ISLR | Chapter 6 Exercises

Marshall McQuillen

8/3/2018

Conceptual

1

- **A**. For a model with k predictors, Best Subset Selection will always have the best *training* RSS. The reason for this is, given a fixed k, there are $\binom{p}{k}$ possible models, and Best Subset Selection considers all of those $\binom{p}{k}$ possibilities.

  In Forward Stepwise Selection, of the total $\binom{p}{k}$ possible models, only the models that contain the $(k-1)$ model produced by Forward Stepwise Selection will be considered for the "best" k-variable model.

  In Backward Stepwise Selection, of the total $\binom{p}{k}$ possible models, the predictors in the "best" k-variable model *must* be a subset of the model with $(k+1)$ predictors.

  In short, Best Subset Selection will have the best (lowest) *training* RSS for a model with k predictors because it considers **all** the possible $\binom{p}{k}$ models, whereas Forward and Backward Stepwise Selection only consider a **subset** of all the possible $\binom{p}{k}$ models.

- **B**. There is no definitive answer for which subset selection method will have the lowest *testing* RSS (overfitting). If there is a large number of predictors, Best Subset Selection has the possibility of finding a model that has a low training RSS but a high testing RSS. Cross validation could be used to estimate the testing error of three models (one for Best Subset Selection, one for Forward Stepwise Selection and one for Backward Stepwise Selection) and a decision on which model has the lowest testing RSS could be made in consideration of the CV results.

- **C**.

  i. True.

  ii. True.

  iii. False, the predictors in the k-variable model identified by Backward Subset Selection are **not** a subset of the predictors in the $(k+1)$-variable model identified by Forward Subset Selection.

  iv. False, the predictors in the k-variable model identified by Forward Stepwise Selection are **not** a subset of the predictors in the $(k+1)$-variable model identified by Backward Stepwise Selection.

  v. False, the predictors in the k-variable model identified by Best Subset Selection are **not necessarily** a subset of the predictors in the $(k+1)$-variable model identified by Best Subset Selection.

2

- **A**. The lasso, relative to least squares is, *iii*, less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

- **B**. Ridge Regression, relative to least squares is, *iii*, less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

- **C**. Non-linear methods, relative to least squares are, *ii*, more flexible and hence will give improved prediction accuracy when their increase in variance is less than their decrease in bias.

3

In the (alternate) cost function for the Lasso…

$$\sum_{i=1}^{n}\left(y_k - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 \; subject\ to \sum_{j=1}^{p}|\beta_j| \leq s$$

As we increase s from 0…

- **A**. …the training RSS will, *iv*, steadily decrease. As the budget (s) for the sum of the regression coefficients increases from 0, each β_j will approach the value it would reach in ordinary least squares regression (no constraint). Therefore, without a constraint, we are letting the regression coefficients "roam freely" to reach their ordinary least squares values. Using the constraint, we are "lassoing" them in (pun intended).
- **B**. …the testing RSS will, *ii*, decrease initially, and then eventually start increasing in a U shape. When s is 0, all the regression coefficients are 0, and the "model" is simply the intercept, β_0, the mean of the response (highly biased, very low variance). As s increases from 0, the model becomes more flexible, allowing for an increasingly better fit to the data up to a point (bottom of the U). Once this point is reached, the model becomes **overly** flexible, overfitting the training data and leading to an increase in the test error (right side of U).
- **C**. …variance will, *iii*, steadily increase. As s increases from 0, the model becomes more and more flexible, and as we know, more flexible models have a higher variance and lower bias. The model will be more influenced by the data it is trained on.
- **D**. …(squared) bias will, *iv*, steadily decrease. As s increases from 0, the model becomes more and more flexible, and as we know, more flexible models have a higher variance and lower bias. The model will have a better chance of representing the *true* relationship between the predictors and the response.
- **E**. …the irreducible error will, *v*, remain constant. The irreducible error is just that, **irreducible**.

4

In the cost function for Ridge Regression…

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

…as we increase λ from 0…

- **A**. …the training RSS will, *iii*, steadily increase. As λ increases from 0, more "weight" is given to the second term in the cost function, thus penalizing large regression coefficients more and more. Increasing λ from 0 restricts the regression coefficients more and more.
- **B**. …the testing RSS will, *ii*, decrease initially, and then eventually start increasing in a U shape. As λ increases from 0, increasingly strict restrictions are put on the magnitude that the regression coefficients can grow too. This has the effect of making the model less flexible and more generalizable, **up to a point**. The left side of the U represents the model's increase in bias being less than it's decrease in variance, and the right side of the U represents the decrease in variance no longer being worth the increase in bias.
- **C**. …the variance will, *iv*, steadily decrease. As λ increases from 0, increasingly strict limits are placed on the regression coefficients, making it less flexible.

- **D**. …the (squared) bias will, *iii*, steadily increase. As λ increases from 0, increasingly strict limits are placed on the regression coefficients, making it less flexible. This will make it increasingly harder for the model to estimate the true relationship between the response and the predictors.
- **E**. …the irreducible error will, *v*, remain constant. The irreducible error is just that, **irreducible**.

5

- **A**. The optimization problem for Ridge Regression is:

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

  Written out when $n = p = 2$, the equation comes to:

$$(y_1 - \beta_0 - \beta_1 x_{11} - \beta_2 x_{12})^2 + (y_2 - \beta_0 - \beta_1 x_{21} - \beta_2 x_{22})^2 + \lambda\left(\beta_1^2 + \beta_2^2\right)$$

  When $\beta_0 = 0$, a small simplification can be made such that the above equation becomes:

$$(y_1 - \beta_1 x_{11} - \beta_2 x_{12})^2 + (y_2 - \beta_1 x_{21} - \beta_2 x_{22})^2 + \lambda\left(\beta_1^2 + \beta_2^2\right)$$

- **B**. Given that $x_{11} = x_{12}$ and $x_{21} = x_{22}$, I will refer to these as simply x_1 and x_2 respectively where applicable. A small rewrite of the preceding equation gives:

$$f(x) = (y_1 - \beta_1 x_1 - \beta_2 x_1)^2 + (y_2 - \beta_1 x_2 - \beta_2 x_2)^2 + \lambda\left(\beta_1^2 + \beta_2^2\right)$$

  Given that the problem above is one of *minimization*, taking the derivative is the first step in showing that $\beta_1 = \beta_2$.

  Calculating the derivative with respect to β_1 can be broken down into the derivative of the three separate terms in the above equation:

$$\frac{\partial}{\partial\beta_1}f(x) = \frac{\partial}{\partial\beta_1}(y_1 - \beta_1 x_1 - \beta_2 x_1)^2 + \frac{\partial}{\partial\beta_1}(y_2 - \beta_1 x_2 - \beta_2 x_2)^2 + \frac{\partial}{\partial\beta_1}\left(\lambda\beta_1^2 + \lambda\beta_2^2\right)$$

  First Term Derivative

$$\frac{\partial}{\partial\beta_1}(y_1 - \beta_1 x_1 - \beta_2 x_1)^2 = 2(y_1 - \beta_1 x_1 - \beta_2 x_1)\cdot(-x_1) = 2\left(-y_1 x_1 + \beta_1 x_1^2 + \beta_2 x_1^2\right)$$

  Second Term Derivative

$$\frac{\partial}{\partial\beta_1}(y_2 - \beta_1 x_2 - \beta_2 x_2)^2 = 2(y_2 - \beta_1 x_2 - \beta_2 x_2)\cdot(-x_2) = 2\left(-y_2 x_2 + \beta_1 x_2^2 + \beta_2 x_2^2\right)$$

  Third Term Derivative

$$\frac{\partial}{\partial\beta_1}\left(\lambda\beta_1^2 + \lambda\beta_2^2\right) = 2\lambda\beta_1$$

Bringing this all together, the derivative of the full equation comes out to:

$$\frac{\partial}{\partial \beta_1} f(x) = 2\left(-y_1 x_1 + \beta_1 x_1^2 + \beta_2 x_1^2\right) + 2\left(-y_2 x_2 + \beta_1 x_2^2 + \beta_2 x_2^2\right) + 2\lambda\beta_1$$

Setting the derivative equal to 0 and solving for β_1:

$$2\left(-y_1 x_1 + \beta_1 x_1^2 + \beta_2 x_1^2\right) + 2\left(-y_2 x_2 + \beta_1 x_2^2 + \beta_2 x_2^2\right) + 2\lambda\beta_1 = 0$$

Divide by 2:

$$\left(-y_1 x_1 + \beta_1 x_1^2 + \beta_2 x_1^2\right) + \left(-y_2 x_2 + \beta_1 x_2^2 + \beta_2 x_2^2\right) + \lambda\beta_1 = 0$$

Group β_1 terms and β_2 terms together:

$$\left(\beta_1 x_1^2 + \beta_1 x_2^2 + \lambda\beta_1\right) + \beta_2 x_1^2 + \beta_2 x_2^2 - y_1 x_1 - y_2 x_2 = 0$$

Factor out β_1 and β_2:

$$\beta_1\left(x_1^2 + x_2^2 + \lambda\right) + \beta_2\left(x_1^2 + x_2^2\right) - y_1 x_1 - y_2 x_2 = 0$$

Add $y_1 x_1$ and $y_2 x_2$ to both sides of the equation:

$$\beta_1\left(x_1^2 + x_2^2 + \lambda\right) + \beta_2\left(x_1^2 + x_2^2\right) = y_1 x_1 + y_2 x_2$$

Repeating the same process, this time taking the derivative **with respect to** β_2, would yield:

$$\beta_2\left(x_1^2 + x_2^2 + \lambda\right) + \beta_1\left(x_1^2 + x_2^2\right) = y_1 x_1 + y_2 x_2$$

Using substitution, we can set the two equations equal to each other:

$$\beta_1\left(x_1^2 + x_2^2 + \lambda\right) + \beta_2\left(x_1^2 + x_2^2\right) = \beta_2\left(x_1^2 + x_2^2 + \lambda\right) + \beta_1\left(x_1^2 + x_2^2\right)$$

Factoring λ out into it's own term:

$$\beta_1\left(x_1^2 + x_2^2\right) + \beta_1\lambda + \beta_2\left(x_1^2 + x_2^2\right) = \beta_2\left(x_1^2 + x_2^2\right) + \beta_2\lambda + \beta_1\left(x_1^2 + x_2^2\right)$$

Subract $\beta_1\left(x_1^2 + x_2^2\right)$ and $\beta_2\left(x_1^2 + x_2^2\right)$ then divide by λ:

$$\beta_1\lambda = \beta_2\lambda \quad thus \quad \beta_1 = \beta_2$$

- **C**. The lasso optimization problem is similar to that of Ridge Regression, with a small change to the third term in the equation:

$$f(x) = \left(y_1 - \beta_1 x_1 - \beta_2 x_1\right)^2 + \left(y_2 - \beta_1 x_2 - \beta_2 x_2\right)^2 + \lambda\left(|\beta_1| + |\beta_2|\right)$$

- **D**. The reason that the lasso coefficients are not unique can be best approached through geometric and visual explanation.

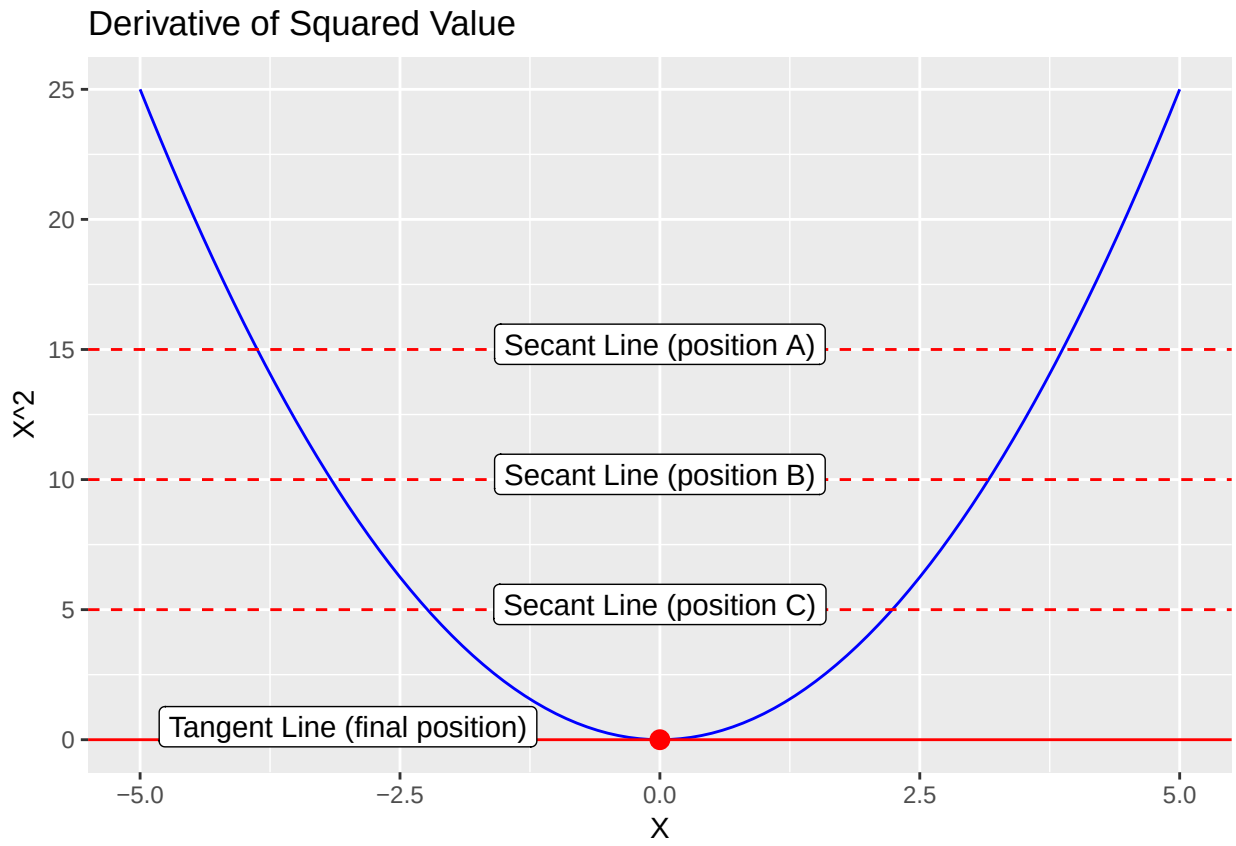
It is known that in order to find the *minimum* of a function, one takes the derivative of said function, sets it (the derivative) equal to 0, and solves for the unkown variable (this by itself does not ensure a *minimum* - the second derivative of the function evaluated at x must also be positive). In this case, the unknown variables are β_1 and β_2.

