Chapter 6 Linear Model Selection and Regularization

Exercise 1

a

The model with all k = predictors obtained from the best subset selection approach is the one whose training RSS the smallest.

b

The model from best subset selection might has the smallest test RSS since it examines the most possible models.

c

i

True. At each step in the Forward stepwise selection, the previous predictor(s) are hold and then a new predictor is added.

ii

True. At each step in the Backward stepwise selection, after one predictor is removed, the remaining predictors are hold.

iii

False. There is no relationship between the predictors in the k-variable model identified by backward stepwise and the predictors in the (k + 1)-variable model identified by forward stepwise selection.

iv

False. There is no relationship between the predictors in the k-variable model identified by forward stepwise and the predictors in the (k+1)-variable model identified by backward stepwise selection.

v

False. There is no evidence nor enough information for this statement.

Exercise 2

a

The lasso, relative to least squares, is less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance. (Figure 6.8)

b

The ridge regression, relative to least squares, is less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance. (Figure 6.5)

c

The non-linear methods relative to least squares. more flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias.

Exercise 3

We can make use of figure 6.5 and 6.8 for this exercise. The s here plays the same role as

$$\frac{1}{\lambda}.$$

a

As we increase s from 0, the training RSS will steadily decrease.

b

As we increase s from 0, the test RSS decrease initially, and then eventually start increasing in a U shape.

c

As we increase s from 0, the variance will steadily increase.

d

As we increase s from 0, the squared bias will steadily decrease.

e

As we increase s from 0, the irreducible error will remain constant.

Exercise 4

We can make use of figure 6.5 and 6.8 for this exercise.

a

As we increase

$$\lambda$$

from 0, the training RSS will steadily increase.

b

As we increase

$$\lambda$$

from 0, the test RSS decrease initially, and then eventually start increasing in a U shape.

c

As we increase

$$\lambda$$

from 0, the variance will steadily decrease.

d

As we increase

$$\lambda$$

from 0, the squared bias will steadily increase.

e

As we increase

$$\lambda$$

from 0, the irreducible error will remain constant.

Exercise 5

a

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

b

Take derivatives with respect to

$$\widehat{\beta_1}$$

and set the equation to zero, we get:

$$-2x_{11}\left(y_1 - \widehat{\beta_1} x_{11} - \widehat{\beta_2} x_{12}\right) - 2x_{21}\left(y_2 - \widehat{\beta_1} x_{21} - \widehat{\beta_2} x_{22}\right) + 2\lambda\widehat{\beta_1} = 0$$

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

Similarly, Take derivatives with respect to

$$\widehat{\beta_2}$$

and set the equation to zero, we get:

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

And since

$$x_{11} = x_{12},\ x_{21} = x_{22}$$

,

$$\lambda\widehat{\beta_1} = \lambda\widehat{\beta_2}$$

Therefore,

$$\widehat{\beta_1} = \widehat{\beta_2}$$

c

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

d

Doing the same steps as in (b), we obtain

$$\frac{\widehat{\beta_1}}{|\widehat{\beta_1}|} = \frac{\widehat{\beta_2}}{|\widehat{\beta_2}|}$$

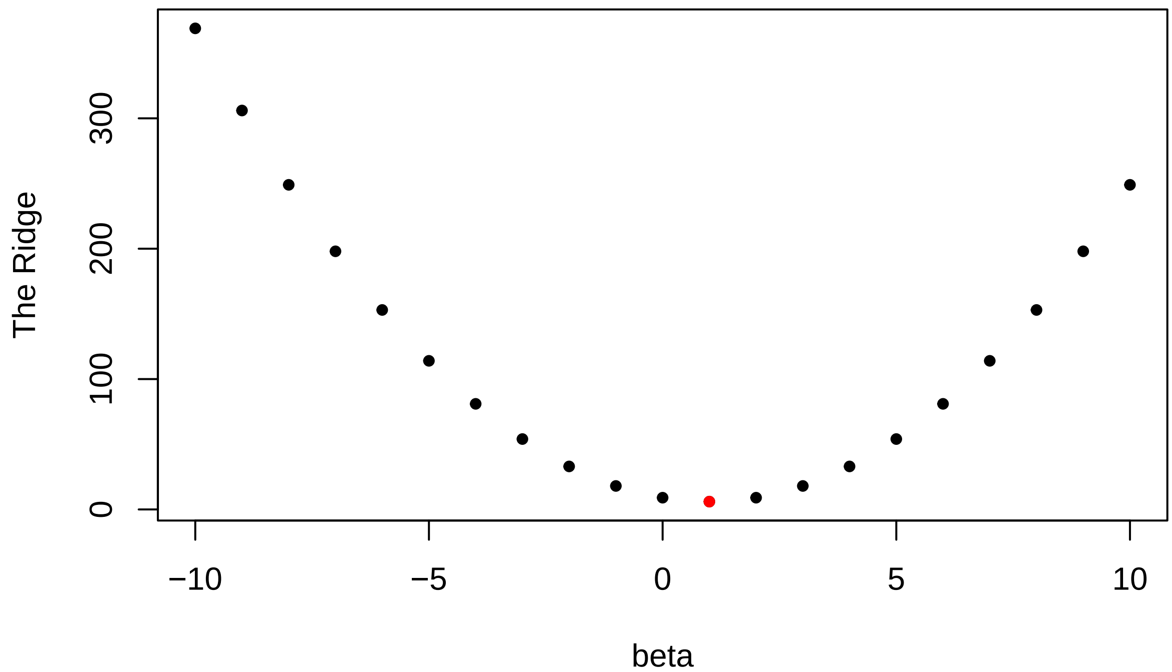
$$\Leftrightarrow \widehat{\beta_1\beta_2} > 0$$

In other words, the estimated coefficients do not have to be the same value, but they should have the same sign.

Exercise 6

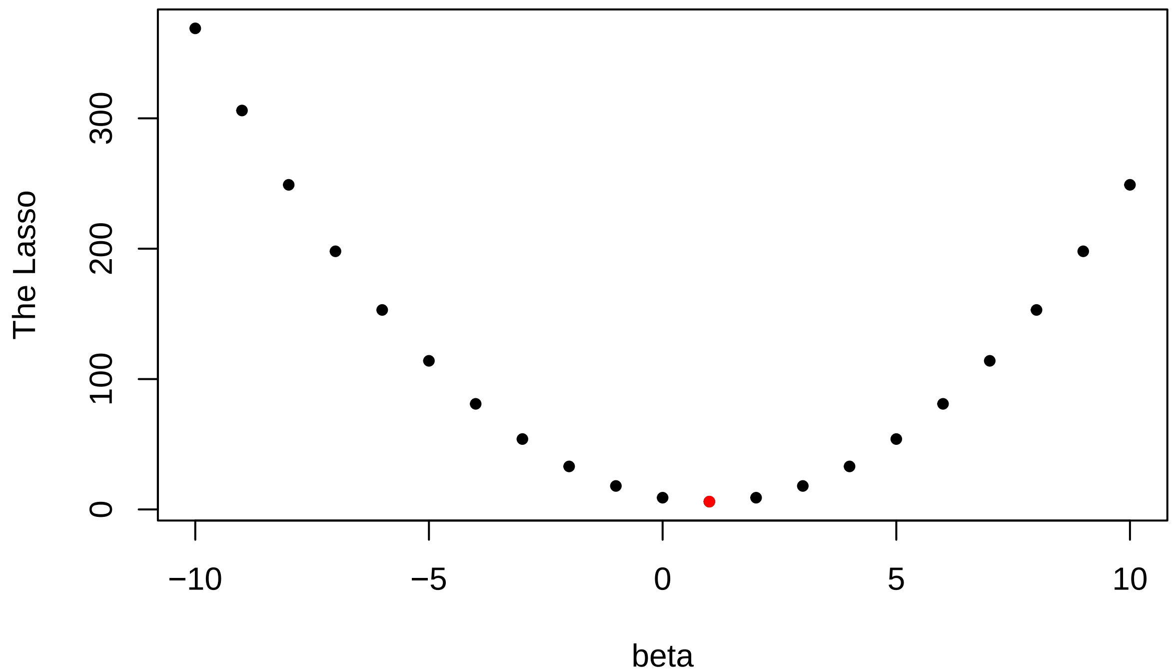
a

```
y = 3
lambda = 2
beta = seq(-10, 10, 1)
plot(beta, (y - beta)^2 + lambda * beta^2, pch = 20, xlab = "beta", ylab = "The Ridge")
estimated_beta = y / (1 + lambda)
points(estimated_beta, (y - estimated_beta)^2 + lambda * estimated_beta^2, col = "red", pch = 20)
```



b

```
y = 3
lambda = 2
beta = seq(-10, 10, 1)
plot(beta, (y - beta)^2 + lambda * beta^2, pch = 20, xlab = "beta", ylab = "The Lasso")
estimated_beta = y / (1 + lambda)
points(estimated_beta, (y - estimated_beta)^2 + lambda * abs(estimated_beta), col = "red", pch = 20)
```



Exercise 7

a

$$\mathcal{L}(Y|X,\beta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2\right)$$

b

posterior = likelihood x prior

$$\mathcal{L}(Y|X,\beta)p(\beta) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2\right)\left[\frac{1}{2b}\exp\left(-\frac{|\beta|}{b}\right)\right]$$

c

Recall from (6.7), the lasso coefficients

$$\beta$$

minimize the quantity

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

Mode value is the value the appears the most often or the probability such that value appearing is the largest. In other words, now we need to prove that the problem of maximising the posterior for

$$\beta$$

is the same as the problem of minimising the quantity above. From (b), we have

$$\mathcal{L}(Y|X,\beta)p(\beta) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2\right)\left[\frac{1}{2b}\exp\left(-\frac{|\beta|}{b}\right)\right]$$

$$log\left(\mathcal{L}(Y|X,\beta)p(\beta)\right) = \ln\left(\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n\left(\frac{1}{2b}\right)\right) - \left(\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2 + \frac{|\beta|}{b}\right)$$

And in order to maximise the posterior for

$$\beta$$

, we need to minimise the last term

$$\left(\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2 + \frac{|\beta|}{b}\right)$$

$$= \frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2 + \frac{1}{b}\sum_{j=1}^{p}|\beta_j|$$

$$= \frac{1}{2\sigma^2}\left(\sum_{i=1}^{n}\epsilon_i^2 + \frac{2\sigma^2}{b}\sum_{j=1}^{p}|\beta_j|\right)$$

which is equivalent to maximising

$$\sum_{i=1}^{n}\epsilon_i^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

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

Here we have come back to the Lasso optimisation problem.

d

$$p(\beta) = \prod_{i=1}^{p} p(\beta_i) = \prod_{i=1}^{p} \frac{1}{\sqrt{2\pi c}} \exp\left(-\frac{\beta_i^2}{2c}\right) = \left(\frac{1}{\sqrt{2\pi c}}\right)^p \exp\left(-\frac{1}{2c}\sum_{i=1}^{p}\beta_i^2\right)$$

So, the posterior for

$$\beta$$

can be written as

$$\mathcal{L}(Y|X,\beta)p(\beta) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2\right)\left(\frac{1}{\sqrt{2\pi c}}\right)^p \exp\left(-\frac{1}{2c}\sum_{i=1}^{p}\beta_i^2\right)$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \left(\frac{1}{\sqrt{2\pi c}}\right)^p \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2 - \frac{1}{2c}\sum_{i=1}^{p}\beta_i^2\right)$$

e

Using the same idea from (c), we would like to maximise

$$\mathcal{L}(Y|X,\beta)p(\beta) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \left(\frac{1}{\sqrt{2\pi c}}\right)^p \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2 - \frac{1}{2c}\sum_{i=1}^{p}\beta_i^2\right)$$

which is equivalent to maximising

$$\ln\left(\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \left(\frac{1}{\sqrt{2\pi c}}\right)^p\right) - \left(\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2 + \frac{1}{2c}\sum_{i=1}^{p}\beta_i^2\right)$$

and minimising the last term

$$\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2 + \frac{1}{2c}\sum_{i=1}^{p}\beta_i^2$$

$$minimise_\beta \left(\frac{1}{2\sigma^2}\sum_{i=1}^{n}\epsilon_i^2 + \frac{1}{2c}\sum_{i=1}^{p}\beta_i^2\right)$$

$$= minimise_\beta \frac{1}{2\sigma^2}\left(\sum_{i=1}^{n}\epsilon_i^2 + \frac{2\sigma^2}{2c}\sum_{i=1}^{p}\beta_i^2\right)$$

$$= minimise_\beta \left(\sum_{i=1}^{n}\epsilon_i^2 + \lambda\sum_{i=1}^{p}\beta_i^2\right) = minimise_\beta \left(\text{RSS} + \lambda\sum_{i=1}^{p}\beta_i^2\right)$$

Here we have come back to the Ridge optimisation problem.

