)
(Intercept)      AtBat       Hits      HmRun       Runs        RBI      Walks      Years     CAtBat      CHits
   90.38494   -1.93801    6.93392   -2.26032    0.74132    0.73435    3.34119    5.76525   -0.16494    0.49189
     CHmRun     CRuns       CRBI     CWalks    LeagueN  DivisionW    PutOuts    Assists     Errors NewLeagueN
    2.38881    0.72991    0.01959   -0.73233  191.09147  -67.36116    0.21138    0.47427   -7.02181 -147.73255
```

# RR: Choosing the best λ using CV

Naturally, we will use cross-validation

```
# We can use the built-in cv.glmnet() function that will test many values of lambda
set.seed (7)
cv.out <- cv.glmnet(x[train,], y[train], alpha = 0)
plot(cv.out)
bestlam <- cv.out$lambda.min
bestlam
[1] 357.2905
```



# RR: Choosing the best $\lambda$ using CV

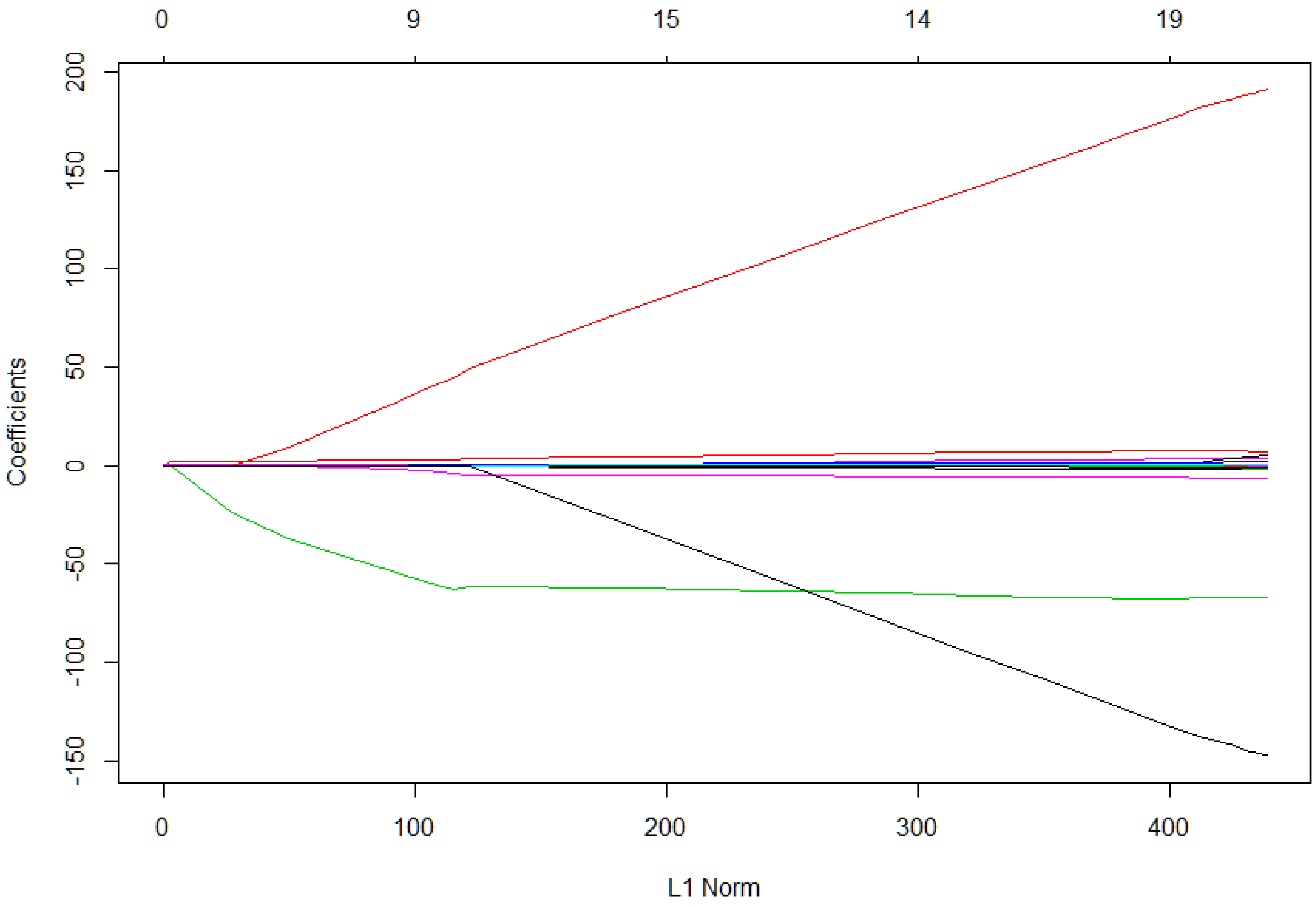
```
# What improvement in the MSE? What are the coefficients?
ridge.pred <- predict(ridge.mod, s = bestlam, newx = x[test,])
mean((ridge.pred - y.test)^2)
[1] 158571.7