The problem arises when one tries to take the derivative of the penalty term, shown below:

$$\frac{\partial}{\partial(\beta_1)}(\lambda(|\beta_1| + |\beta_2|))$$

The geometric manner of understanding the derivative can be understood as the slope of *the* tangeant line to the function $f(x)$ evaluated at x (the operative word being "the," not "a"). As the secant line moves from position A to B to C in the plot of $f(x) = x^2$ below, the two points of each secant line that intersect $f(x)$ will eventually converge to one point (red point in plot), and there wil be *one* line that is tangeant to $f(x)$ at that point. This line represents the solution to the problem.

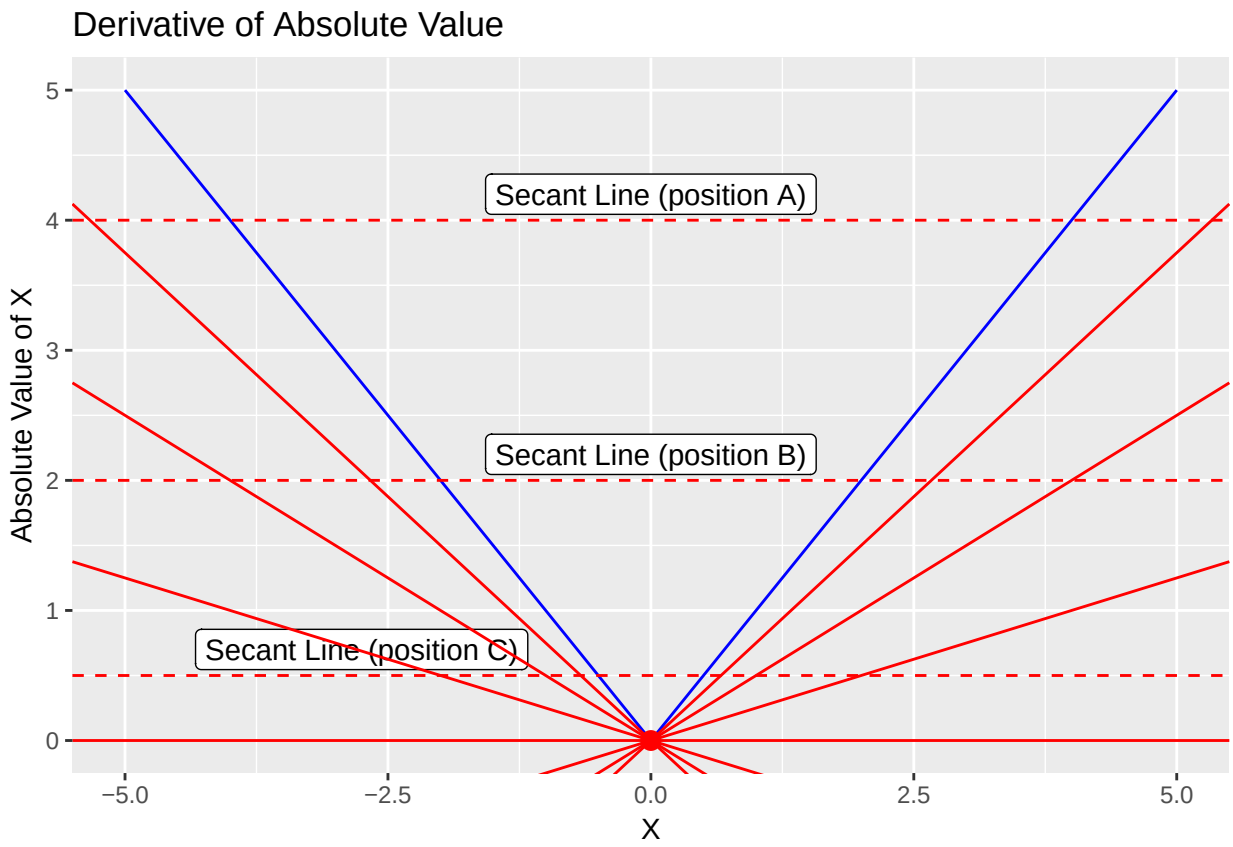
```
suppressPackageStartupMessages(library(ggplot2))
x <- seq(-5,5,0.1)

# plot X^2
label.df <- data.frame(labels = c('Secant Line (position A)',
                                  'Secant Line (position B)',
                                  'Secant Line (position C)',
                                  'Tangent Line (final position)'),
                       x = c(0, 0, 0, -3),
                       y = c(15.2, 10.2, 5.2, 0.5))

ggplot(data.frame(x = x,
                  y = x^2), aes(x, y)) +
    geom_line(col = 'blue') +
    geom_hline(yintercept = 15, col = 'red', lty = 2) +
    geom_hline(yintercept = 10, col = 'red', lty = 2) +
    geom_hline(yintercept = 5, col = 'red', lty = 2) +
    geom_hline(yintercept = 0, col = 'red', lty = 1) +
    geom_point(data = data.frame(x = 0,
                                 y = 0), aes(x, y), col = 'red', cex = 3) +
    geom_label(data = label.df, aes(x = x, y = y, label = labels)) +
    xlab("X") +
    ylab("X^2") +
    ggtitle("Derivative of Squared Value")
```



Alternatively, as shown below in the plot of $f(x) = |x|$, there are many (infinitely many to be exact), tangent lines to $f(x)$ evaluated at $x = 0$. Although the two points of the secant line to $f(x)$ eventually converge on the same point, there are numerous tangential lines at this point, each with its own unique slope. This shows that the equation is non-differentiable and therefore there is no unique solution. (more in-depth, linear algebra take on the problem here)

```
# plot abs(x)
label.df <- data.frame(labels = c('Secant Line (position A)',
                                  'Secant Line (position B)',
                                  'Secant Line (position C)'),
                       x = c(0, 0, -2.75),
                       y = c(4.2, 2.2, 0.7))

ggplot(data.frame(x = x,
                  y = abs(x)), aes(x, y)) +
    geom_line(col = 'blue') +
    geom_hline(yintercept = 4, col = 'red', lty = 2) +
    geom_hline(yintercept = 2, col = 'red', lty = 2) +
    geom_hline(yintercept = 0.5, col = 'red', lty = 2) +
    geom_label(data = label.df, aes(x = x, y = y, label = labels)) +
    geom_abline(slope = 0.75, intercept = 0, col = 'red', lty = 1) +
    geom_abline(slope = 0.5, intercept = 0, col = 'red', lty = 1) +
    geom_abline(slope = 0.25, intercept = 0, col = 'red', lty = 1) +
    geom_abline(slope = 0, intercept = 0, col = 'red', lty = 1) +
    geom_abline(slope = -0.75, intercept = 0, col = 'red', lty = 1) +
```

```
  geom_abline(slope = -0.5, intercept = 0, col = 'red', lty = 1) +
  geom_abline(slope = -0.25, intercept = 0, col = 'red', lty = 1) +
  geom_point(data = data.frame(x = 0,
                               y = 0), aes(x, y), col = 'red', cex = 3) +
  xlab("X") +
  ylab("Absolute Value of X") +
  ggtitle("Derivative of Absolute Value")
```



6

- **A**. Considering the Ridge Regression loss function where $p = 1$, the equation…

$$\sum_{j=1}^{p} (y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

…can be re-written as the following.

$$f(\beta_j) = (y_j - \beta_j)^2 + \lambda \beta_j^2$$

A quick derivation shows that equation 6.12 is solved by 6.14:

$$\frac{\partial f(\beta_j)}{\partial \beta_j} = 2\,(y_j - \beta_j)\,(-1) + 2\lambda\beta_j$$

$$2\left(y_j - \beta_j\right)(-1) + 2\lambda\beta_j = 0$$

$$-2\left(y_j - \beta_j\right) + 2\lambda\beta_j = 0$$

$$-2y_j + 2\beta_j + 2\lambda\beta_j = 0$$

$$-y_j + \beta_j + \lambda\beta_j = 0$$

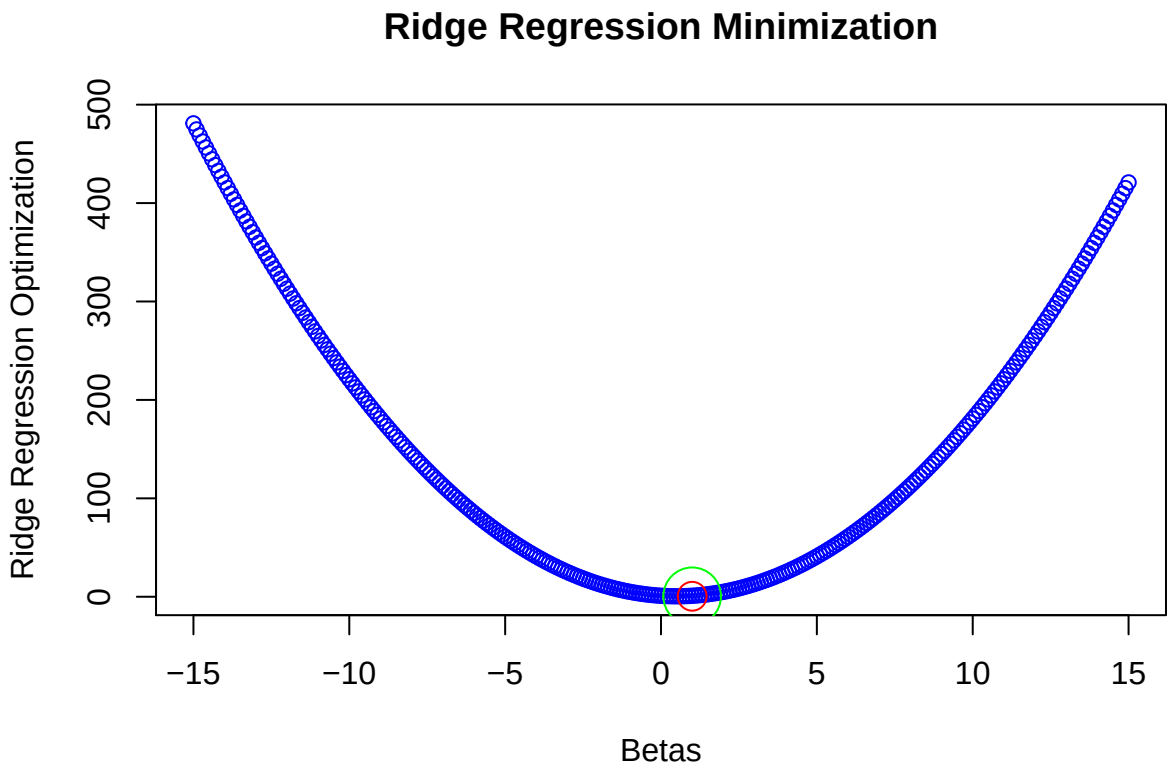
$$-y_j + \beta_j(1 + \lambda) = 0$$

$$\beta_j(1 + \lambda) = y_j$$

$$\beta_j = \frac{y_j}{(1 + \lambda)}$$

The following code holds $y_j = \lambda = 1$ and plots a sequence of β_j estimates in blue. The minimum of those estimates is circled in red, and the minimum estimated by the equation above is circled in green, both encompassing 0.5

```
ridge <- function(y, lambda, beta) {
    return((y - beta)^2 + lambda*beta^2)
}

y = 1
lambda = 1
betas <- seq(-15, 15, 0.1)

f.x <- ridge(y, lambda, betas)

function_estimate <- f.x[which.min(f.x)]
equation_estimate <- y/(lambda + 1)

plot(x = betas,
     y = f.x,
     col = "blue",
     xlab = 'Betas',
     ylab = "Ridge Regression Optimization",
     main = "Ridge Regression Minimization")
points(function_estimate, col = 'red', cex = 2)
points(equation_estimate, col = 'green', cex = 4)
```




```
function_estimate == equation_estimate
```

```
## [1] TRUE
```