Exercise 8

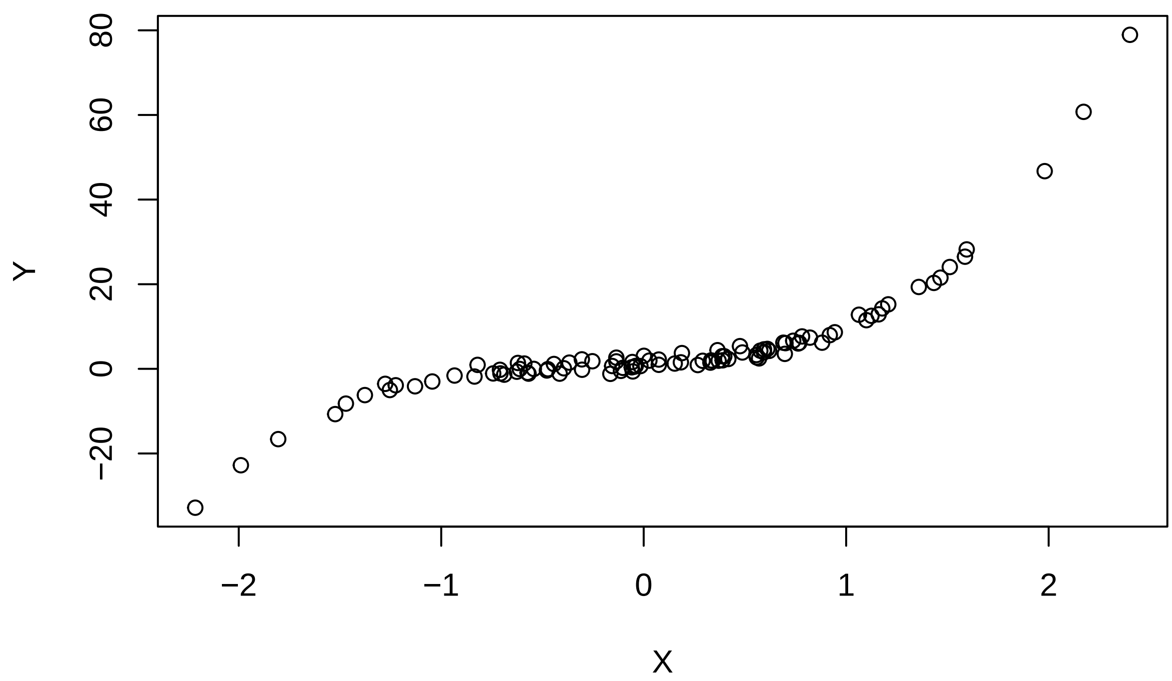
a

```
set.seed(1)
X = rnorm(100)
noise = rnorm(100)
```

b

```
Y = 1 + 2*X + 3*X^2 + 4*X^3 + noise
```

```
plot(X, Y)
```



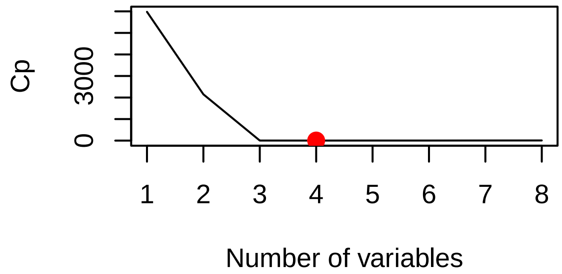
c

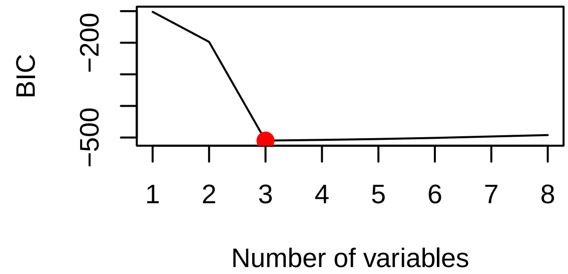
```
library(leaps)
```

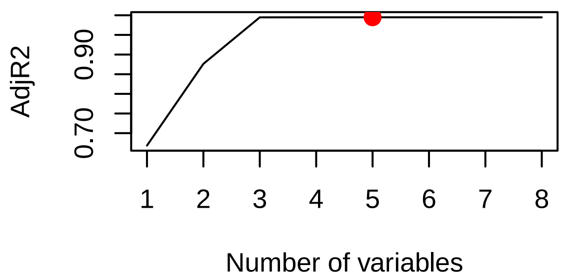
```r
data = data.frame(Y, X)
regfit_full = regsubsets(Y ~ poly(X, 10), data = data)
regfit_summary = summary(regfit_full)
regfit_summary
```

```
## Subset selection object
## Call: regsubsets.formula(Y ~ poly(X, 10), data = data)
## 10 Variables  (and intercept)
##                Forced in Forced out
## poly(X, 10)1      FALSE      FALSE
## poly(X, 10)2      FALSE      FALSE
## poly(X, 10)3      FALSE      FALSE
## poly(X, 10)4      FALSE      FALSE
## poly(X, 10)5      FALSE      FALSE
## poly(X, 10)6      FALSE      FALSE
## poly(X, 10)7      FALSE      FALSE
## poly(X, 10)8      FALSE      FALSE
## poly(X, 10)9      FALSE      FALSE
## poly(X, 10)10     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          poly(X, 10)1 poly(X, 10)2 poly(X, 10)3 poly(X, 10)4 poly(X, 10)5
## 1  ( 1 ) "*"          " "          " "          " "          " "
## 2  ( 1 ) "*"          " "          "*"          " "          " "
## 3  ( 1 ) "*"          "*"          "*"          " "          " "
## 4  ( 1 ) "*"          "*"          "*"          " "          "*"
## 5  ( 1 ) "*"          "*"          "*"          "*"          "*"
## 6  ( 1 ) "*"          "*"          "*"          "*"          "*"
## 7  ( 1 ) "*"          "*"          "*"          "*"          "*"
## 8  ( 1 ) "*"          "*"          "*"          "*"          "*"
##          poly(X, 10)6 poly(X, 10)7 poly(X, 10)8 poly(X, 10)9 poly(X, 10)10
## 1  ( 1 ) " "          " "          " "          " "          " "
## 2  ( 1 ) " "          " "          " "          " "          " "
## 3  ( 1 ) " "          " "          " "          " "          " "
## 4  ( 1 ) " "          " "          " "          " "          " "
## 5  ( 1 ) " "          " "          " "          " "          " "
## 6  ( 1 ) " "          " "          " "          " "          "*"
## 7  ( 1 ) " "          "*"          " "          " "          "*"
## 8  ( 1 ) " "          "*"          " "          "*"          "*"
```

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

plot(regfit_summary$cp, xlab = 'Number of variables', ylab = 'Cp', type = 'l')
cp_min = which.min(regfit_summary$cp)
points(cp_min, regfit_summary$cp[cp_min], col= 'red', cex = 2, pch = 20)

plot(regfit_summary$bic, xlab = 'Number of variables', ylab = 'BIC', type = 'l')
bic_min = which.min(regfit_summary$bic)
points(bic_min, regfit_summary$bic[bic_min], col= 'red', cex = 2, pch = 20)

plot(regfit_summary$adjr2, xlab = 'Number of variables', ylab = 'AdjR2', type = 'l')
adjr2_max = which.max(regfit_summary$adjr2)
points(adjr2_max, regfit_summary$adjr2[adjr2_max], col= 'red', cex = 2, pch = 20)
```








```
coef(regfit_full, 3)
```

```
##  (Intercept) poly(X, 10)1 poly(X, 10)2 poly(X, 10)3
##     4.454162   108.363598    45.043813    60.156861
```

```
coef(regfit_full, 4)
```

```
##  (Intercept) poly(X, 10)1 poly(X, 10)2 poly(X, 10)3 poly(X, 10)5
##     4.454162   108.363598    45.043813    60.156861     1.480188
```

```
coef(regfit_full, 5)
```

```
##  (Intercept) poly(X, 10)1 poly(X, 10)2 poly(X, 10)3 poly(X, 10)4 poly(X, 10)5
##     4.454162   108.363598    45.043813    60.156861     1.257095     1.480188
```

d

The result obtained from forward stepwise selection is similar to the result obtained from (c)

```
regfit_fwd = regsubsets(Y ~ poly(X, 10), data = data,
                        nvmax = 10, method = 'forward')
coef(regfit_fwd, 3)
```

```
## (Intercept) poly(X, 10)1 poly(X, 10)2 poly(X, 10)3
##    4.454162   108.363598    45.043813    60.156861
```

```
coef(regfit_fwd, 4)
```

```
## (Intercept) poly(X, 10)1 poly(X, 10)2 poly(X, 10)3 poly(X, 10)5
##    4.454162   108.363598    45.043813    60.156861     1.480188
```

```
coef(regfit_fwd, 5)
```

```
## (Intercept) poly(X, 10)1 poly(X, 10)2 poly(X, 10)3 poly(X, 10)4 poly(X, 10)5
##    4.454162   108.363598    45.043813    60.156861     1.257095     1.480188
```

And so is backward stepwise selection.