# in this case, it got worse, in contrast to chapter results!

# ridge again
rreg2 <- glmnet(x, y, alpha = 0)

predict(rreg2, type = "coefficients", s = bestlam)[1:20,]
 (Intercept)        AtBat         Hits        HmRun         Runs          RBI        Walks        Years       CAtBat        CHits
 17.98653140   0.08411834   0.83177568   0.68925817   1.05138274   0.87894452   1.58791179   1.56259857   0.01134104   0.05604141
      CHmRun        CRuns         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists       Errors   NewLeagueN
  0.39785795   0.11183616   0.11813278   0.05673971  21.05095331 -76.20171988   0.16058669   0.02721512  -1.27171087   9.30936778

# from chapter:
(Intercept)        AtBat       Hits        HmRun        Runs        RBI       Walks       Years      CAtBat       CHits
     9.8849       0.0314     1.0088       0.1393      1.1132     0.8732      1.8041      0.1307      0.0111      0.0649
    CHmRun        CRuns       CRBI       CWalks     LeagueN     DivisionW    PutOuts    Assists       Errors   NewLeagueN
    0.4516        0.129     0.1374       0.0291     27.1823      -91.6341     0.1915     0.0425      -1.8124       7.2121
```

# RR: REALLY Choosing the best λ using CV

```
DRAFT CODE - to be improved
# set up k-fold cross-validation of MSE
# set up k folds
k = 10
# set.seed(7)