- **B**. Considering the Lasso Regression loss function where $p = 1$, the equation…

$$\sum_{j=1}^{p} (y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

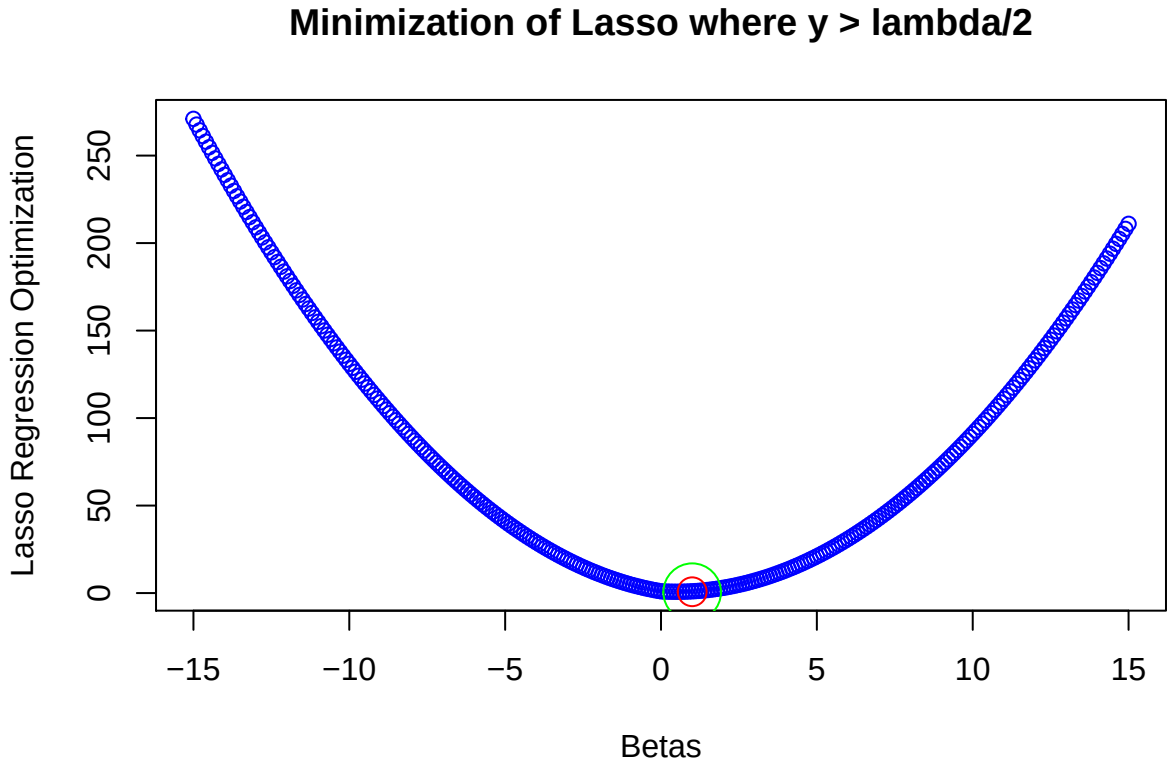
…can be re-written as the following.

$$f(\beta_j) = (y_j - \beta_j)^2 + \lambda |\beta_j|$$

The following code holds $y_j = \lambda = 1$ and plots a sequence of β_j estimates in blue. The minimum of those estimates is circled in red, and the minimum estimated by the equation (where $y_j > \frac{\lambda}{2}$) above is circled in green.

```
lasso <- function(beta, y=y, lambda=lambda) {
    return((y - beta)^2 + lambda*abs(beta))
}

y = 1
lambda = 1
betas <- seq(-15, 15, 0.1)

f.x <- lasso(betas, y, lambda)
```

```r
function_estimate <- f.x[which.min(f.x)]
equation_estimate <- y - (lambda/2)

plot(x = betas,
     y = f.x,
     col = "blue",
     xlab = 'Betas',
     ylab = "Lasso Regression Optimization",
     main = "Minimization of Lasso where y > lambda/2")
points(function_estimate, col = 'red', cex = 2)
points(equation_estimate, col = 'green', cex = 4)
```



7

(see references 1, 2 and pages 265 - 270 of ESL for more information)

- **A**. The Frequentist perspective, shown below, models the likelihood of the data as the element wise product of the output of the assumed PDF or PMF (the Gaussian PDF in this case), governed by a set of parameters (θ), evaluated for every x_i. The likelihood should (from the Frequentist perspective) be thought of as a function of θ, holding $\mathbf{X}$ constant, and the *Maximum Likelihood Estimate* is the parameter or set of parameters θ that maximizes this quantity.

$$L(\theta|\mathbf{X}) = \prod_{i=1}^{N} g_\theta(x_i) \quad where \quad g_\theta(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

  The above can be read as, "The likelihood that θ is the parameter or set of parameters that governs the probability distribution/population from which the data came *given the observed data*, is equal to the element wise product of the predefined probability distribution/mass function with θ as the parameter or set of parameters, evaluated for each x_i in the data set."

  With regard to linear regression, μ is the linear combination of each β_j and $x_{i,j}$, $\theta = \beta$, and x in the Gaussian PDF is y_i. Therefore, the above equation becomes:

$$L(\beta|\mathbf{X}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{\left(y_i - \beta_0 + \sum_{j=1}^{P} \beta_j x_{i,j} + \epsilon_i\right)^2}{2\sigma^2}}$$

Applied

8

- **A**.

```
set.seed(5)
X <- rnorm(n=100)
noise <- rnorm(n=100)
```

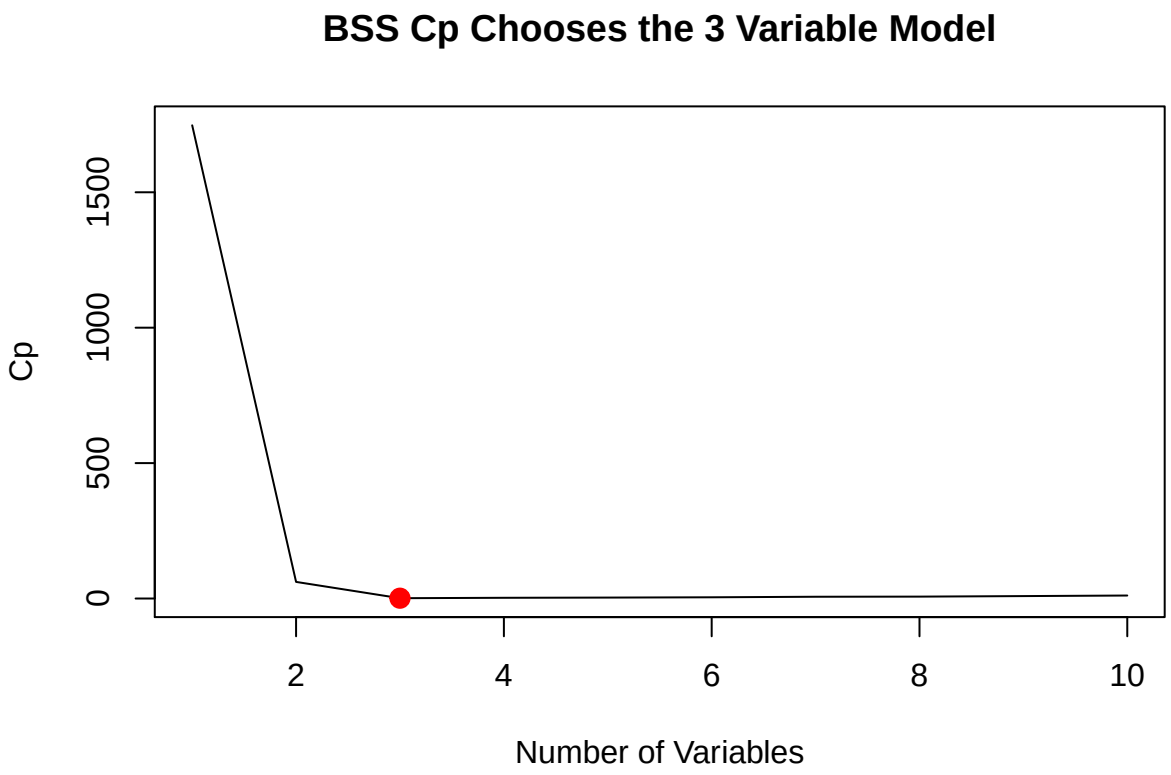
- **B**.

```
beta_0 <- 1
beta_1 <- -2
beta_2 <- 3.75
beta_3 <- 5
Y <- beta_0 + beta_1*X + beta_2*(X^2) + beta_3*(X^3) + noise
```

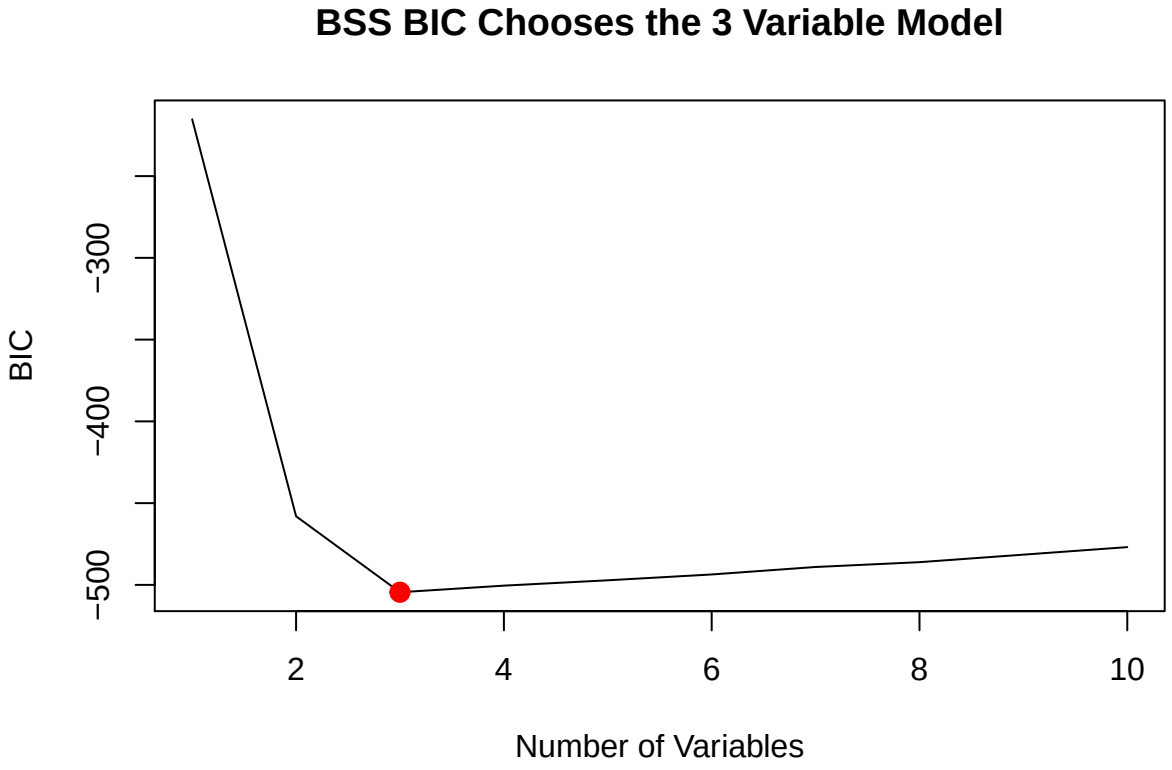
- **C**. Using Best Subset Selection below, C_p, *BIC and adjusted* R^2 all select the model with three variables. The coefficient estimimates come out to $Y = 1.072 - 1.583X + 3.695X^2 + 4.890X^3$

```
library(leaps)
df <- data.frame(X, Y)

# Best Subset Selection
# in poly(), setting raw = TRUE return raw polynomials, as opposed to
# orthogonal
regfit.full <- regsubsets(Y ~ poly(X, 10, raw = TRUE),
                          data = df,
                          nvmax = 10,
```