```
regfit_bwd = regsubsets(Y ~ poly(X, 10), data = data,
                        nvmax = 10, method = 'backward')
coef(regfit_bwd, 3)
```

```
## (Intercept) poly(X, 10)1 poly(X, 10)2 poly(X, 10)3
##    4.454162   108.363598    45.043813    60.156861
```

```
coef(regfit_bwd, 4)
```

```
## (Intercept) poly(X, 10)1 poly(X, 10)2 poly(X, 10)3 poly(X, 10)5
##    4.454162   108.363598    45.043813    60.156861     1.480188
```

```
coef(regfit_bwd, 5)
```

```
## (Intercept) poly(X, 10)1 poly(X, 10)2 poly(X, 10)3 poly(X, 10)4 poly(X, 10)5
##    4.454162   108.363598    45.043813    60.156861     1.257095     1.480188
```

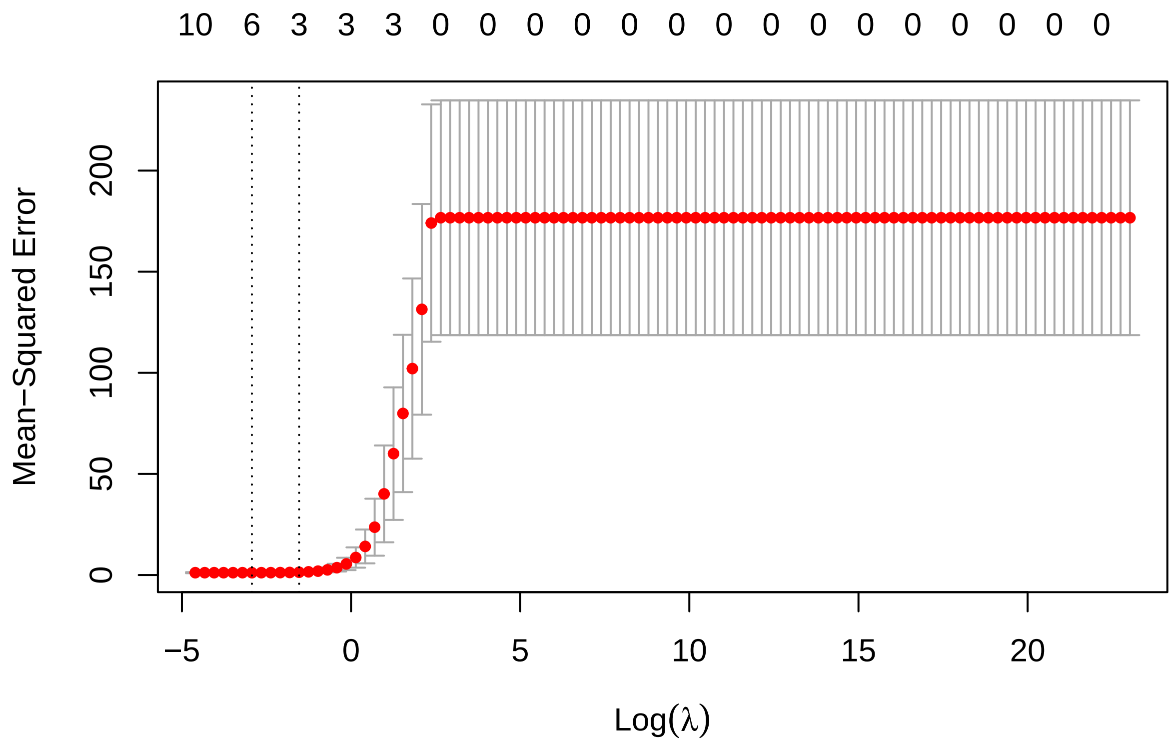
e

```
library(glmnet)
```

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

```
## Loaded glmnet 4.1-6
```

```
set.seed(1)

grid = 10^seq(10, -2, length = 100)
cv_lasso = cv.glmnet(poly(X, 10), Y, alpha = 1, lambda = grid)
plot(cv_lasso)
```



The Lasso result suggests that there 6 significant predictors for Y. The most significant predictors, however, still

$$X,\ X^2,\ X^3$$

and its coefficients are still virtually the same as in (c)

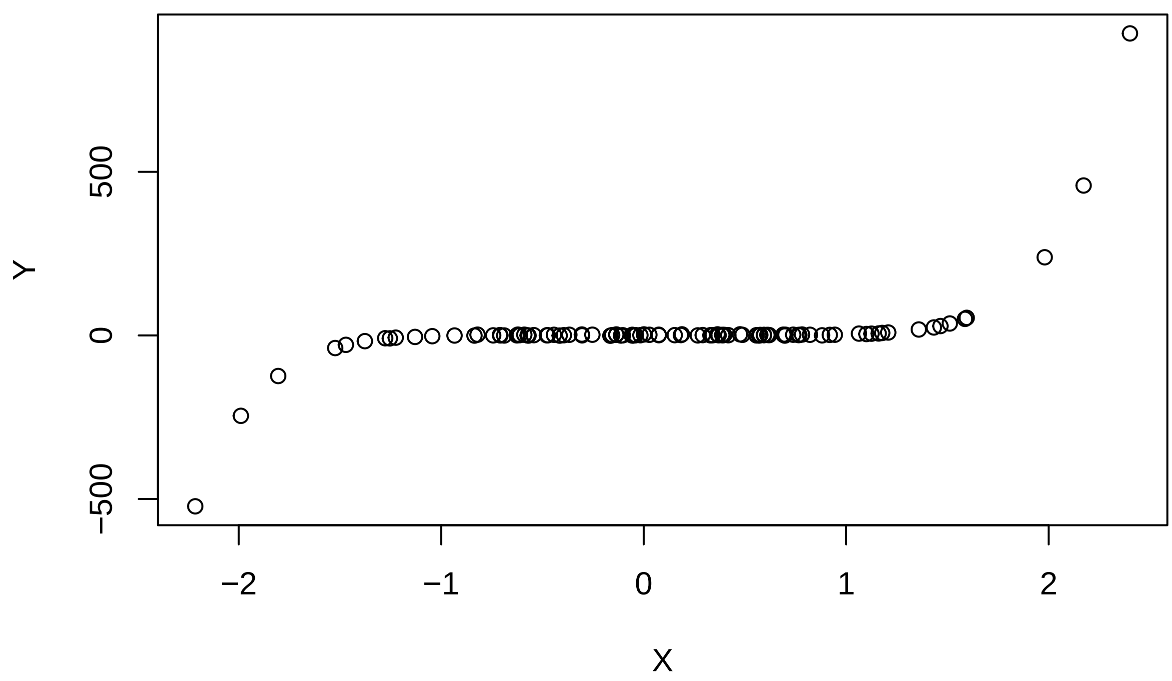
```
best_lambda = cv_lasso$lambda.min
coef(glmnet(poly(X, 10), Y, alpha = 1, lambda = best_lambda))
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept)   4.4541625
## 1           107.8299276
## 2            44.5101434
## 3            59.6231907
## 4             0.7234251
## 5             0.9465184
## 6             .
## 7             .
## 8             .
## 9             .
## 10           -0.4175596
```

f

Subset selection

```
Y = 1 + 2*X^7 + noise
plot(X, Y)
```




```
regfit_full = regsubsets(Y ~ poly(X, 10), data = data.frame(X, Y))
summary(regfit_full)
```

```
## Subset selection object
## Call: regsubsets.formula(Y ~ poly(X, 10), data = data.frame(X, Y))
## 10 Variables  (and intercept)
##                Forced in Forced out
## poly(X, 10)1      FALSE      FALSE
## poly(X, 10)2      FALSE      FALSE
## poly(X, 10)3      FALSE      FALSE
## poly(X, 10)4      FALSE      FALSE
## poly(X, 10)5      FALSE      FALSE
## poly(X, 10)6      FALSE      FALSE
## poly(X, 10)7      FALSE      FALSE
## poly(X, 10)8      FALSE      FALSE
## poly(X, 10)9      FALSE      FALSE
## poly(X, 10)10     FALSE      FALSE
```

```
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          poly(X, 10)1 poly(X, 10)2 poly(X, 10)3 poly(X, 10)4 poly(X, 10)5
## 1  ( 1 ) " "          " "          "*"          " "          " "
## 2  ( 1 ) "*"          " "          "*"          " "          " "
## 3  ( 1 ) "*"          " "          "*"          " "          "*"
## 4  ( 1 ) "*"          "*"          "*"          " "          "*"
## 5  ( 1 ) "*"          "*"          "*"          "*"          "*"
## 6  ( 1 ) "*"          "*"          "*"          "*"          "*"
## 7  ( 1 ) "*"          "*"          "*"          "*"          "*"
## 8  ( 1 ) "*"          "*"          "*"          "*"          "*"
##          poly(X, 10)6 poly(X, 10)7 poly(X, 10)8 poly(X, 10)9 poly(X, 10)10
## 1  ( 1 ) " "          " "          " "          " "          " "
## 2  ( 1 ) " "          " "          " "          " "          " "
## 3  ( 1 ) " "          " "          " "          " "          " "
## 4  ( 1 ) " "          " "          " "          " "          " "
## 5  ( 1 ) " "          " "          " "          " "          " "
## 6  ( 1 ) " "          "*"          " "          " "          " "
## 7  ( 1 ) "*"          "*"          " "          " "          " "
## 8  ( 1 ) "*"          "*"          " "          " "          "*"
```

```
coef(regfit_full, 1)
```

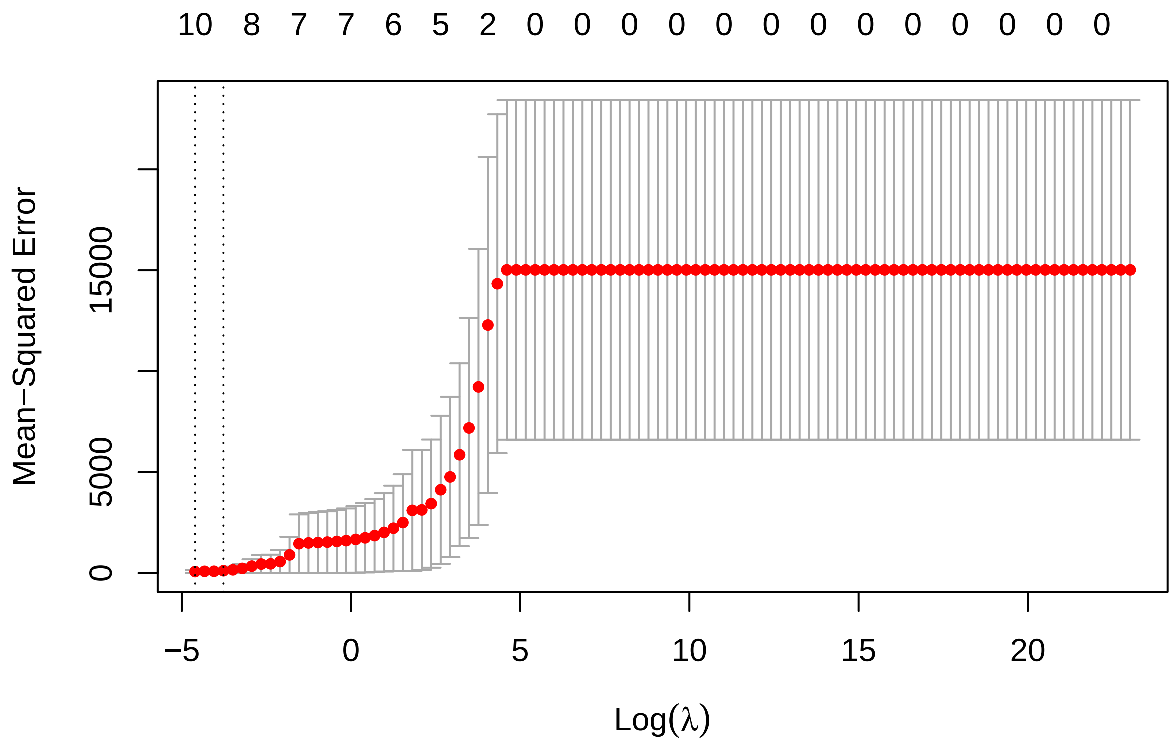
```
##  (Intercept) poly(X, 10)3
##     9.402906   886.454335
```