# set up cv MSE table
cv.mse = matrix (NA ,k,length(grid), dimnames =list(NULL , paste (1:100) ))


# loop through all folds
  # compile mean MSE of all folds for each number of predictors
for(i in 1:length(grid))
{
   for(j in 1:k)
  {
   folds = sample(1:k, nrow(Hitters), replace = TRUE)
   ridge.mod = cv.glmnet(x[train,], y[train], alpha = 0)
   ridge.pred <- predict(ridge.mod, s=grid[i], newx=x[test,])
   cv.mse[j,i] = mean((Hitters$Salary[folds == j] - ridge.pred)^2)
  }
  }
}
```

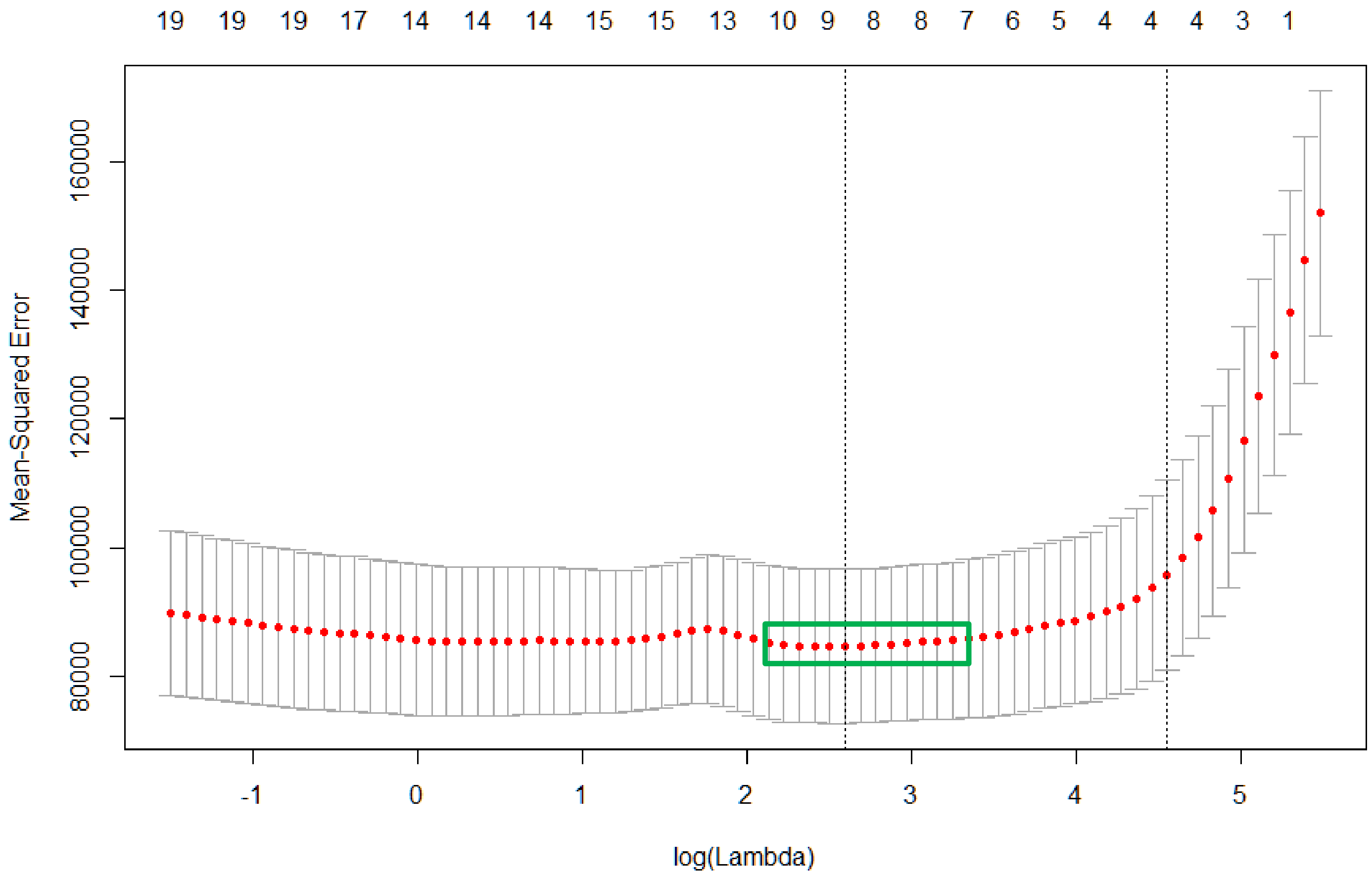
# The Lasso

```
#### glmnet automatically scales predictors #### (unless: ,standardize = F)
# if alpha = 1, perform the Lasso
lasso.mod <- glmnet(x[train,], y[train], alpha = 1, lambda = grid)
plot(lasso.mod)
```



# The Lasso

```
#Let's cross-validate to get the best lambda
set.seed(7)
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] 152424.1

bestlam
[1] 13.45176
log(bestlam)
[1] 2.59911
```



# The Lasso

```
#Let's use a lambda of 2.8
lasso.pred <- predict(lasso.mod, s = 2.8, newx=x[test,])
mean((lasso.pred - y.test)^2) # before was 152424.1
[1] 142898.3

#Let's use a lambda of 2.4
lasso.pred <- predict(lasso.mod, s = 2.4, newx=x[test,])
mean((lasso.pred - y.test)^2) # before was 152424.1
[1] 142391.8
```

**So we will need to sample multiple times and/or average our models!**