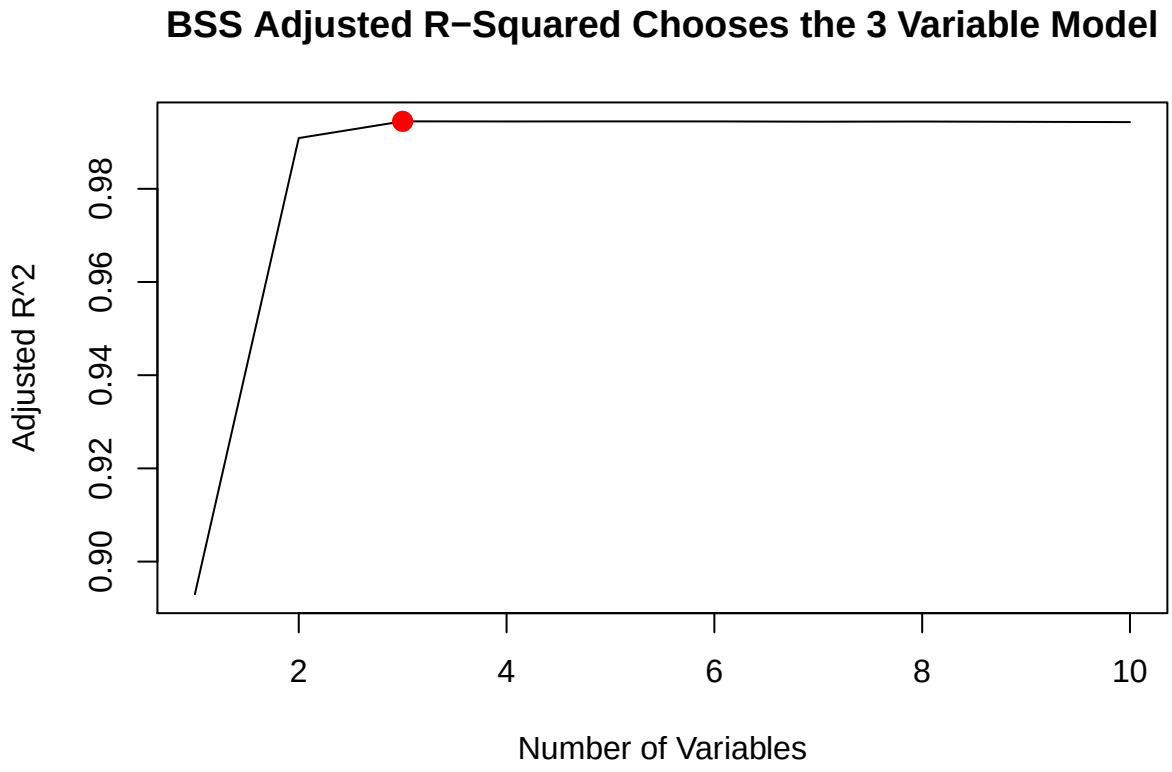
```
                          method = 'exhaustive')
reg.summary <- summary(regfit.full)

# Cp
plot(reg.summary$cp,
     xlab = 'Number of Variables',
     ylab = 'Cp',
     main = "BSS Cp Chooses the 3 Variable Model",
     type = 'l')
points(which.min(reg.summary$cp),
       reg.summary$cp[which.min(reg.summary$cp)],
       col = 'red',
       cex = 2,
       pch = 20)
```




```
# BIC
plot(reg.summary$bic,
     xlab = 'Number of Variables',
     ylab = 'BIC',
     main = "BSS BIC Chooses the 3 Variable Model",
     type = 'l')
points(which.min(reg.summary$bic),
       reg.summary$bic[which.min(reg.summary$bic)],
       col = 'red',
       cex = 2,
       pch = 20)
```




```
# Adj R62
plot(reg.summary$adjr2,
     xlab = 'Number of Variables',
     ylab = 'Adjusted R^2',
     main = "BSS Adjusted R-Squared Chooses the 3 Variable Model",
     type = 'l')
points(which.max(reg.summary$adjr2),
      reg.summary$adjr2[which.max(reg.summary$adjr2)],
      col = 'red',
      cex = 2,
      pch = 20)
```



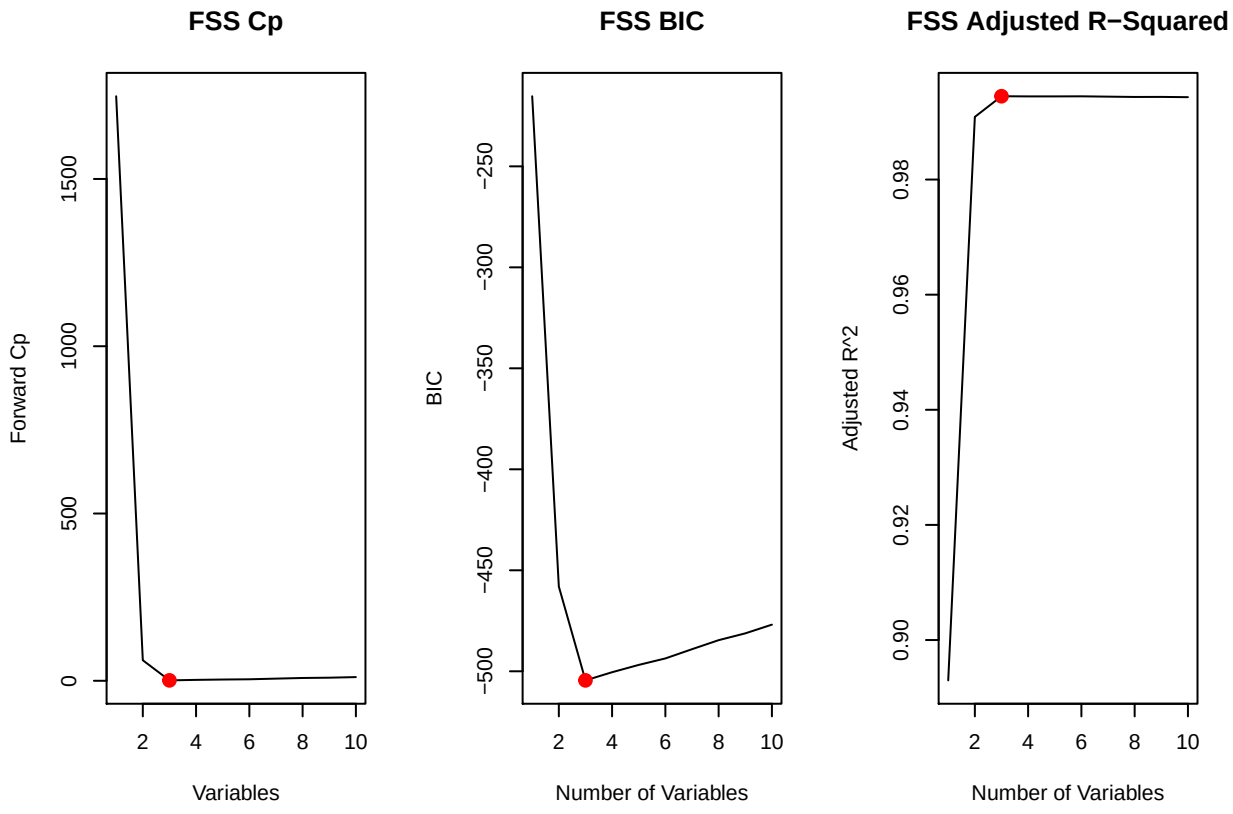

```
coef(regfit.full, id = 3)
```

```
##              (Intercept) poly(X, 10, raw = TRUE)1 poly(X, 10, raw = TRUE)2
##                 1.072028               -1.583063                3.695184
## poly(X, 10, raw = TRUE)3
##                 4.889762
```

- **D**. C_p, *BIC and adjusted* R^2 from both Forward and Backward Stepwise Selection all select the model with three variables as well.

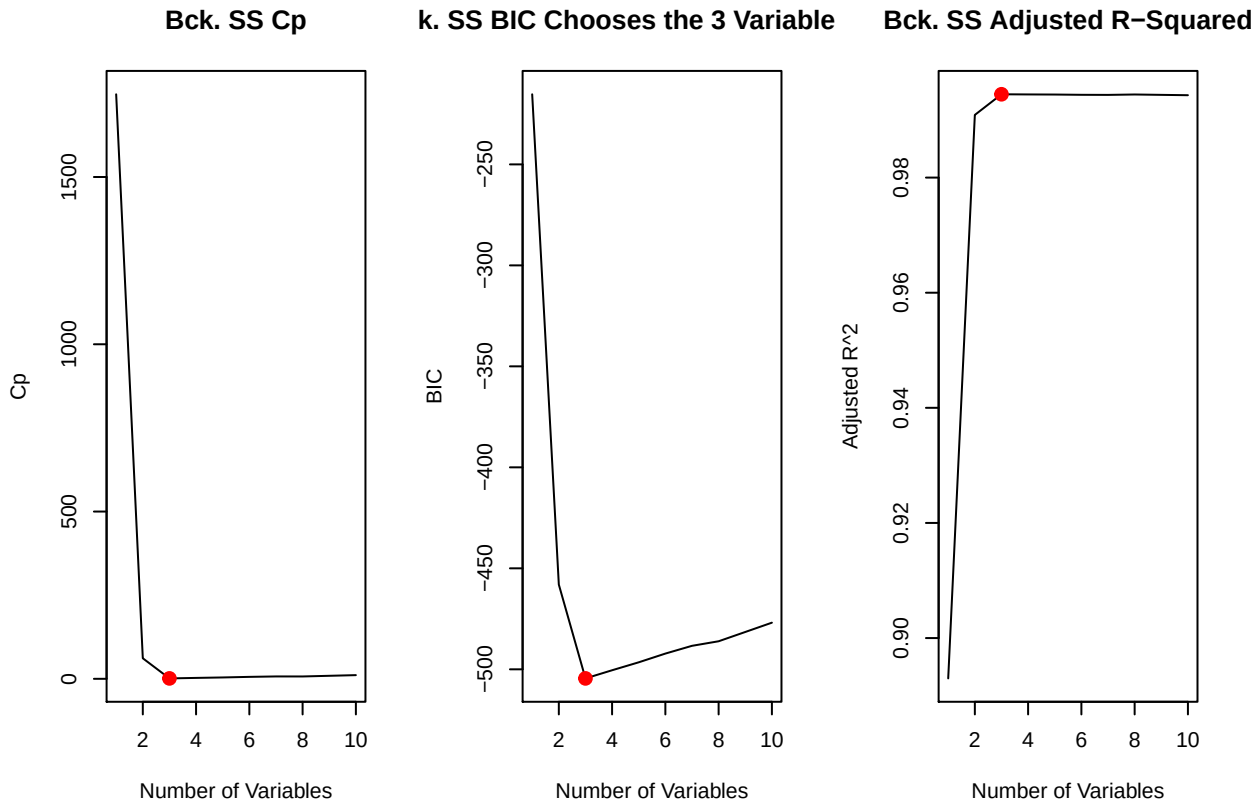
```
# Forward and Stepwise Selection
regfit.forward <- regsubsets(Y ~ poly(X, 10, raw = TRUE),
                             data = df,
                             nvmax = 10,
                             method = 'forward')
reg.summary.forward <- summary(regfit.forward)

par(mfrow = c(1,3))

# Cp
plot(reg.summary.forward$cp,
     xlab = 'Variables',
     ylab = 'Forward Cp',
     main = "FSS Cp",
     type = 'l')
points(which.min(reg.summary.forward$cp),
       reg.summary.forward$cp[which.min(reg.summary.forward$cp)],
       col = 'red',
       cex = 2,
       pch = 20)
```

```r
# BIC
plot(reg.summary.forward$bic,
     xlab = 'Number of Variables',
     ylab = 'BIC',
     main = "FSS BIC",
     type = 'l')
points(which.min(reg.summary.forward$bic),
       reg.summary.forward$bic[which.min(reg.summary.forward$bic)],
       col = 'red',
       cex = 2,
       pch = 20)

# Adj R62
plot(reg.summary.forward$adjr2,
     xlab = 'Number of Variables',
     ylab = 'Adjusted R^2',
     main = "FSS Adjusted R-Squared",
     type = 'l')
points(which.max(reg.summary.forward$adjr2),
      reg.summary.forward$adjr2[which.max(reg.summary.forward$adjr2)],
      col = 'red',
      cex = 2,
      pch = 20)
```




```r
# Backward Stepwise Selection
regfit.backward <- regsubsets(Y ~ poly(X, 10, raw = TRUE),
                              data = df,
                              nvmax = 10,
```

```
                            method = 'backward')
reg.summary.backward <- summary(regfit.backward)

par(mfrow = c(1,3))

# Cp
plot(reg.summary.backward$cp,
     xlab = 'Number of Variables',
     ylab = 'Cp',
     main = "Bck. SS Cp",
     type = 'l')
points(which.min(reg.summary.backward$cp),
       reg.summary.backward$cp[which.min(reg.summary.backward$cp)],
       col = 'red',
       cex = 2,
       pch = 20)

# BIC
plot(reg.summary.backward$bic,
     xlab = 'Number of Variables',
     ylab = 'BIC',
     main = "Bck. SS BIC Chooses the 3 Variable Model",
     type = 'l')
points(which.min(reg.summary.backward$bic),
       reg.summary.backward$bic[which.min(reg.summary.backward$bic)],
       col = 'red',
       cex = 2,
       pch = 20)

# Adj R62
plot(reg.summary.backward$adjr2,
     xlab = 'Number of Variables',
     ylab = 'Adjusted R^2',
     main = "Bck. SS Adjusted R-Squared",
     type = 'l')
points(which.max(reg.summary.backward$adjr2),
      reg.summary.backward$adjr2[which.max(reg.summary.backward$adjr2)],
      col = 'red',
      cex = 2,
      pch = 20)
```



- **E**. Using the λ value selected using cross validation, Lasso Regression selects X^2, X^3 *and* X^5 with coefficients of 3.5, 3.6 and 0.2 respectively (along with an intercept of 1.31). This is different from the true coefficients of $\beta_0 = 1,\ \beta_1(X) = -2,\ \beta_2(X^2) = 3.75,\ \beta_3(X^3) = 5$.