```
coef(regfit_full, 3)
```

```
##  (Intercept) poly(X, 10)1 poly(X, 10)3 poly(X, 10)5
##     9.402906   671.023798   886.454335   355.572453
```

Lasso

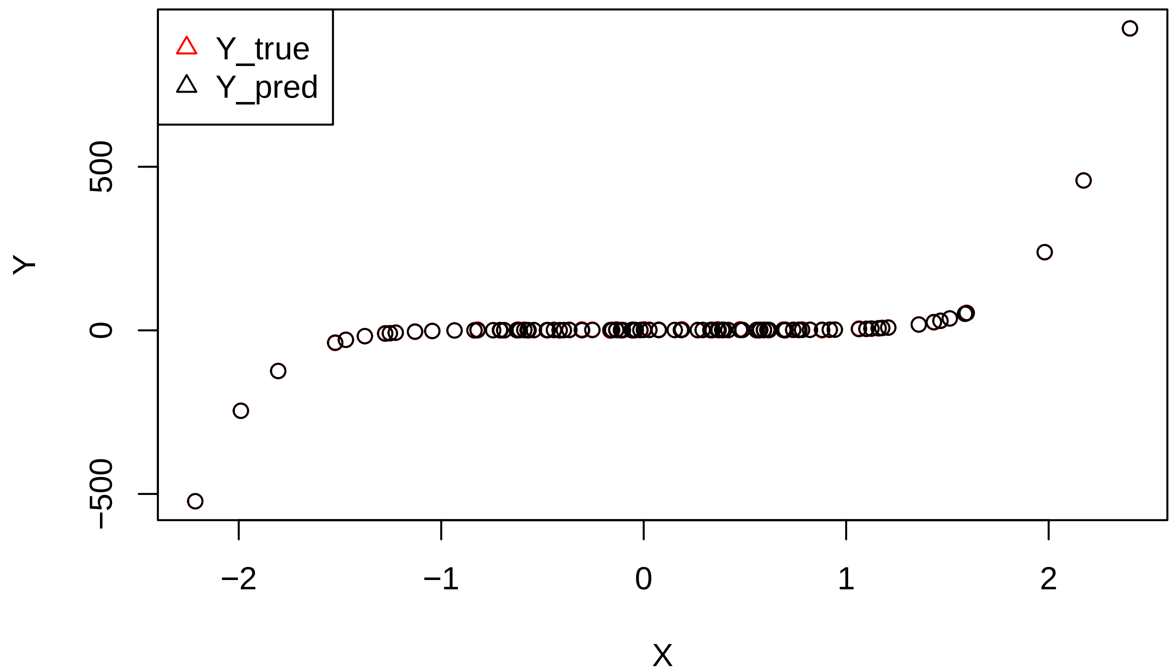
```
set.seed(1)

grid = 10^seq(10, -2, length = 100)
cv_lasso = cv.glmnet(poly(X, 10), Y, alpha = 1, lambda = grid)
plot(cv_lasso)
```




```
best_lambda = cv_lasso$lambda.min
coef(glmnet(poly(X, 10), Y, alpha = 1, lambda = best_lambda))
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept)   9.402906463
## 1           670.923797559
## 2           277.219375898
## 3           886.354335285
## 4           163.598846589
## 5           355.472452813
## 6            31.192187419
## 7            57.571882808
## 8            -0.007946591
## 9            -0.195840969
## 10           -0.851229512
```

```
coefficients = coef(glmnet(poly(X, 10), Y, alpha = 1, lambda = best_lambda))
Y_pred = poly(X, 10) %*% coefficients[2:11] + coefficients[1]
plot(X, Y, col = 'red')
points(X, Y_pred)
legend(legend = c('Y_true', 'Y_pred'),
       col = c('red', 'black'),
       x = 'topleft', lty = c(0, 0), pch = c(2, 2))
```



The subset selection and Lasso approach results agree with each other. However, both models overfit the data.

Exercise 9

```
library(ISLR2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
college = College %>% relocate(Apps, .before = Private)
head(college)
```

```
##                              Apps Private Accept Enroll Top10perc Top25perc
## Abilene Christian University 1660     Yes   1232    721        23        52
## Adelphi University           2186     Yes   1924    512        16        29
## Adrian College               1428     Yes   1097    336        22        50
## Agnes Scott College           417     Yes    349    137        60        89
## Alaska Pacific University     193     Yes    146     55        16        44
## Albertson College             587     Yes    479    158        38        62
##                              F.Undergrad P.Undergrad Outstate Room.Board Books
## Abilene Christian University        2885         537     7440       3300   450
## Adelphi University                  2683        1227    12280       6450   750
## Adrian College                      1036          99    11250       3750   400
## Agnes Scott College                  510          63    12960       5450   450
## Alaska Pacific University            249         869     7560       4120   800
## Albertson College                    678          41    13500       3335   500
##                              Personal PhD Terminal S.F.Ratio perc.alumni Expend
## Abilene Christian University     2200  70       78      18.1          12   7041
## Adelphi University               1500  29       30      12.2          16  10527
## Adrian College                   1165  53       66      12.9          30   8735
## Agnes Scott College               875  92       97       7.7          37  19016
## Alaska Pacific University        1500  76       72      11.9           2  10922
## Albertson College                 675  67       73       9.4          11   9727
##                              Grad.Rate
## Abilene Christian University        60
## Adelphi University                  56
## Adrian College                      54
## Agnes Scott College                 59
## Alaska Pacific University           15
## Albertson College                   55
```

```r
row.names(college) = NULL
```

a

```r
set.seed(1)
train_indices = sample(777, 666)
train_data = college[train_indices, ]
test_data = college[-train_indices, ]
Y_test_true = test_data$Apps
```

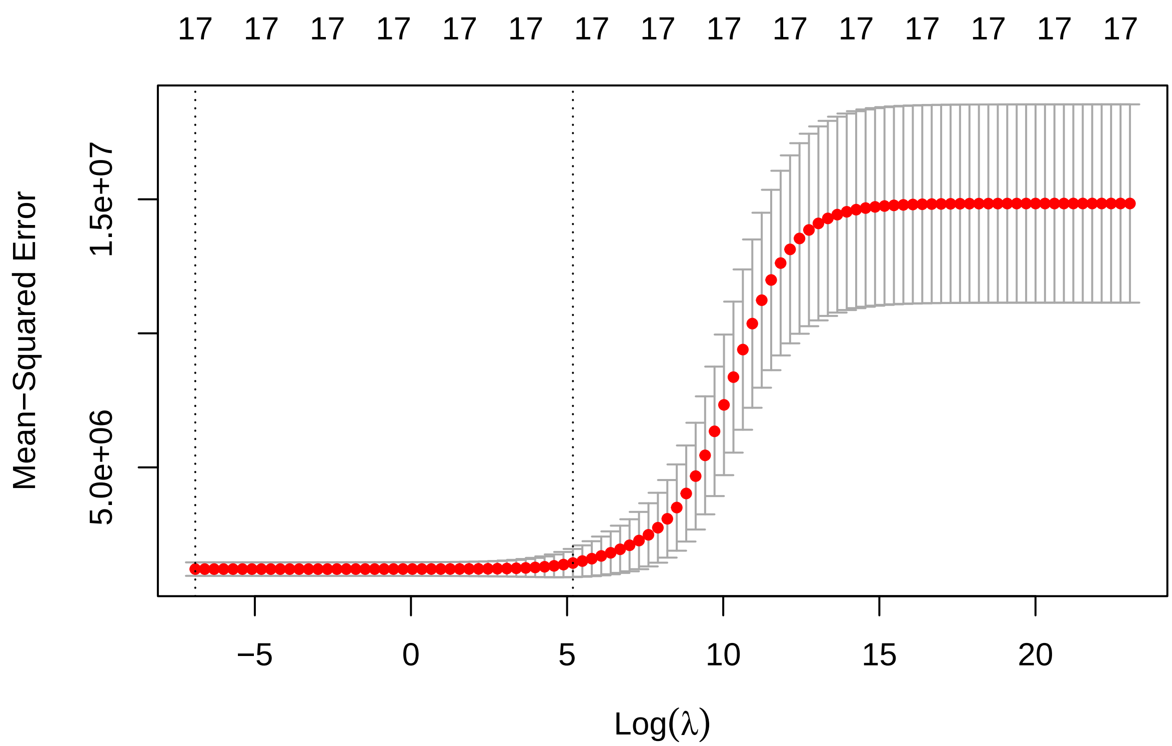
b

```r
lsr_model = lm(Apps ~ ., data = train_data)
Y_lsr_pred = predict(lsr_model, newdata = test_data)
lsr_rmse = sqrt(mean((Y_lsr_pred - Y_test_true)^2))
lsr_rmse
```

```
## [1] 1258.575
```

c

```
set.seed(1)
lambdas = 10^seq(10, -3, length = 100)
ridge_cv = cv.glmnet(model.matrix(Apps ~ ., data = train_data)[, -1],
                    train_data[, 1], lambda = lambdas, alpha = 0)
best_ridge_lambda = ridge_cv$lambda.min
plot(ridge_cv)
```




```
ridge_model = glmnet(model.matrix(Apps ~ ., data = train_data)[, -1],
                      train_data[, 1], lambda = best_ridge_lambda, alpha = 0)

Y_ridge_pred = predict(ridge_model, s = best_ridge_lambda,
                        newx = model.matrix(Apps ~ ., data = test_data)[, -1])

ridge_rmse = sqrt(mean(Y_ridge_pred - Y_test_true)^2)
ridge_rmse
```

```
## [1] 28.58395
```

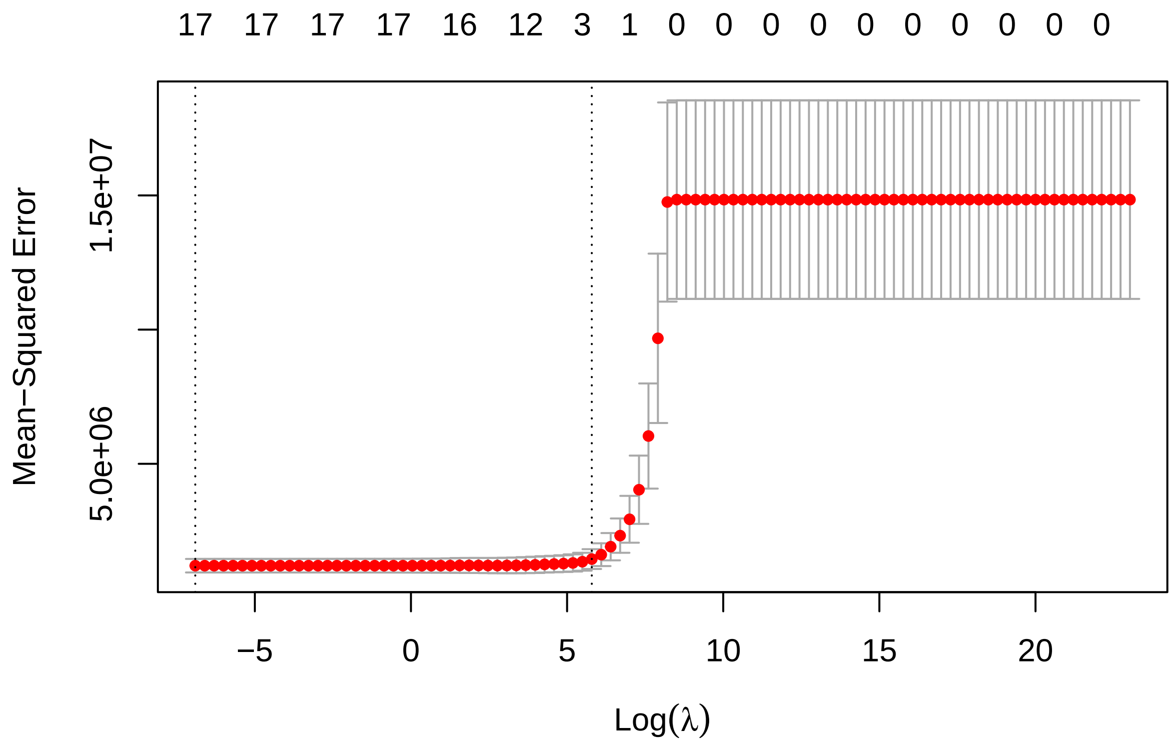
```
coef(ridge_model)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
```