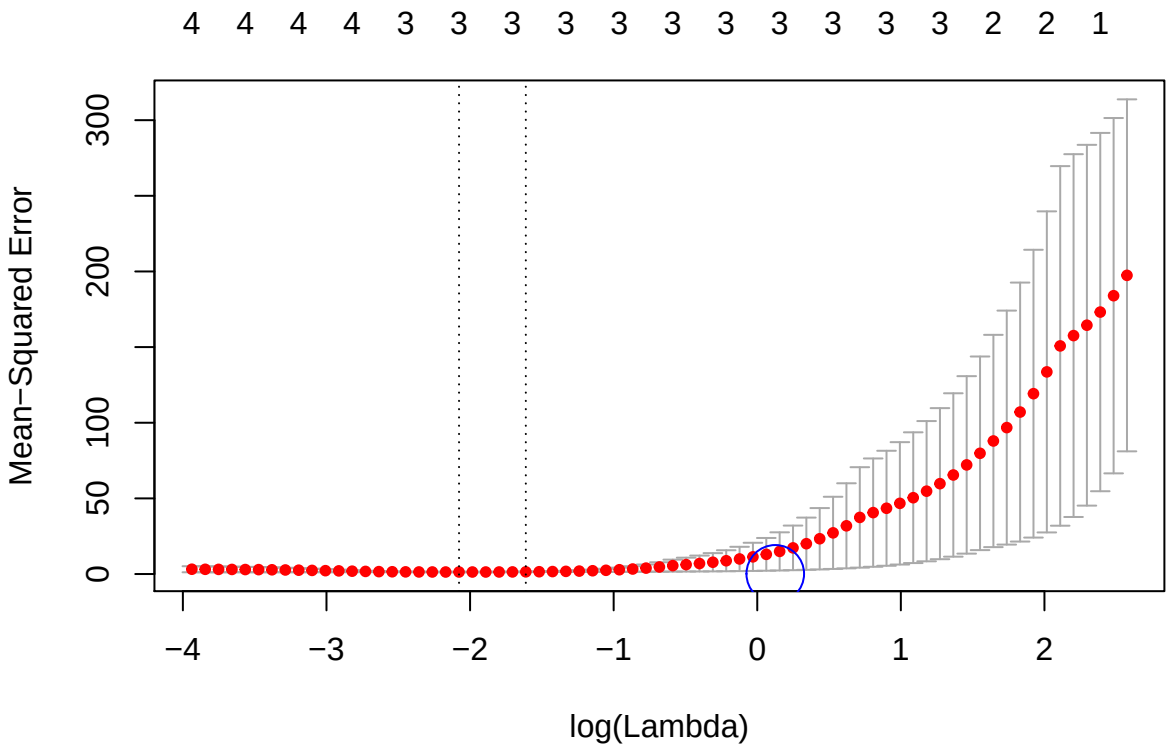
```
# Lasso Regression
set.seed(5)
library(glmnet)
```

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

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
model.x <- model.matrix(Y ~ poly(X, 10, raw = TRUE), data = df)[, -1]

par(mfrow = c(1,1))

set.seed(1)
cv.out <- cv.glmnet(model.x, Y, alpha = 1)
plot(cv.out)
best_lam <- cv.out$lambda.min
points(best_lam,
       best_lam,
       col = 'blue',
       cex = 4)
```




```
best_lam
```

```
## [1] 0.1253245
```

```
lasso.mod <- glmnet(model.x, Y, alpha = 1, lambda = best_lam)
coef(lasso.mod)
```

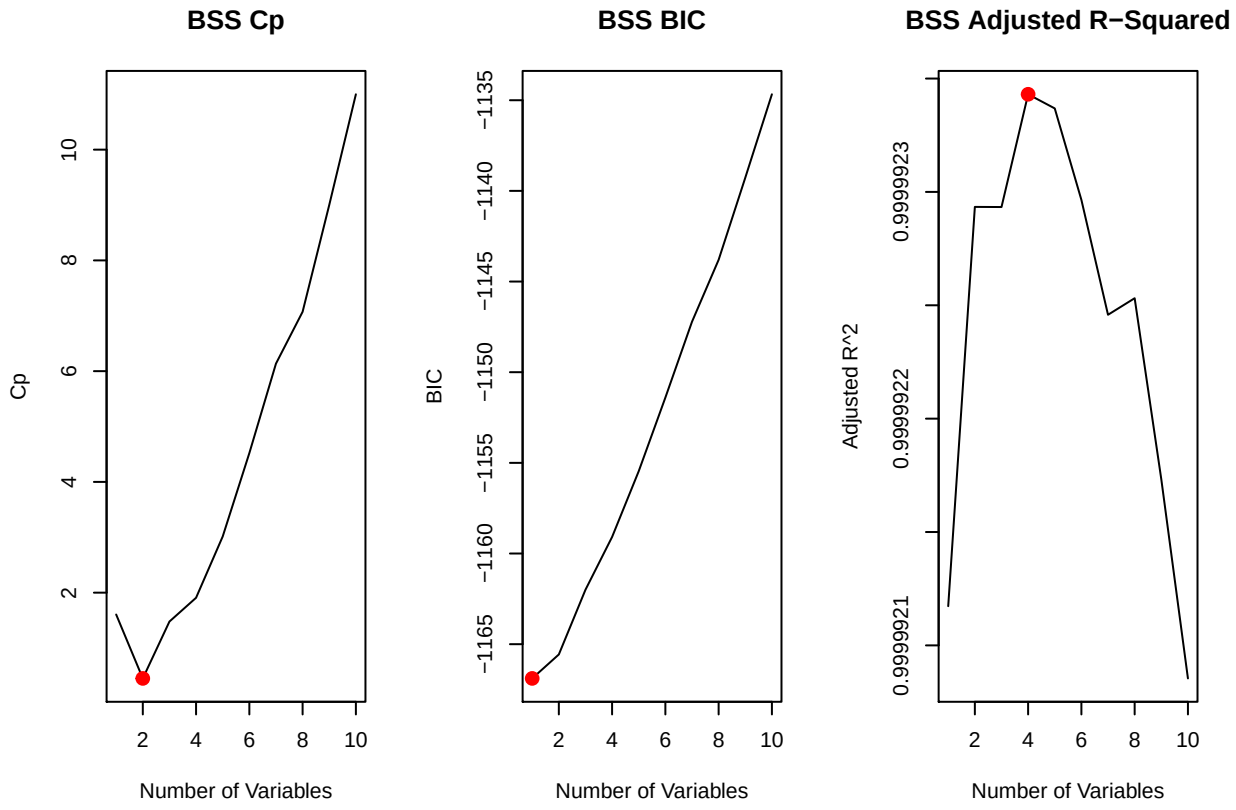
```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                      s0
## (Intercept)                   1.3135583
## poly(X, 10, raw = TRUE)1      .
## poly(X, 10, raw = TRUE)2      3.5019559
## poly(X, 10, raw = TRUE)3      3.5942028
## poly(X, 10, raw = TRUE)4      .
## poly(X, 10, raw = TRUE)5      0.1930986
## poly(X, 10, raw = TRUE)6      .
## poly(X, 10, raw = TRUE)7      .
## poly(X, 10, raw = TRUE)8      .
## poly(X, 10, raw = TRUE)9      .
## poly(X, 10, raw = TRUE)10     .
```

- **F**. Setting a new response variable such that $Y = 5 + 6X^7 + \epsilon$, C_p, *BIC and Adjusted* R^2 suggest the 2, 1 and 4 variable model respectively. *BIC* did correctly identify X^7, as well as correctly identify the coefficients as 5.02 and 6 for β_0 *and* β_1 respectively.

The Lasso also selected the correct coefficients, β_0 *and* β_7, as well as chose coefficients that are relatively close to the true values, although not as close as those obtained using Best Subset Selection. **This goes to show that when the response Y is a function of multiple predictors, the Lasso will produced a biased estimate of the model. However, when the response is a function of only a couple coefficients (in this case 1), the Lasso will approximate the true relationship relatively well.**

```
set.seed(5)
X = rnorm(100)
```

```r
noise <- rnorm(100)
beta_0 <- 5
beta_7 <- 6

Y <- beta_0 + beta_7*X^7 + noise

df <- data.frame(X, Y)

# Best Subset Selection
regfit.full <- regsubsets(Y ~ poly(X, 10, raw = TRUE),
                          data = df,
                          nvmax = 10,
                          method = 'exhaustive')
reg.summary <- summary(regfit.full)

par(mfrow = c(1,3))

# Cp
plot(reg.summary$cp,
     xlab = 'Number of Variables',
     ylab = 'Cp',
     main = "BSS Cp",
     type = 'l')
points(which.min(reg.summary$cp),
       reg.summary$cp[which.min(reg.summary$cp)],
       col = 'red',
       cex = 2,
       pch = 20)

# BIC
plot(reg.summary$bic,
     xlab = 'Number of Variables',
     ylab = 'BIC',
     main = "BSS BIC",
     type = 'l')
points(which.min(reg.summary$bic),
       reg.summary$bic[which.min(reg.summary$bic)],
       col = 'red',
       cex = 2,
       pch = 20)

# Adj R62
plot(reg.summary$adjr2,
     xlab = 'Number of Variables',
     ylab = 'Adjusted R^2',
     main = "BSS Adjusted R-Squared",
     type = 'l')
points(which.max(reg.summary$adjr2),
      reg.summary$adjr2[which.max(reg.summary$adjr2)],
      col = 'red',
      cex = 2,
      pch = 20)
```




```
coef(regfit.full, id = 1)
```

```
##              (Intercept) poly(X, 10, raw = TRUE)7
##                 5.018642                 5.999258
```

```
# Lasso Regression
set.seed(5)
model.x <- model.matrix(Y ~ poly(X, 10, raw = TRUE), data = df)[, -1]

lasso.mod <- glmnet(model.x, Y, alpha = 1)
predict(lasso.mod, s = best_lam, type = 'coefficients')
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                1
## (Intercept)                5.521197
## poly(X, 10, raw = TRUE)1   .
## poly(X, 10, raw = TRUE)2   .
## poly(X, 10, raw = TRUE)3   .
## poly(X, 10, raw = TRUE)4   .
## poly(X, 10, raw = TRUE)5   .
## poly(X, 10, raw = TRUE)6   .
## poly(X, 10, raw = TRUE)7   5.824377
## poly(X, 10, raw = TRUE)8   .
## poly(X, 10, raw = TRUE)9   .
## poly(X, 10, raw = TRUE)10  .
```

9

- **A**. *Train/Test Split*

```
library(ISLR)
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
```

```
## The following object is masked from 'package:glmnet':
##
##     auc
```

```
set.seed(5)
train_idx <- sample(c(TRUE, FALSE), nrow(College), rep = TRUE)
test_idx <- (!train_idx)
```

- **B**. Ordinary Least Squares returns a testing RMSE of **998.93**.

```
lm.mod <- lm(Apps ~ .,
         data = College,
         subset = train_idx)
lm.pred <- predict(lm.mod, newdata = College[test_idx,])
lm.test.mse <- mse(College[test_idx, 'Apps'], lm.pred)
sqrt(lm.test.mse)
```

```
## [1] 1143.314
```

- **C**. Ridge Regression produces a slightly higher testing RMSE than OLS, coming in at **1003**.

```
x = model.matrix(Apps ~ ., data = College)
y = College$Apps

lambda.grid <- 10^seq(10, -2, length = 100)

cv.out <- cv.glmnet(x[train_idx,],
                    y[train_idx],
                    alpha = 0,
                    lambda = lambda.grid)

best.lambda <- cv.out$lambda.min
ridge.mod <- glmnet(x, y, alpha = 0, lambda = best.lambda)
ridge.pred <- predict(ridge.mod,
                      s = best.lambda,
                      newx = x[test_idx,])
ridge.test.mse <- mse(y[test_idx], ridge.pred)
sqrt(ridge.test.mse)
```

```
## [1] 1002.997
```

- **D**. Similar to Ridge Regression, the Lasso has a slightly higher testing RMSE than the OLS, at 993.41. The coefficients are shown below.