```
##                        s0
## (Intercept) -5.991409e+02
## PrivateYes  -4.121533e+02
## Accept       1.652312e+00
## Enroll      -1.076249e+00
## Top10perc    4.883471e+01
## Top25perc   -1.346838e+01
## F.Undergrad  6.880337e-02
## P.Undergrad  5.768048e-02
## Outstate    -8.026011e-02
## Room.Board   1.551268e-01
## Books        1.559113e-01
## Personal     5.961177e-03
## PhD         -9.676085e+00
## Terminal     5.222348e-01
## S.F.Ratio    1.562605e+01
## perc.alumni -1.133992e-01
## Expend       6.031901e-02
## Grad.Rate    6.590556e+00
```

d

```r
set.seed(1)
lambdas = 10^seq(10, -3, length = 100)
lasso_cv = cv.glmnet(model.matrix(Apps ~ ., data = train_data)[, -1],
                     train_data[, 1], lambda = lambdas, alpha = 1)
best_lasso_lambda = lasso_cv$lambda.min
plot(lasso_cv)
```




```r
lasso_model = glmnet(model.matrix(Apps ~ ., data = train_data)[, -1],
                     train_data[, 1], lambda = best_lasso_lambda, alpha = 1)

Y_lasso_pred = predict(lasso_model, s = best_lambda,
                       newx = model.matrix(Apps ~ ., data = test_data)[, -1])

lasso_rmse = sqrt(mean(Y_lasso_pred - Y_test_true)^2)
lasso_rmse
```

```
## [1] 28.58374
```

Just another way to calculate predicted values

```r
Y_lasso_pred = model.matrix(Apps ~ ., data = test_data)[, -1] %*% coef(lasso_model)[2:18] + coef(lasso_

lasso_rmse = sqrt(mean(Y_lasso_pred - Y_test_true)^2)
lasso_rmse
```

```
## [1] 28.58374
```

```r
coef(lasso_model)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
```

```
## (Intercept) -5.991301e+02
## PrivateYes  -4.121532e+02
## Accept       1.652310e+00
## Enroll      -1.076220e+00
## Top10perc    4.883412e+01
## Top25perc   -1.346790e+01
## F.Undergrad  6.879886e-02
## P.Undergrad  5.768022e-02
## Outstate    -8.025915e-02
## Room.Board   1.551258e-01
## Books        1.559113e-01
## Personal     5.960282e-03
## PhD         -9.675641e+00
## Terminal     5.217603e-01
## S.F.Ratio    1.562556e+01
## perc.alumni -1.133733e-01
## Expend       6.031866e-02
## Grad.Rate    6.590366e+00
```

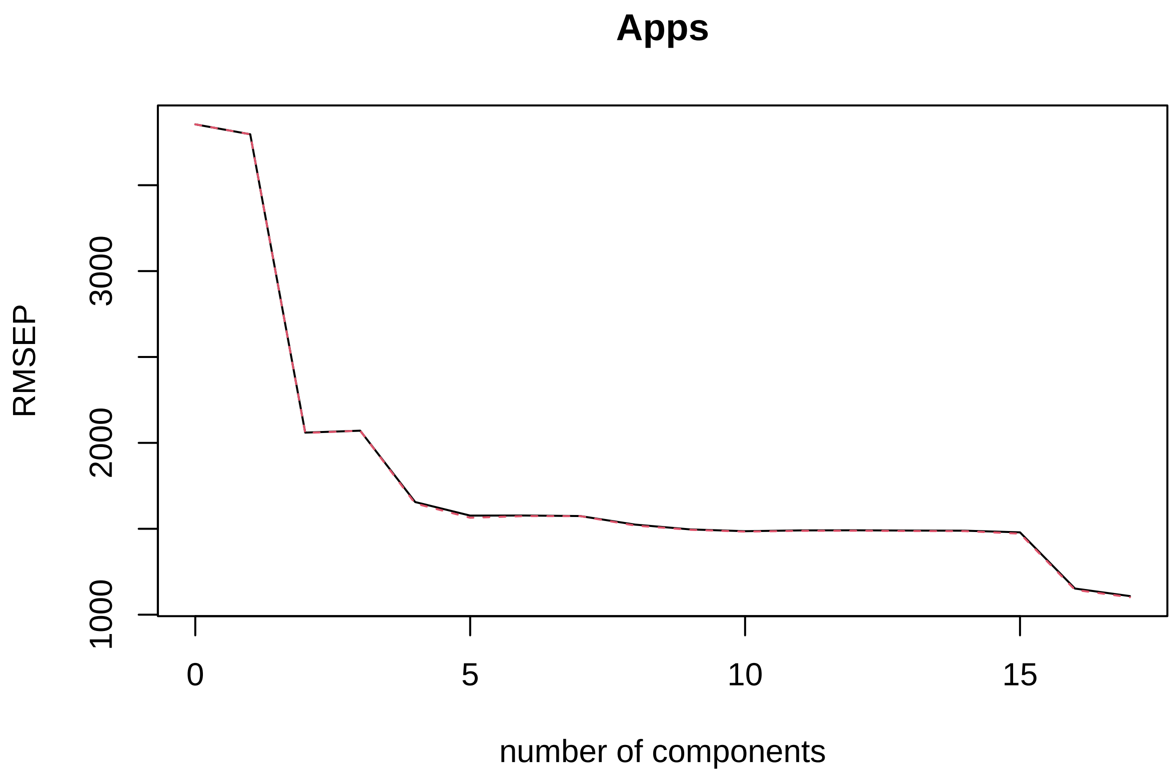
e

```r
library(pls)
```

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

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

```r
set.seed(1)
pcr_model = pcr(Apps ~ ., data = train_data, scale = T, validation = 'CV')
validationplot(pcr_model)
```



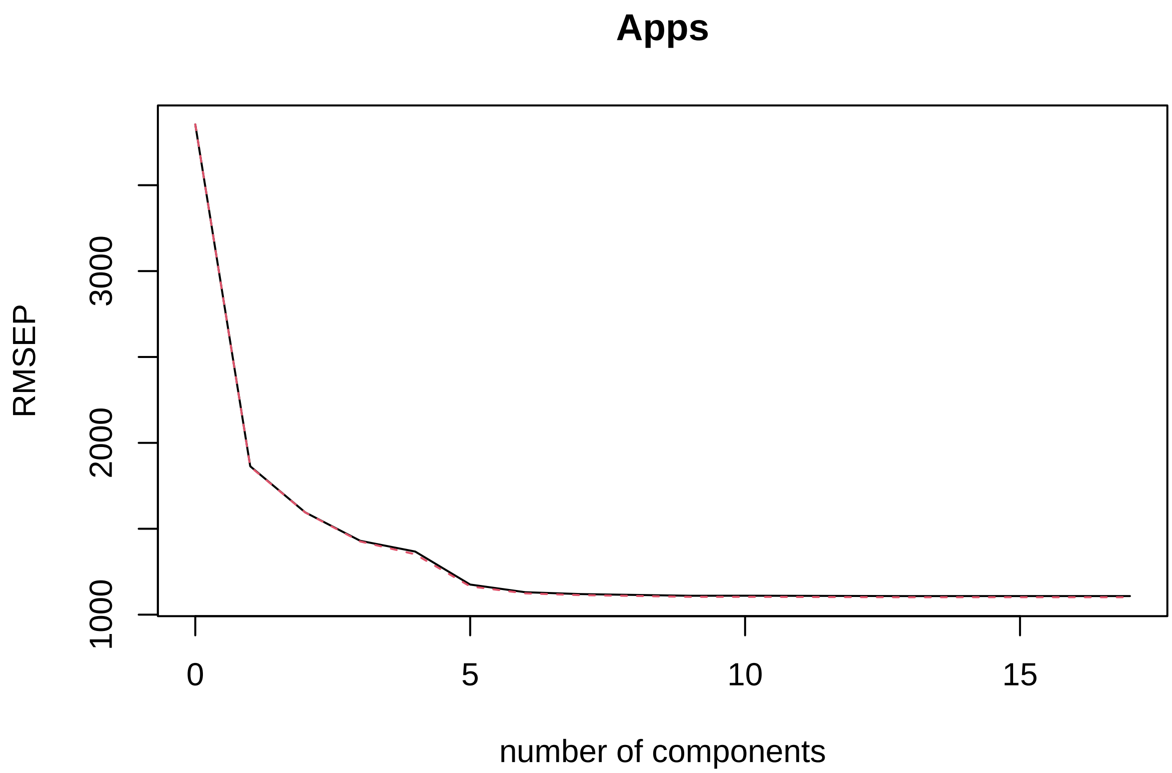
PCR is more difficult to interpret since it doesn't preform variable selection nor produce coefficient estimates.

```
pcr_model = pcr(Apps ~ ., data = train_data, scale = T, ncomp = 17)
Y_pcr_pred = predict(pcr_model, newdata = test_data[, 2:18])
pcr_rmse = sqrt(mean(Y_pcr_pred - Y_test_true)^2)
pcr_rmse
```

```
## [1] 33.35977
```

f

```
set.seed(1)
plsr_model = plsr(Apps ~ ., data = train_data, scale = T, validation = 'CV')
validationplot(plsr_model)
```




```r
summary(plsr_model)
```

```
## Data:    X dimension: 666 17
##  Y dimension: 666 1
## Fit method: kernelpls
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            3854     1864     1596     1430     1367     1175     1131
## adjCV         3854     1861     1595     1426     1351     1165     1124
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1120     1115     1110      1110      1110      1109      1108
## adjCV     1114     1109     1104      1104      1103      1102      1101
##        14 comps  15 comps  16 comps  17 comps
## CV         1108      1108      1108      1108
## adjCV      1102      1102      1102      1102
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       25.58    42.45    62.50    65.05    68.19    73.25    77.04    80.61
## Apps    77.50    84.44    87.58    90.80    92.65    93.05    93.14    93.19
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       82.42     85.07     87.58     90.96     92.80     95.07     97.11
```

```
## Apps    93.27     93.30     93.32     93.33     93.33     93.33     93.33
##       16 comps  17 comps
## X        98.31    100.00
## Apps     93.33     93.33
```

```
plsr_model = plsr(Apps ~ ., data = train_data, scale = T, ncomp = 13)
Y_plsr_pred = predict(plsr_model, newdata = test_data[, 2:18])
plsr_rmse = sqrt(mean(Y_plsr_pred - Y_test_true)^2)
plsr_rmse
```

```
## [1] 14.54268
```

g

We can predict the number of college applications received pretty accurately using Ridge Regression, Lasso, PCR or Partial least square approach. The Ridge and Lasso models have the same results and are interpretable. The PCR and PLS methods provide different results and are difficult to interpret.

Exercise 10

a

```
set.seed(1)
X = matrix(rnorm(1000 * 20), 1000, 20)
e = rnorm(1000)
B = rnorm(20) + 1

for (i in c(1,3,5,7,9)){
  B[i] = 0
}

Y = X %*% B + e
```

b

```
train_data = data.frame(Y, X)[1:100, ]
test_data = data.frame(Y, X)[101: 1000, ]
head(train_data)
```