```
cv.out <- cv.glmnet(x[train_idx,],
                    y[train_idx],
                    alpha = 1,
                    lambda = lambda.grid)
```

```
best.lambda <- cv.out$lambda.min
lasso.mod <- glmnet(x, y, alpha = 0, lambda = best.lambda)
lasso.pred <- predict(lasso.mod,
                      s = best.lambda,
                      newx = x[test_idx,])
lasso.test.mse <- mse(y[test_idx], lasso.pred)
sqrt(lasso.test.mse)
```

```
## [1] 993.4065
```

```
predict(lasso.mod, s = best.lambda, type = 'coefficients')
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                           1
## (Intercept) -565.44436277
## (Intercept)    .
## PrivateYes  -501.14562863
## Accept         1.52059509
## Enroll        -0.64842120
## Top10perc     46.78469349
## Top25perc    -12.20864720
## F.Undergrad    0.04354884
## P.Undergrad    0.04275676
## Outstate      -0.07954578
## Room.Board     0.15882475
## Books          0.02993927
## Personal       0.02665051
## PhD           -8.24595036
## Terminal      -3.63832744
## S.F.Ratio     15.43376088
## perc.alumni   -0.96459601
## Expend         0.07843444
## Grad.Rate      8.91838396
```

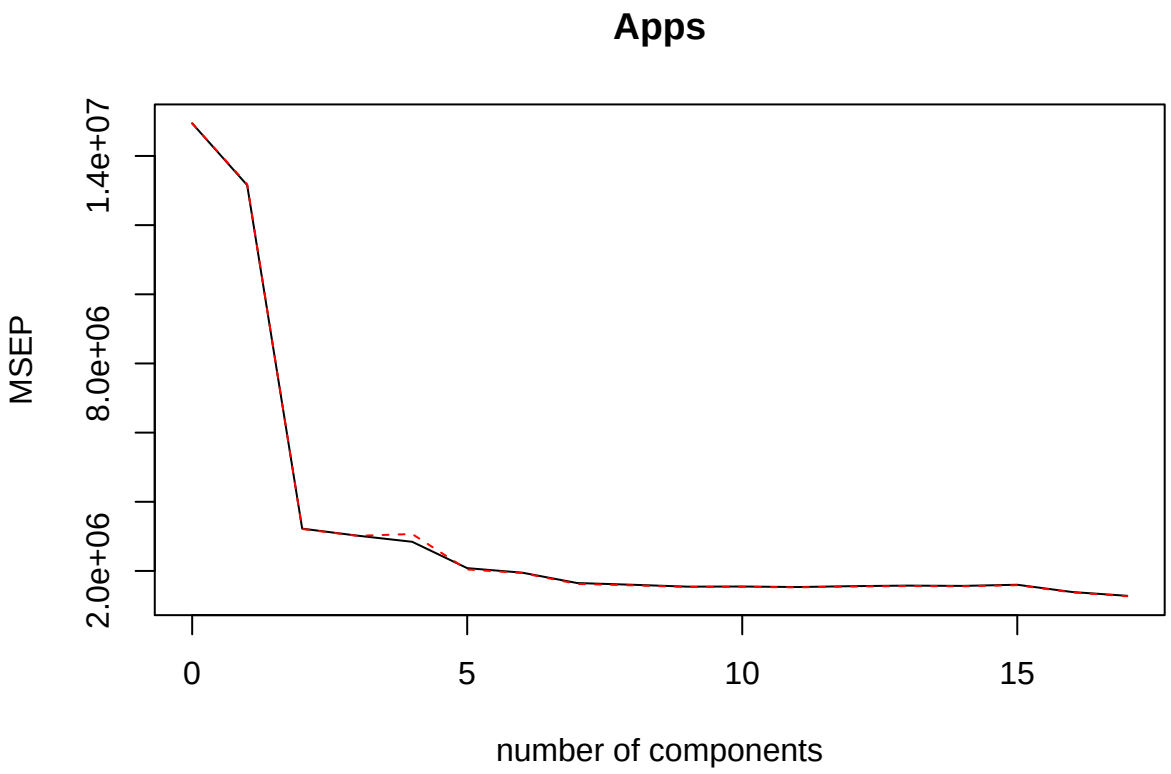
- **E**. Using 5 principal components, PCR returns a testing RMSE of 1885.01, double the error of OLS.

```
library(pls)
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
set.seed(5)
pcr.fit <- pcr(Apps ~ .,
               data = College[train_idx,],
               scale = TRUE,
               validation = 'CV')
validationplot(pcr.fit, val.type = 'MSEP')
```

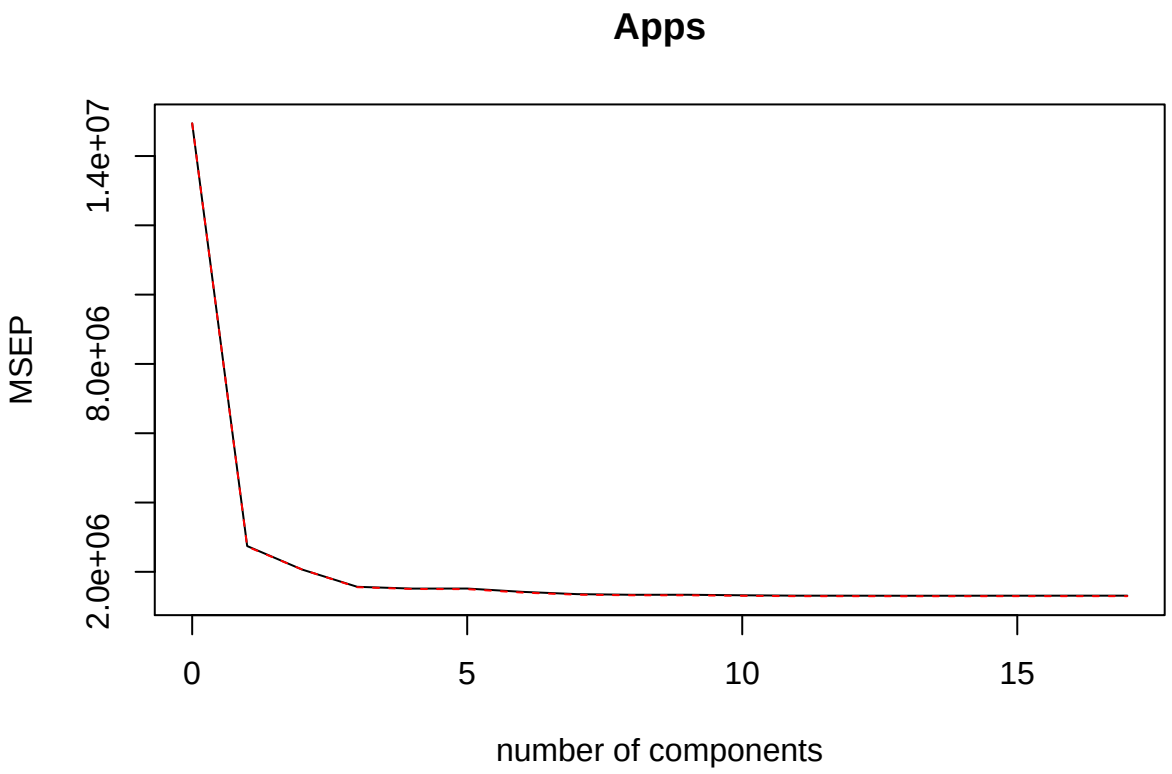



```
pcr.pred <- predict(pcr.fit, College[test_idx, ], ncomp = 5)
pcr.test.mse <- mse(College[test_idx, 'Apps'], pcr.pred)
sqrt(pcr.test.mse)
```

```
## [1] 1885.047
```

- **F**. Partial Least Squares with 3 components returns a testing RMSE of 1665.24, also well above the RMSE of OLS.

```
pls.fit <- plsr(Apps ~ .,
                data = College[train_idx,],
                scale = TRUE,
                validation = 'CV')
validationplot(pls.fit, val.type = 'MSEP')
```

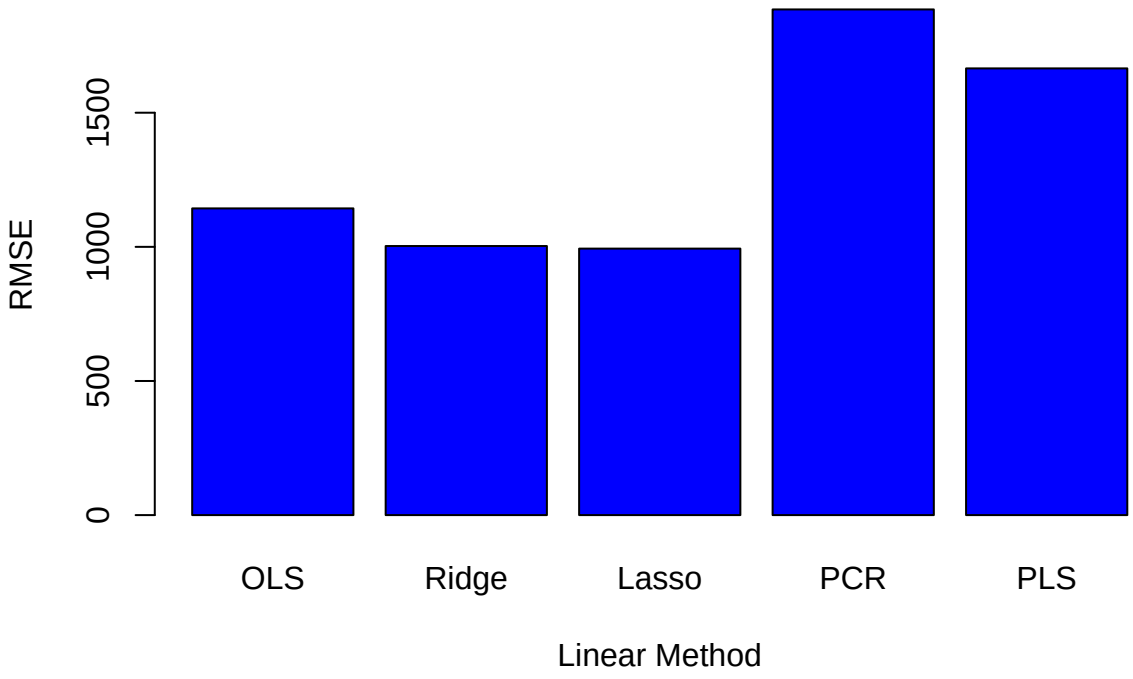



```
pls.pred <- predict(pls.fit, College[test_idx,], ncomp = 3)
pls.test.mse <- mse(College[test_idx, 'Apps'], pls.pred)
sqrt(pls.test.mse)
```

```
## [1] 1665.244
```

- **G**. Plotting the testing RMSE's of the five methods tested, it is clear that OLS, Ridge and Lasso are far superior to Principal Component Regression and Partial Least Squares. Since OLS produced a testing RMSE of just under 1000, this means that, *on average*, the prediction for the number of applications will be off by 983 using OLS.

```
errors <- c(sqrt(lm.test.mse),
  sqrt(ridge.test.mse),
  sqrt(lasso.test.mse),
  sqrt(pcr.test.mse),
  sqrt(pls.test.mse))
barplot(errors,
        col = 'blue',
        names = c('OLS','Ridge','Lasso','PCR','PLS'),
        ylab = 'RMSE',
        xlab = 'Linear Method',
        main = 'Principal Component Methods Fall Behind')
```



10

- **A**. Creating 20 attributes randomly generated from a Normal Distribution and a response variable that is a function of only some of the attributes.

```
set.seed(5)
X <- matrix(rnorm(1000* 20), nrow = 1000, ncol = 20)
betas <- sample(seq(-50, 50, 0.25), 20)
noise <- rnorm(20)
for (i in 1:20) {
    if (i %% 2 == 0) {
        betas[i] <- 0
    }
}
y <- X %*% betas + noise
df <- data.frame(x = X, y = y)
```

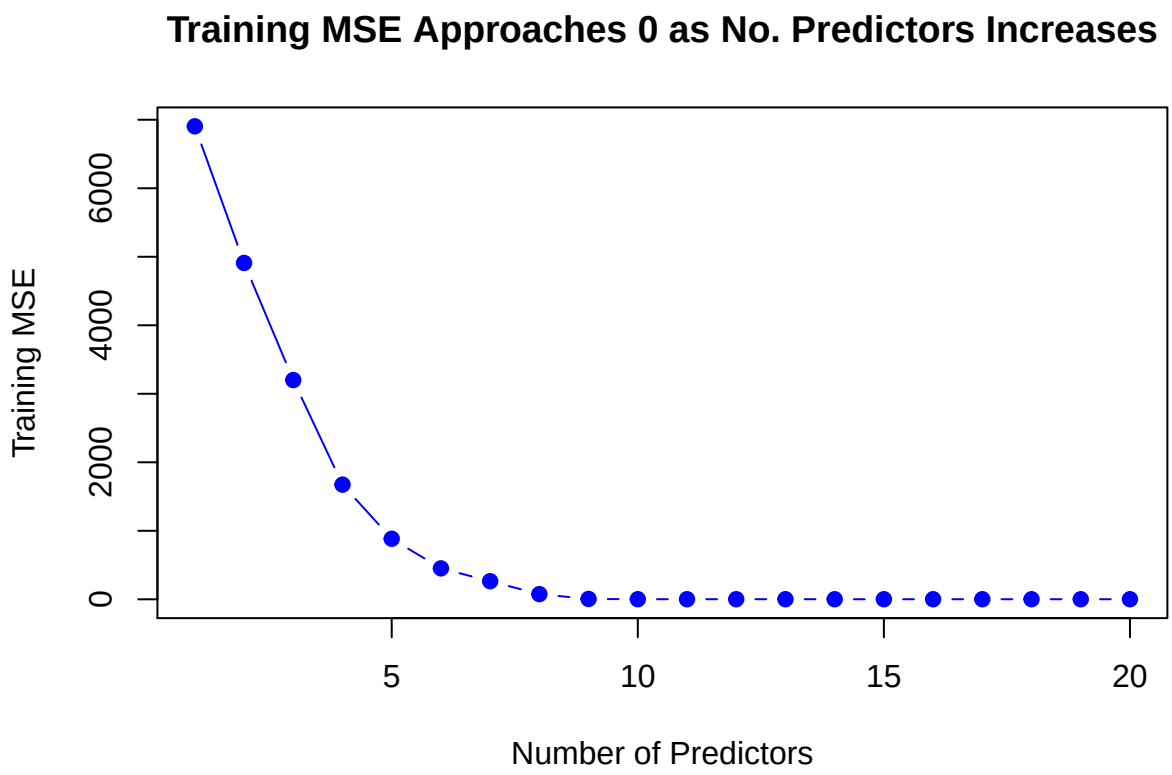
- **B**.

```
train_idx <- sample(1:nrow(df), 100)
```

- **C**. As one would expect, the training error approaches zero.

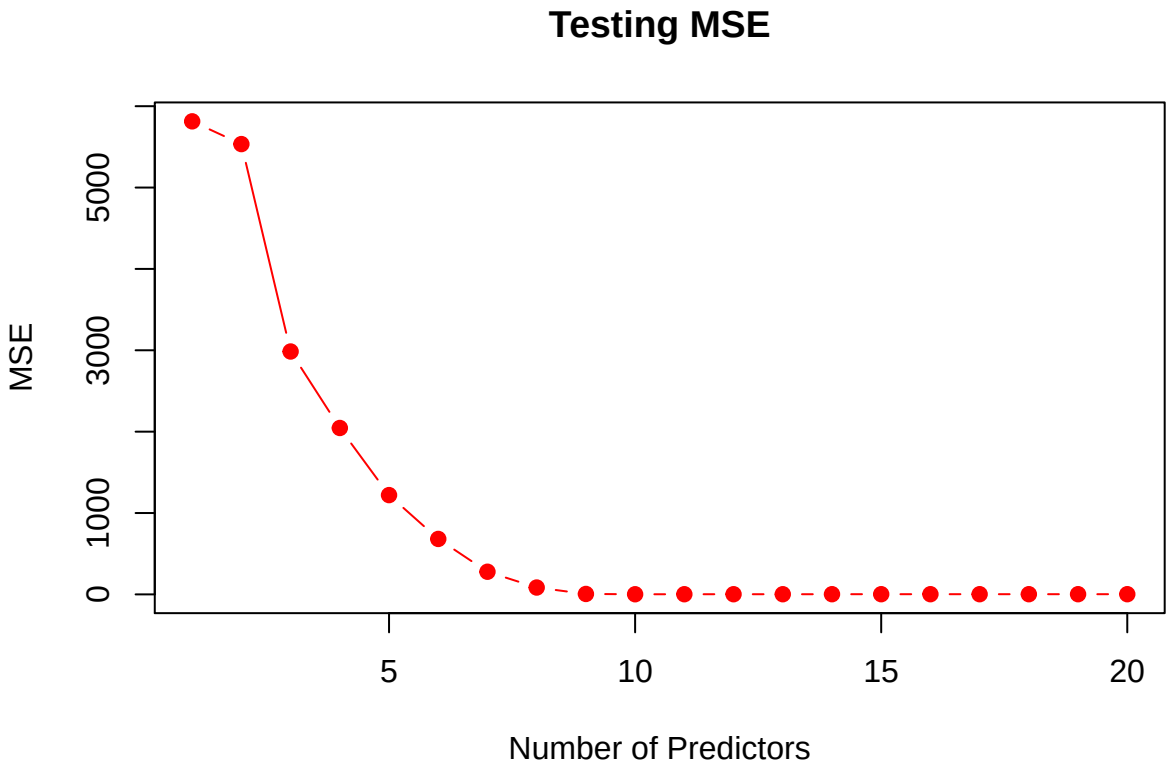
```
best.subset <- regsubsets(y ~ .,
                          data = df,
                          subset = train_idx,
                          nvmax = 20,
                          method = 'exhaustive')
best.subset.summary <- summary(best.subset)
par(mfrow = c(1,1))
```

```r
plot(best.subset.summary$rss / 100,
     pch = 19,
     col = 'blue',
     type = 'b',
     xlab = 'Number of Predictors',
     ylab = 'Training MSE',
     main = 'Training MSE Approaches 0 as No. Predictors Increases')
```



- **D**.

```r
df.test.matrix <- model.matrix(y ~ ., data = df[-train_idx,])
test.errors <- NULL
for (i in 1:20) {
    coefs <- coef(best.subset, id = i)
    predictions <- df.test.matrix[, names(coefs)] %*% coefs
    test.errors[i] <- mse(y[-train_idx], predictions)
}
plot(test.errors,
     pch = 19,
     col = 'red',
     type = 'b',
     xlab = 'Number of Predictors',
     ylab = 'MSE',
     main = 'Testing MSE')
```



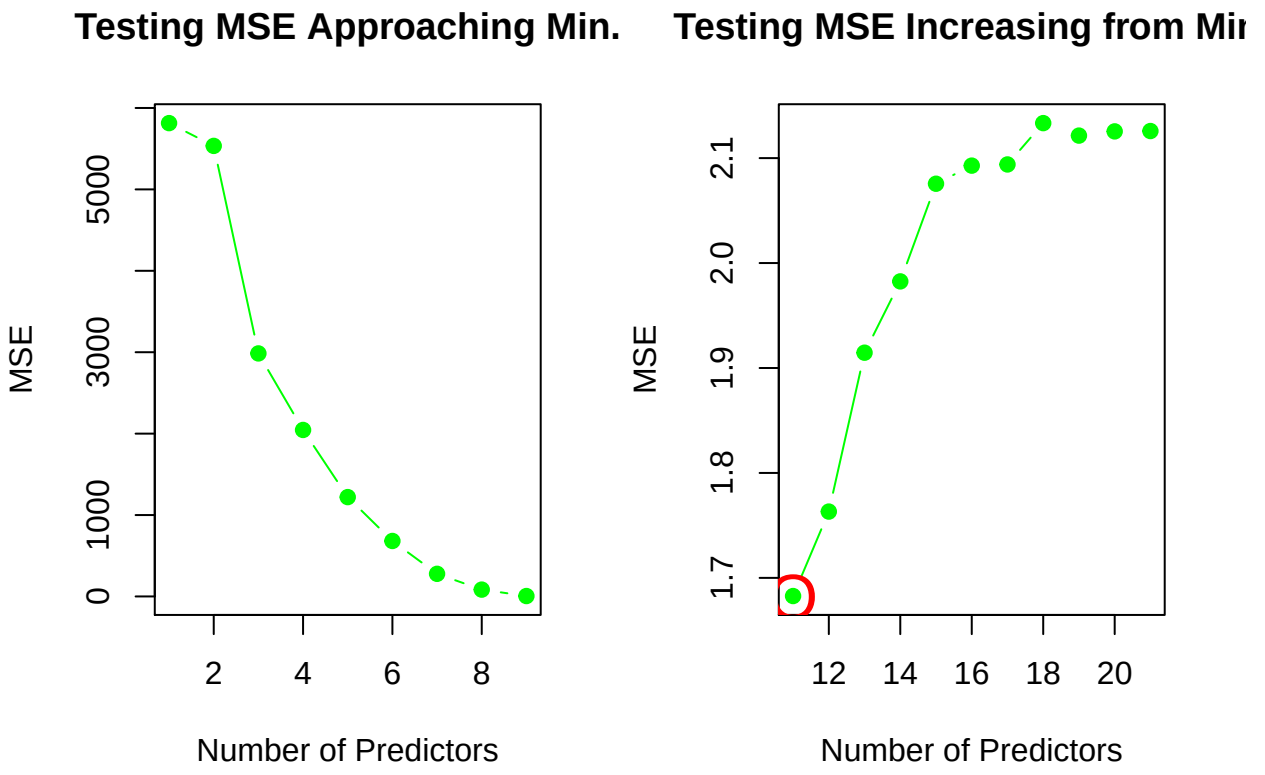
- **E**. As one would expect, the testing MSE is minimized at a model with an intermediate amount of predictors. This resembles the classic "U" shape that the testing error will take when moving from a simple model to a more complex model (**overfitting**). (This is imperceptible to the eye in the above plot, however if I plot models 1 through 9 on one graph and 10 through 20 on another it becomes clear; these plots are shown below)

```
which.min(test.errors)
```

```
## [1] 10
```

```
par(mfrow = c(1, 2))
plot(test.errors[1:9],
     pch = 19,
     col = 'green',
     type = 'b',
     xlab = 'Number of Predictors',
     ylab = 'MSE',
     main = 'Testing MSE Approaching Min.')
plot(test.errors[10:20],
     pch = 19,
     col = 'green',
     type = 'b',
     xlab = 'Number of Predictors',
     ylab = 'MSE',
     main = 'Testing MSE Increasing from Min.',
     xaxt = 'n')
axis(1, at = seq(0, 10, 2), labels = c('10','12','14','16','18','20'))
points(which.min(test.errors) - 9,
       test.errors[which.min(test.errors)],
       col = 'red',
       pch = 'O',
```

```
      cex = 2)
```



- **F**. Illustrated in the table below, the coefficient estimates are very close to the true values (note that the coefficients excluded from the table were correctly set to 0 in the model).

```
estimates <- rep(0, 20)
estimates[1] <- coef(best.subset, id = 10)[2]
estimates[3] <- coef(best.subset, id = 10)[3]
estimates[5] <- coef(best.subset, id = 10)[4]
estimates[7] <- coef(best.subset, id = 10)[5]
estimates[9] <- coef(best.subset, id = 10)[6]
estimates[11] <- coef(best.subset, id = 10)[7]
estimates[13] <- coef(best.subset, id = 10)[8]
estimates[15] <- coef(best.subset, id = 10)[9]
estimates[17] <- coef(best.subset, id = 10)[10]
estimates[19] <- coef(best.subset, id = 10)[11]
coef_df <- data.frame(betas,estimates)
coef_df <- t(coef_df)[, c(1,3,5,7,9,11,13,15,17,19)]
rownames(coef_df) <- c("True_value", "Estimate")
colnames(coef_df) <- c('X1','X3','X5','X7','X9','X11','X13','X15','X17','X19')
knitr::kable(coef_df, digits = 2, caption = 'Coefficients')
```

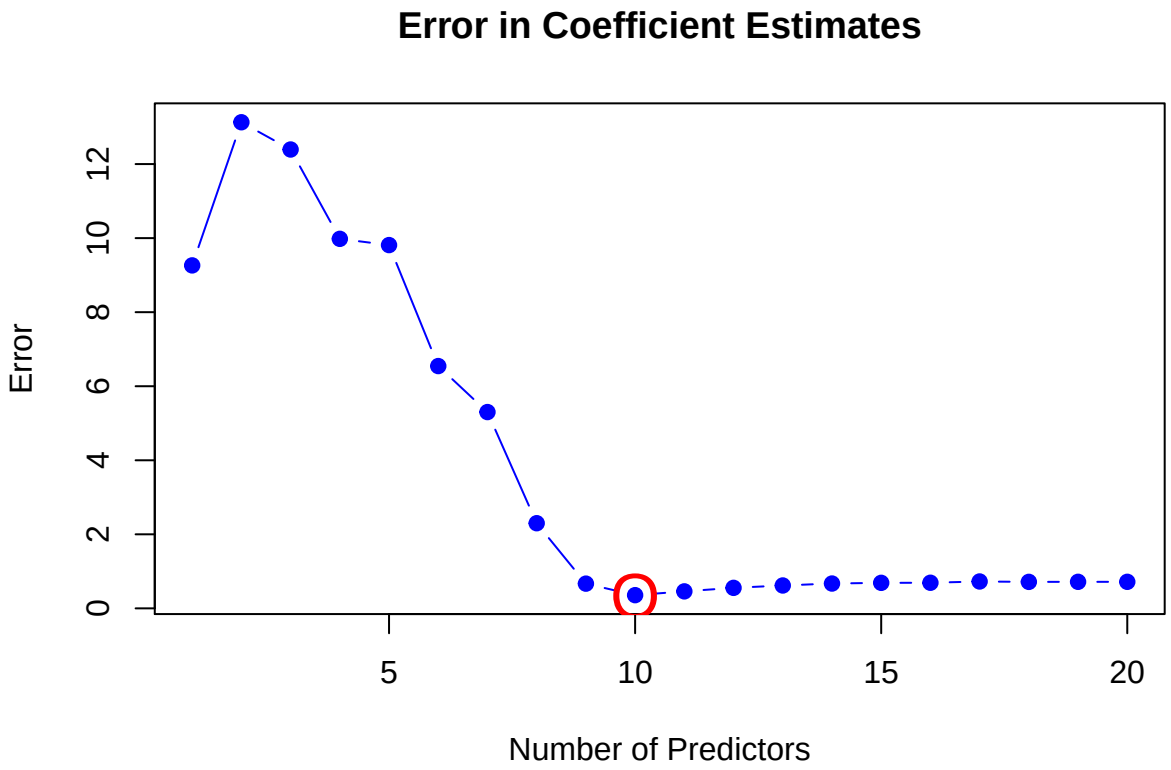
Table 1: Coefficients

	X1	X3	X5	X7	X9	X11	X13	X15	X17	X19
True_value	-8.25	-33.25	34.50	-22.50	-44.25	43.50	1.75	-19.25	12.50	-27.50
Estimate	-8.32	-33.27	34.52	-22.63	-44.17	43.57	1.86	-18.97	12.51	-27.54

- **G**. Plotting the error in the coefficient estimates as $\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_k^2)^2}$ on the y axis and the number of

predictors in the model as that number increases on the x axis, it is clear that this error is minimized at 10, the same value at which the testing MSE was minimized.