```
##           Y         X1          X2          X3         X4         X5         X6
## 1 -6.319744 -0.6264538  1.13496509 -0.88614959  0.7391149 -1.1346302 -1.5163733
## 2  4.938041  0.1836433  1.11193185 -1.92225490  0.3866087  0.7645571  0.6291412
## 3  2.200177 -0.8356286 -0.87077763  1.61970074  1.2963972  0.5707101 -1.6781940
## 4 -5.389456  1.5952808  0.21073159  0.51926990 -0.8035584 -1.3516939  1.1797811
## 5 -4.284060  0.3295078  0.06939565 -0.05584993 -1.6026257 -2.0298855  1.1176545
## 6  5.955736 -0.8204684 -1.66264885  0.69641761  0.9332510  0.5904787 -1.2377359
##            X7         X8         X9        X10        X11        X12
```

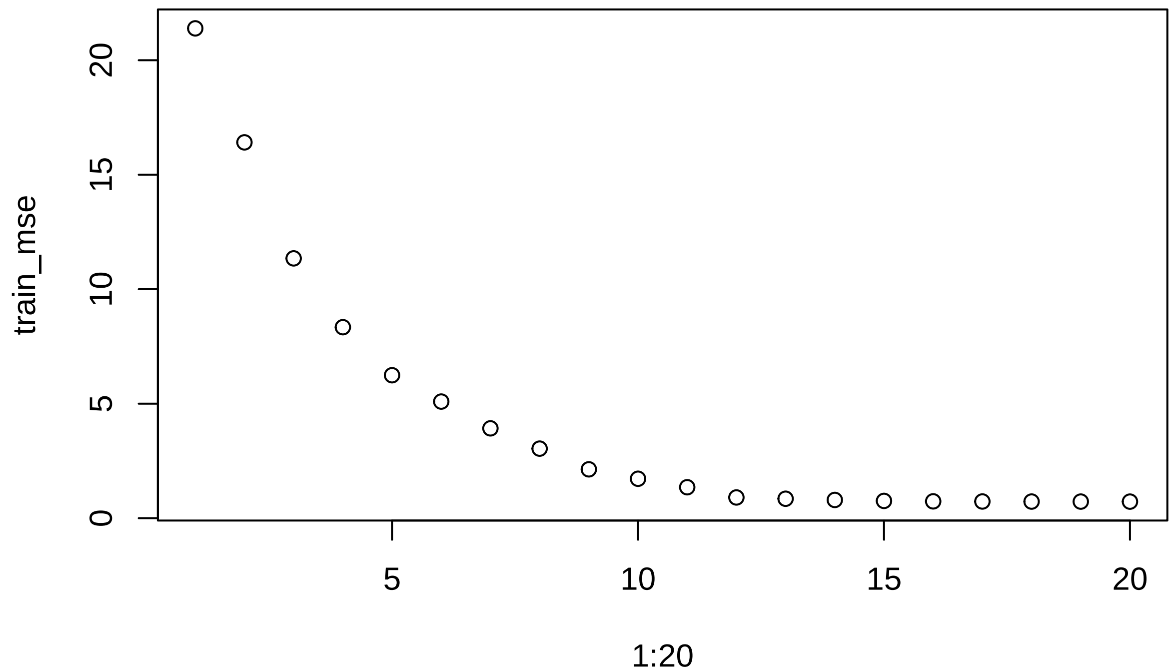
```
## 1 -0.61882708 -1.3254177  0.2637034 -1.2171201 -0.8043316 -1.4115219
## 2 -1.10942196  0.9519797 -0.8294518 -0.9462293 -1.0565257  1.0838697
## 3 -2.17033523  0.8600044 -1.4616348  0.0914098 -1.0353958  1.1702224
## 4 -0.03130307  1.0607903  1.6839902  0.7013513 -1.1855604  0.2947545
## 5 -0.26039848 -0.3505840 -1.5443243  0.6734224 -0.5004395 -0.5544277
## 6  0.53443047 -0.1307656 -0.1908871  1.2655534 -0.5249887 -0.4034407
##           X13        X14        X15        X16        X17        X18        X19
## 1 -0.93910663  0.2264537  0.5232667 -0.2139090  0.8576341  1.0496171  0.9514099
## 2  1.39366493 -0.8185942  0.9935537 -0.1067233 -1.6253951  0.2903237  0.4570987
## 3  1.62581486 -0.8471526  0.2737370 -0.4645893 -0.2342783  1.2421262 -0.3586935
## 4  0.40900106 -1.9843326 -0.6949193 -0.6842725 -1.0326545 -0.6850857 -1.0458614
## 5 -0.09255856 -0.8127788 -0.7180502 -0.7908007 -1.1411412 -0.6677681  0.3075345
## 6  0.20609871  1.4616707 -0.1019895 -0.3389638 -1.5219369  0.9409138  1.9943876
##           X20
## 1 -2.07771241
## 2 -0.45446091
## 3 -0.16555991
## 4  0.89765209
## 5 -0.02948916
## 6  1.85838843
```

c

```
predict_regsubsets = function ( object , newdata , id , ...) {
  form = as.formula ( object $ call [[2]])
  mat = model.matrix ( form , newdata )
  coefi = coef ( object , id = id )
  xvars = names ( coefi )
  mat [ , xvars ] %*% coefi
}
```

```
ss_model = regsubsets(Y ~ ., data = train_data, nvmax = 20)

train_mse = rep(NA, 20)
for (i in 1:20){
  Y_train_pred = predict_regsubsets(ss_model, train_data, id = i)
  train_mse[i] = mean((Y_train_pred - train_data$Y)^2)
}
```

```
plot(1:20, train_mse)
```




```
which.min(train_mse)
```

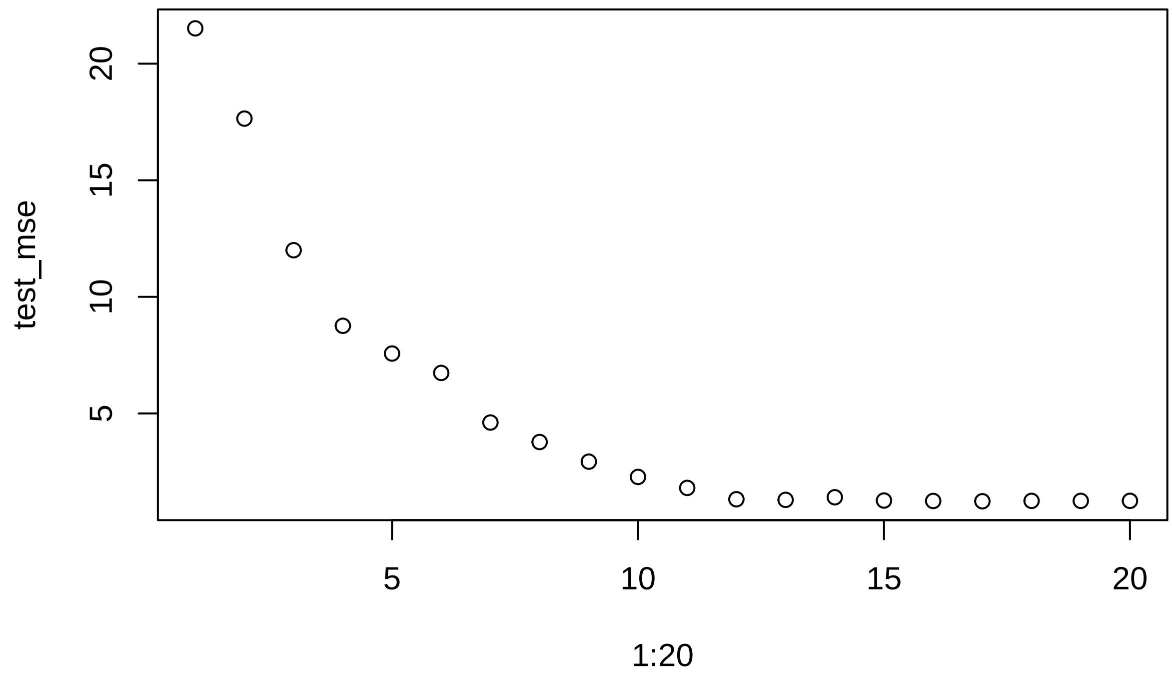
```
## [1] 20
```

d

```
test_mse = rep(NA, 20)
for (i in 1:20){
  Y_test_pred = predict_regsubsets(ss_model, test_data, id = i)
  test_mse[i] = mean((Y_test_pred - test_data$Y)^2)
}

dim(Y_test_pred)
```

```
## [1] 900   1
```

```
plot(1:20, test_mse)
```




```
which.min(test_mse)
```

```
## [1] 17
```

e

The test set MSE takes on its minimum value for the 17-variable model. Specifically, it excludes the first, the seventh and ninth variable (recall that the true values for variable number 1, 3, 5, 7, 9 are zero).

```
coef(ss_model, id = 17)
```

```
## (Intercept)          X2          X3          X4          X5          X6
##  0.08281437 -1.89582277 -0.19897622  3.16985652  0.08566017  2.30132409
##          X8         X10         X11         X12         X13         X14
## -0.15497368 -0.89299492  1.65265479  0.61091183  1.35426550  0.99837433
##         X15         X16         X17         X18         X19         X20
##  2.25435748 -0.60155986 -0.81274417 -0.24446678  0.22429176  1.12188454
```

f

The mean square of the difference in coefficients is 0.097 and its standard deviation is 0.047.

```r
best_coef = coef(ss_model, id = 17)
best_variables = rownames(data.frame(best_coef))
difference = rep(NA, 17)
for (i in 2:length(best_variables)){
  num = as.numeric(gsub('X', '', best_variables[i]))
  sqr = sqrt((B[num] - best_coef[i])^2)
  difference[i-1] = sqr
}

mean(B)
```

```
## [1] 0.4369269
```

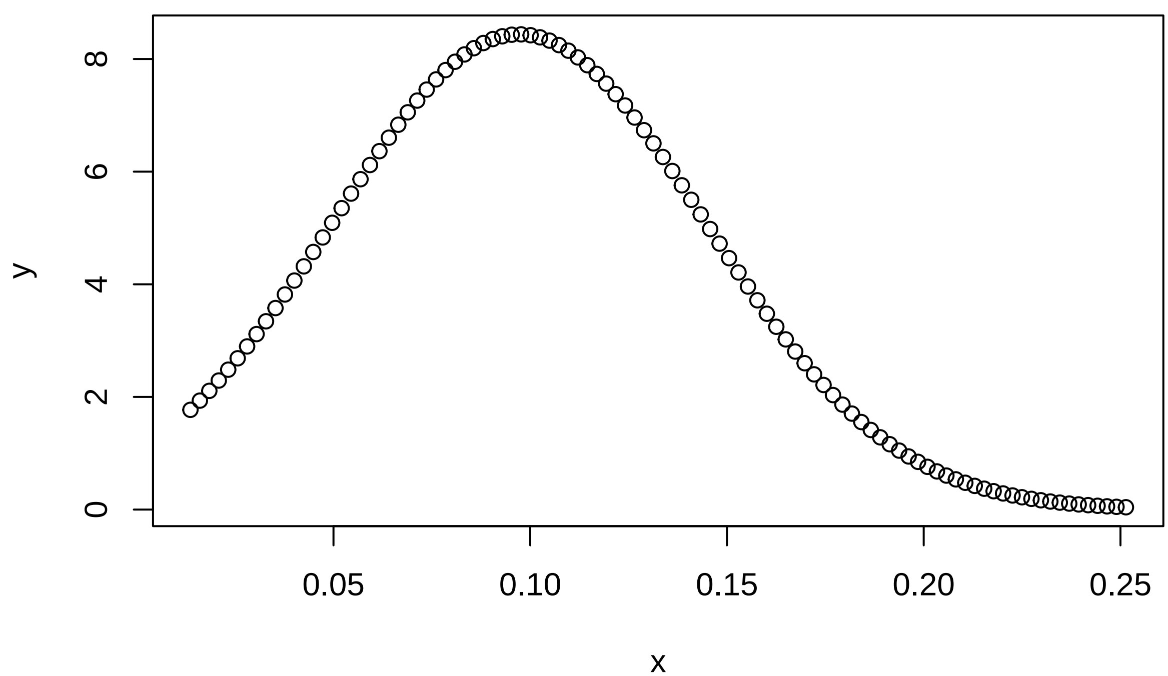
```r
mean(difference)
```

```
## [1] 0.09718582
```

```r
sd(difference)
```

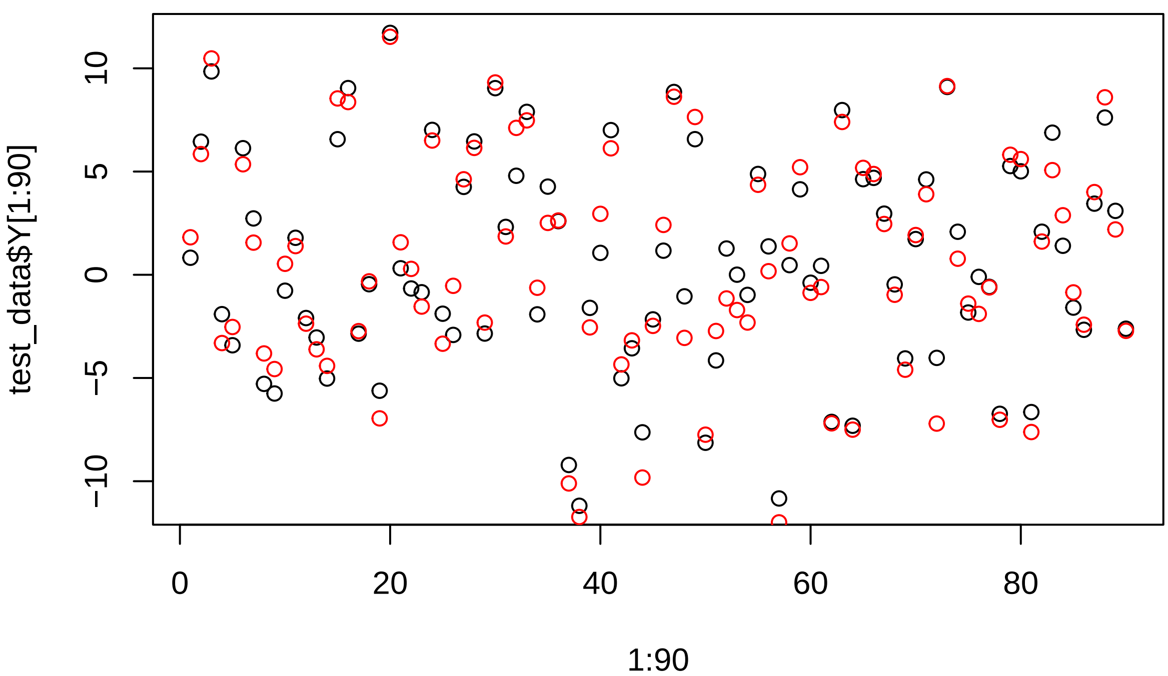
```
## [1] 0.04727607
```

```r
x = seq(min(B), max(B), length = 100) * sd(difference) + mean(difference)
y = dnorm(x, mean(difference), sd(difference))
plot(x, y)
```



Here we can have a look at how accurate the prediction is from the first 90 test observations. It is not too surprising or disapointing.

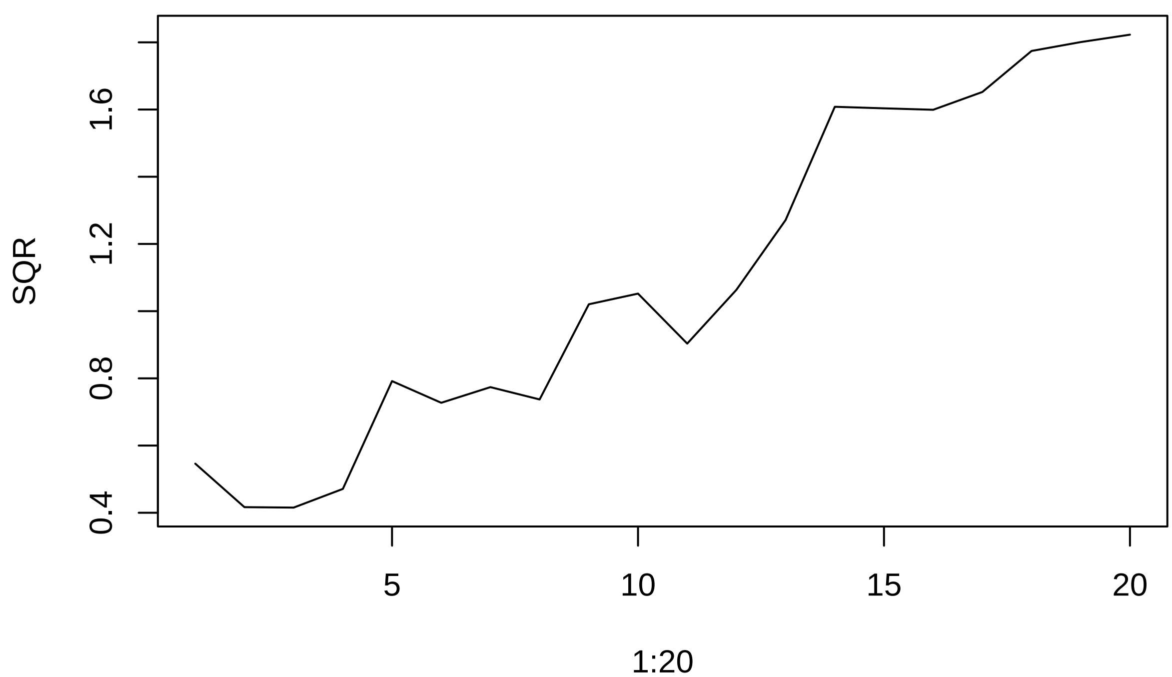
```
Y_test_pred = predict_regsubsets(ss_model, test_data, id = 17)
plot(1:90, test_data$Y[1:90])
points(1:90, Y_test_pred[1:90], col = 'red')
```



g

```
SQR = rep(NA, 20)

for (r in 1:20){
  variable_names = rownames(data.frame(coef(ss_model, id = r)))
  coefficients = coef(ss_model, id = r)
  sqr = 0
  for (i in 2:length(variable_names)){
    num = as.numeric(gsub('X', '', variable_names[i]))
    sqr = sqr + sqrt((B[num] - coefficients[i])^2)
  }
  SQR[r] = sqr
}
```

```r
plot(1:20, SQR, type = 'l')
```



The plot displaying the difference in coefficients is pretty different from the test MSE plot. The more variables used, the greater the difference. This makes sense since it is more likely to get closer to the true coefficients if we use less variables. However, we should pay more attention to the response values.

Exercise 11

```r
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio lstat medv
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3  4.98 24.0
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8  9.14 21.6
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8  4.03 34.7
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7  2.94 33.4
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7  5.33 36.2
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7  5.21 28.7
```

```r
dim(Boston)
```

```
## [1] 506  13
```

```
set.seed(1)

shuffled_data = Boston[sample(1:506, ), ]
train_data = shuffled_data[1:400, ]
test_data = shuffled_data[401:506, ]
```

a

Subset selection

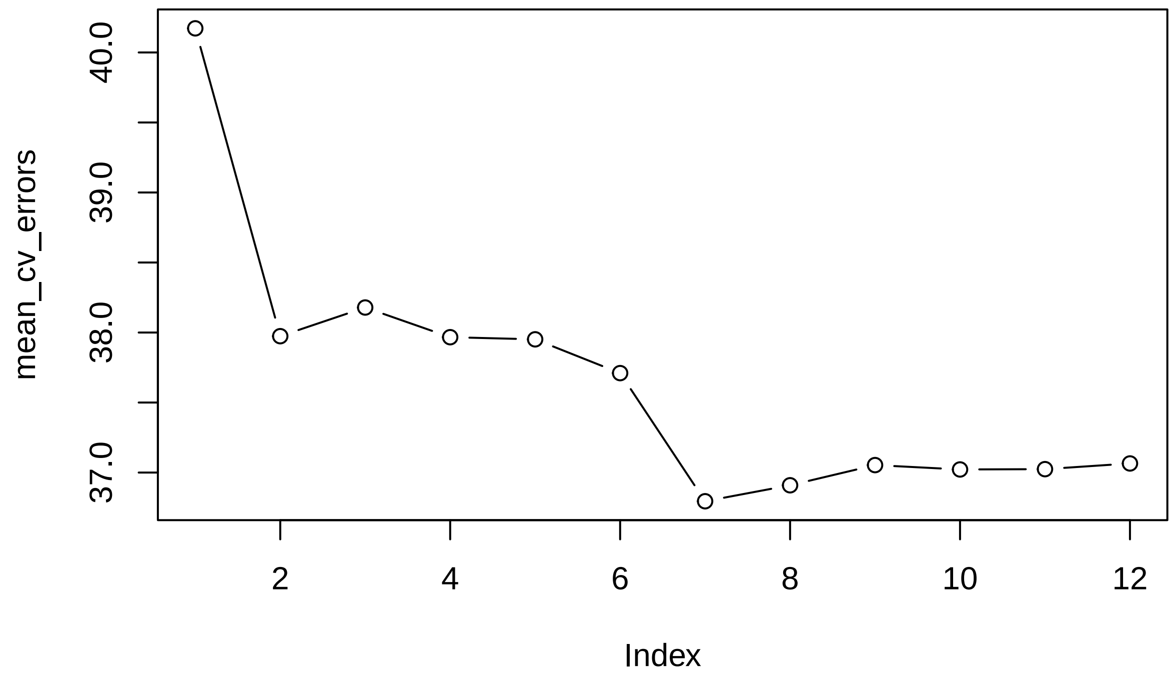
```
set.seed(1)

k = 10
n = nrow(train_data)
folds = sample(rep(1:k, length = n))
cv_errors = matrix(NA, k, 12, dimnames = list(NULL, paste(1:12)))
```

```
for (j in 1:k){

  best_fit = regsubsets(crim ~., data = train_data[folds != j, ], nvmax = 12)
  for (i in 1:12){
    pred = predict_regsubsets(best_fit, train_data[folds == j, ], id = i)
    cv_errors[j, i] = mean((train_data$crim[folds == j] - pred)^2)
  }
}
```

```
mean_cv_errors = apply(cv_errors, 2 ,mean)
mean_cv_errors
```

```
##        1        2        3        4        5        6        7        8
## 40.17205 37.97388 38.17854 37.96661 37.95132 37.71017 36.79517 36.90919
##        9       10       11       12
## 37.05325 37.02213 37.02439 37.06494
```