```
par(mfrow = c(1,1))
r <- seq(1, 20)
y <- rep(NA, 20)
col_names <- colnames(df)[1:20]
names(betas) <- col_names
for (i in 1:20) {
    cfs <- coef(best.subset, id = i)
    cfs <- cfs[2:length(cfs)]
    y[i] <- sqrt(sum((betas[names(cfs)] - cfs)^2))
}
plot(x = r,
     y = y,
     type = 'b',
     col = 'blue',
     pch = 19,
     xlab = 'Number of Predictors',
     ylab = 'Error',
     main = 'Error in Coefficient Estimates')
points(which.min(y),
       y[which.min(y)],
       col = 'red',
       pch = 'O',
       cex = 2)
```



11

- **A**. Comparing Best Subset Selection, Lasso, Ridge and Principal Components Regression, Best Subset Selection comes out slightly ahead of the Ridge and Lasso, all three of which have a MSE in the 47 neighborhood. PCR is slightly behind the other three, with a MSE of 52.

```
library(MASS)

set.seed(5)
train_idx <- sample(1:nrow(Boston), 400)

y.test <- Boston[-train_idx, 'crim']
y.train <- Boston[train_idx, 'crim']
x.train <- model.matrix(crim ~ . - crim, data = Boston[train_idx,])
x.test <- model.matrix(crim ~ . - crim, data = Boston[-train_idx,])

# BSS
best.subset <- regsubsets(crim ~ .,
                          data = Boston[train_idx,],
                          nvmax = 14)
best.subset.test.mse <- NULL
for (i in 1:13) {
    coefs <- coef(best.subset, id = i)
    predictions <- x.test[, names(coefs)] %*% coefs
    best.subset.test.mse[i] <- mse(y.test, predictions)
}

# Lasso
cv.out <- cv.glmnet(x = x.train, y = y.train, alpha = 1)
best.lambda <- cv.out$lambda.min
lasso.mod <- glmnet(x = x.train, y = y.train, alpha = 1, lambda = best.lambda)
lasso.pred <- predict(lasso.mod,
                      s = best.lambda,
                      newx = x.test)
lasso.test.mse <- mse(y.test, lasso.pred)

# Ridge
cv.out <- cv.glmnet(x = x.train, y = y.train, alpha = 0)
best.lambda <- cv.out$lambda.min
ridge.mod <- glmnet(x = x.train, y = y.train, alpha = 0, lambda = best.lambda)
ridge.pred <- predict(ridge.mod,
                      s = best.lambda,
                      newx = x.test)
ridge.test.mse <- mse(y.test, ridge.pred)

# PCR
pcr.mod <- pcr(crim ~ .,
               data = Boston[train_idx,],
               scale = TRUE,
               validation = 'CV')
pcr.pred <- predict(pcr.mod, Boston[-train_idx, ], ncomp = 7)
pcr.test.mse <- mse(y.test, pcr.pred)
errors <- round(c(best.subset.test.mse[which.min(best.subset.test.mse)],
          ridge.test.mse,
```

```
          lasso.test.mse,
          pcr.test.mse), 2)
best.subset.test.mse[which.min(best.subset.test.mse)]
```

```
## [1] 47.12506
```

```
ridge.test.mse
```

```
## [1] 47.96965
```

```
lasso.test.mse
```

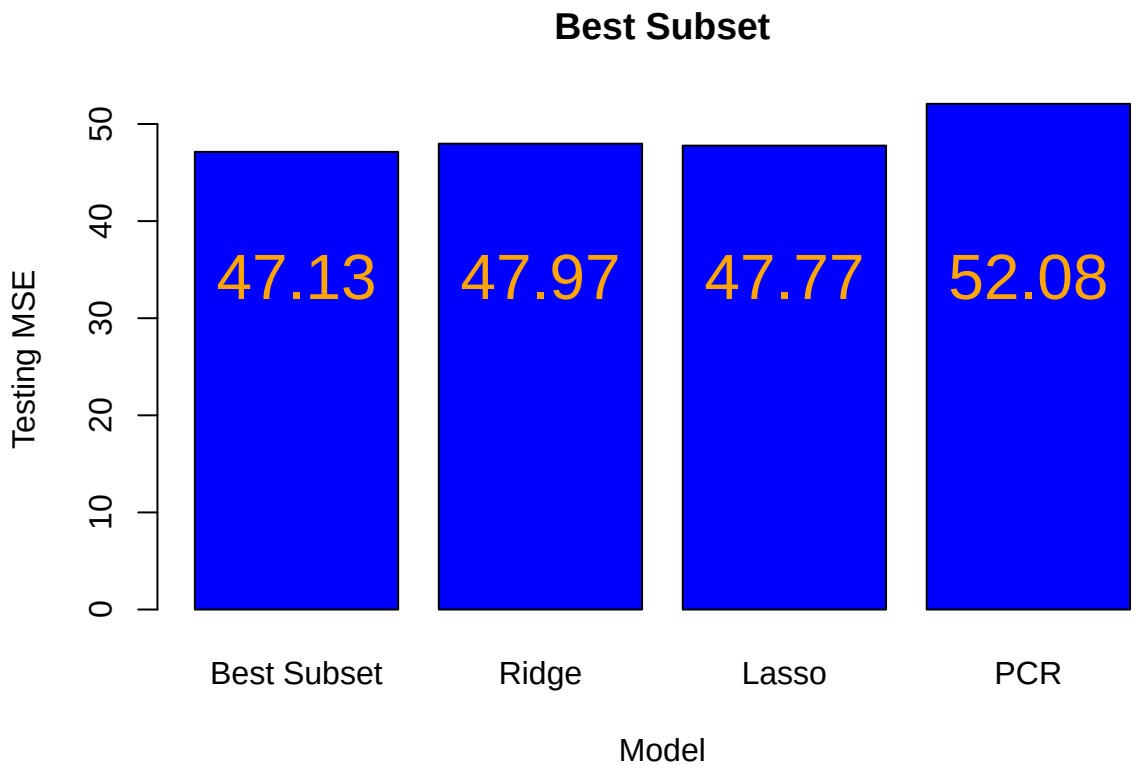
```
## [1] 47.7716
```

```
pcr.test.mse
```

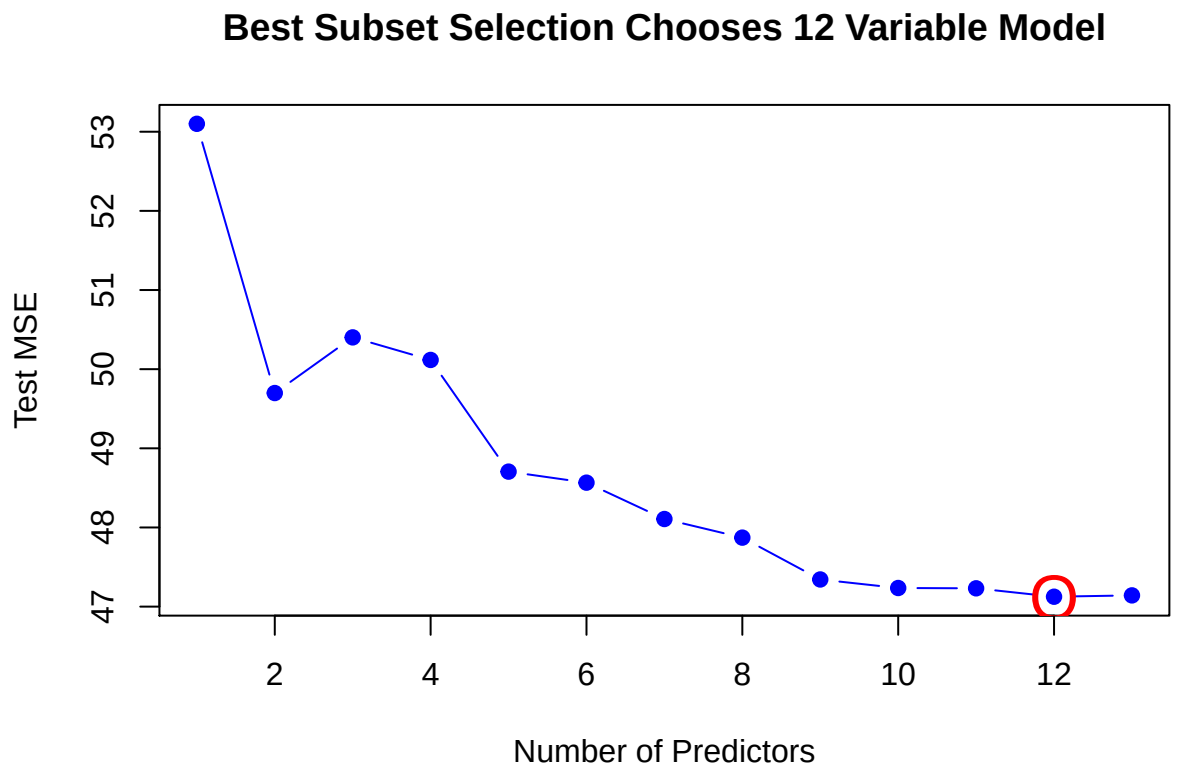
```
## [1] 52.07587
```

- **B**. The errors displayed above are plotted below. I would suggest the model produced by Lasso over the 12 variable model put forth by Best Subset Selection. The reason for this is, although the model select by Lasso has a slightly higher testing error than BSS, since the model sets the *tax* and *age* coefficients to zero it is a slightly simpler model.

```
g <- barplot(c(best.subset.test.mse[which.min(best.subset.test.mse)],
          ridge.test.mse,
          lasso.test.mse,
          pcr.test.mse),
        names = c('Best Subset','Ridge','Lasso','PCR'),
        col = 'blue',
        xlab = 'Model',
        ylab = 'Testing MSE',
        main = 'Best Subset')
text(g, y = 30, label = errors, col = 'orange', pos = 3, cex = 2)
```



```r
plot(best.subset.test.mse,
     pch = 19,
     col = 'blue',
     type = 'b',
     xlab = 'Number of Predictors',
     ylab = 'Test MSE',
     main = 'Best Subset Selection Chooses 12 Variable Model')
points(which.min(best.subset.test.mse),
       best.subset.test.mse[which.min(best.subset.test.mse)],
       col = 'red',
       pch = 'O',
       cex = 2)
```




```r
coef(lasso.mod)
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) 13.79194629
## (Intercept)  .
## zn           0.02933442
## indus       -0.04763901
## chas        -0.65785759
## nox         -6.30685994
## rm           0.08673653
## age          .
## dis         -0.74582479
## rad          0.49961832
## tax          .
## ptratio     -0.13778315
## black       -0.01036563
```

```
## lstat        0.04820130
## medv        -0.14847164
```

- **D**. As stated above, no, the model produced by Lasso omits the *tax* and *age* variables from the model.

Resources & References

[1] Bayesian Linear Regression Slides

[2] Bayesian vs. Frequentist Linear Regression