```
plot(mean_cv_errors, type = 'b')
```




```r
best_ss_model = regsubsets(crim ~., data = train_data, nvmax = 12)
coef(best_ss_model, 7)
```

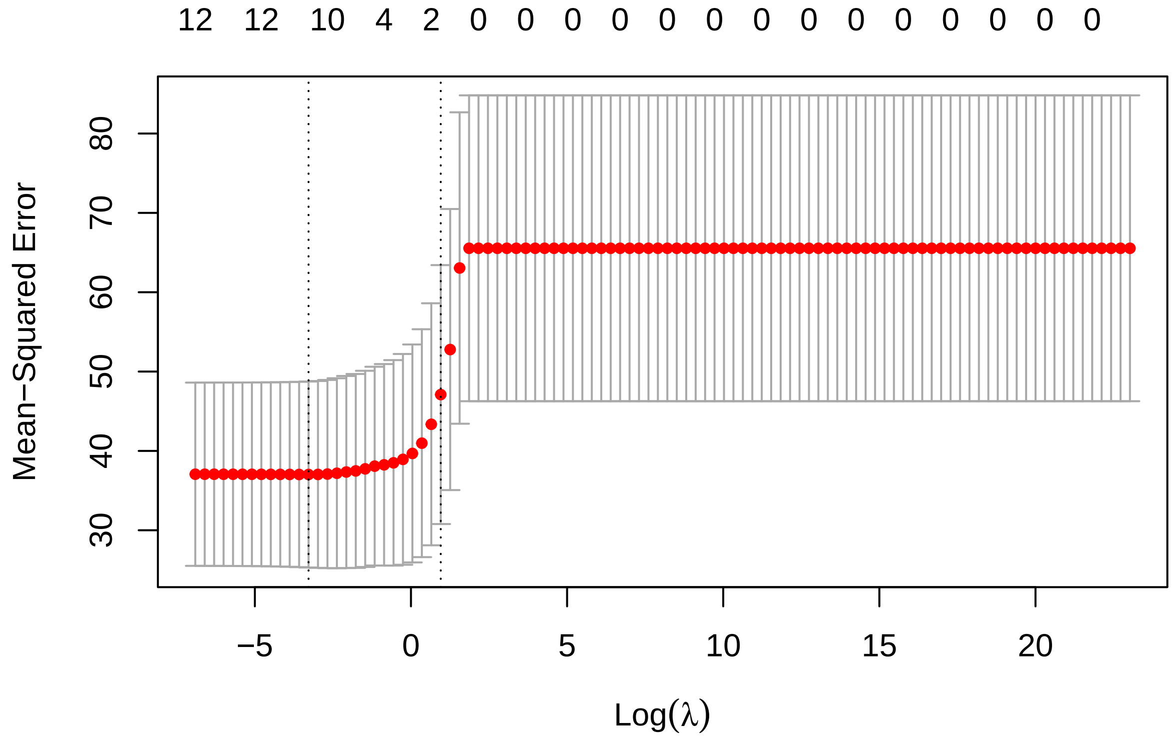
```
##  (Intercept)           zn          nox          dis          rad      ptratio
##  15.45958608   0.03684795 -10.53510479  -0.76967231   0.53526634  -0.36784734
##        lstat         medv
##   0.13598482  -0.16174992
```

```r
ss_test_pred = predict_regsubsets(best_ss_model, test_data, id = 7)
ss_test_rmse = sqrt(mean(ss_test_pred - test_data$crim)^2)
ss_test_rmse
```

```
## [1] 0.4836526
```

Lasso

```r
set.seed(1)
lambdas = 10^seq(10, -3, length = 100)
lasso_cv = cv.glmnet(model.matrix(crim ~ ., data = train_data)[, -1],
                     train_data[, 1], lambda = lambdas, alpha = 1)
best_lasso_lambda = lasso_cv$lambda.min
plot(lasso_cv)
```

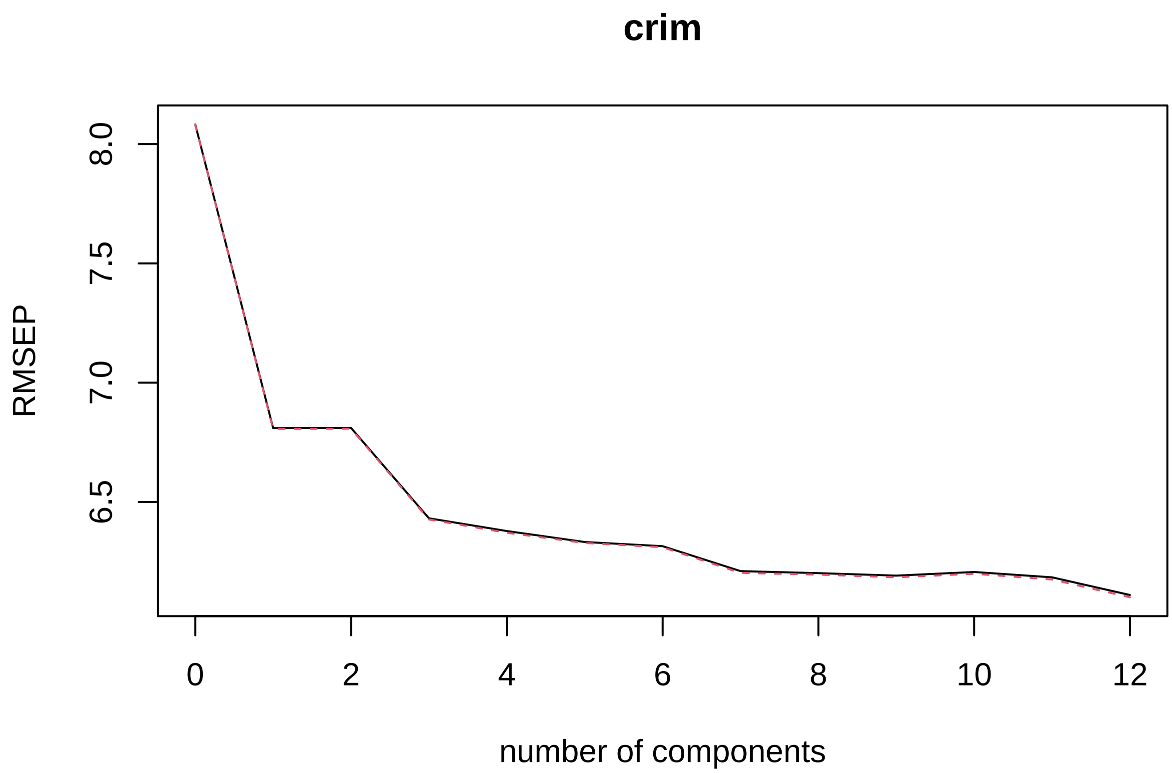



```r
lasso_model = glmnet(model.matrix(crim ~ ., data = train_data)[, -1],
                     train_data[, 1], lambda = best_lasso_lambda, alpha = 1)

crim_lasso_pred = predict(lasso_model, s = best_lasso_lambda,
                          newx = model.matrix(crim ~ ., data = test_data)[, -1])

lasso_test_rmse = sqrt(mean(crim_lasso_pred - test_data$crim)^2)
lasso_test_rmse
```

```
## [1] 0.4310731
```

PCR

```r
set.seed(1)
pcr_model = pcr(crim ~ ., data = train_data, scale = T, validation = 'CV')
validationplot(pcr_model)
```

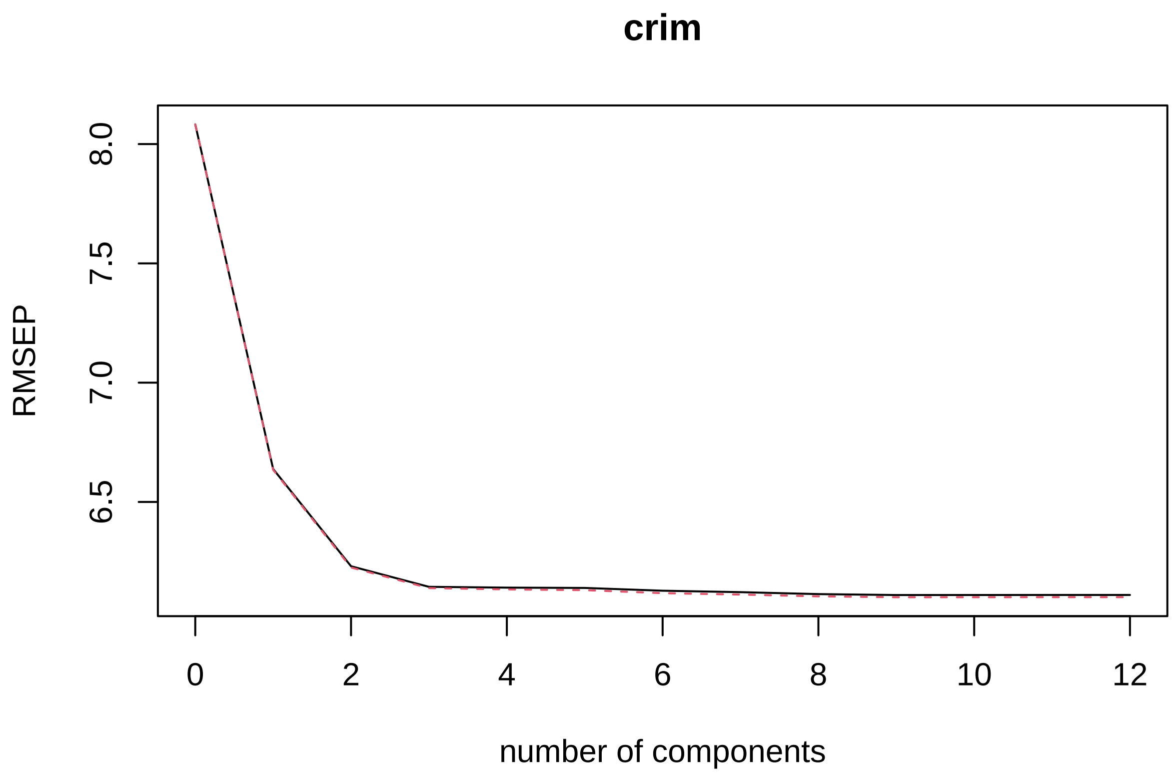



```
pcr_model = pcr(crim ~ ., data = train_data, scale = T, ncomp = 12)
crim_pcr_pred = predict(pcr_model, newdata = test_data)
pcr_rmse = sqrt(mean(crim_pcr_pred - test_data$crim)^2)
pcr_rmse
```

```
## [1] 0.5577684
```

Partial Least Square

```
set.seed(1)
plsr_model = plsr(crim ~ ., data = train_data, scale = T, validation = 'CV')
validationplot(plsr_model)
```




```
data.frame(plsr_model$validation$adj)
```

```
##      X1.comps X2.comps X3.comps X4.comps X5.comps X6.comps X7.comps X8.comps
## crim  43.1557 37.55865 36.28363 35.84525 35.63679 35.41837 35.36053 35.32641
##      X9.comps X10.comps X11.comps X12.comps
## crim 35.31127  35.31072  35.31094  35.31095
```

```
plsr_model = plsr(crim ~ ., data = train_data, scale = T, ncomp = 10)
crim_plsr_pred = predict(plsr_model, newdata = test_data)
plsr_rmse = sqrt(mean(crim_plsr_pred - test_data$crim)^2)
plsr_rmse
```

```
## [1] 0.4501447
```

b

Among the four approaches from (a), we could choose the Lasso as the best model since it gives us the lowest test error and it is also more interpretable, relative to PCR and PLS.

c

```
coef(lasso_model)
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept) 10.53098566
## zn           0.03147466
## indus       -0.06489504
## chas        -0.56239142
## nox         -5.97899262
## rm           0.16039482
## age          .
## dis         -0.69181560
## rad          0.52278267
## tax          .
## ptratio     -0.27725091
## lstat        0.14532994
## medv        -0.15018838
```

Using all variables would lead to overfitting. The Lasso model only uses 10 over 12 variables.