# 6.8: Exercises

- Exercise 8
- Exercise 9
- Exercise 10
- Exercise 11

# Exercise 8

In this exercise, we will generate simulated data, and will then use this data to perform best subset selection.

a. Use the **rnorm()** function to generate a predictor *X* of length $n = 100$, as well as a noise vector $\epsilon$ of length $n = 100$.

```
set.seed(1)
x = rnorm(100)
epsilon = rnorm(100)
```

b. Generate a response vector *Y* of length $n = 100$ according to the model

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$$

where $\beta_0, \beta_1, \beta_2$, and $\beta_3$ are constants of your choice.

```
y = 1 + 2 * x + 3 * x^2 + 4 * x^3 + epsilon
```

$\beta_0 = 1, \beta_1 = 2, \beta_2 = 3, \beta_3 = 4$

c. Use the **regsubsets()** function to perform best subset selection in order to choose the best model containing the predictors $X, X^2, \ldots, X^{10}$. What is the best model obtained according to $C_p$, BIC, and adjusted $R^2$? Show some plots to provide evidence for your answer, and report the coefficients of the best model obtained. Note you will need to use the **data.frame()** function to create a single data set containing both *X* and *Y*.

```
library(leaps)
```

```
## Warning: package 'leaps' was built under R version 3.4.4
```

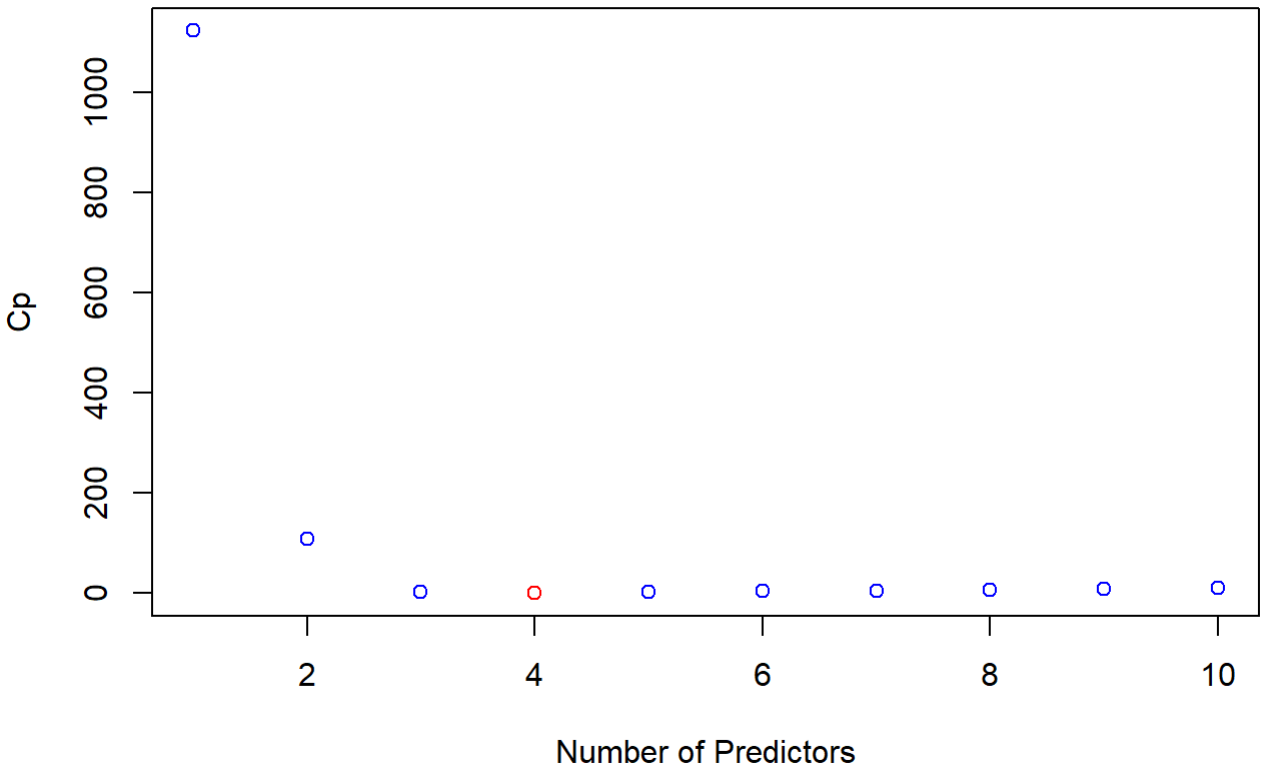
```
data = data.frame(x = x, y = y)
regfit.full = regsubsets(y ~ poly(x, 10, raw = TRUE), data = data, nvmax = 10)
reg.summary = summary(regfit.full)
reg.summary
```

```
## Subset selection object
## Call: regsubsets.formula(y ~ poly(x, 10, raw = TRUE), data = data,
##     nvmax = 10)
## 10 Variables  (and intercept)
##                           Forced in Forced out
## poly(x, 10, raw = TRUE)1      FALSE      FALSE
## poly(x, 10, raw = TRUE)2      FALSE      FALSE
## poly(x, 10, raw = TRUE)3      FALSE      FALSE

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

```
## 3  ( 1 )  " "                         " "
## 4  ( 1 )  " "                         " "
## 5  ( 1 )  " "                         " "
## 6  ( 1 )  "*"                         "*"
## 7  ( 1 )  " "                         "*"
## 8  ( 1 )  " "                         "*"
## 9  ( 1 )  " "                         "*"
## 10  ( 1 ) "*"                         "*"
##           poly(x, 10, raw = TRUE)9 poly(x, 10, raw = TRUE)10
## 1  ( 1 )  " "                         " "
## 2  ( 1 )  " "                         " "
## 3  ( 1 )  " "                         " "
## 4  ( 1 )  " "                         " "
## 5  ( 1 )  " "                         " "
## 6  ( 1 )  "*"                         " "
## 7  ( 1 )  " "                         "*"
## 8  ( 1 )  "*"                         "*"
## 9  ( 1 )  "*"                         "*"
## 10  ( 1 ) "*"                         "*"
```

```
idx = which.min(reg.summary$cp)
plot(reg.summary$cp, xlab = 'Number of Predictors', ylab = 'Cp', col = 'blue')
points(idx, reg.summary$cp[idx], col = 'red')
```

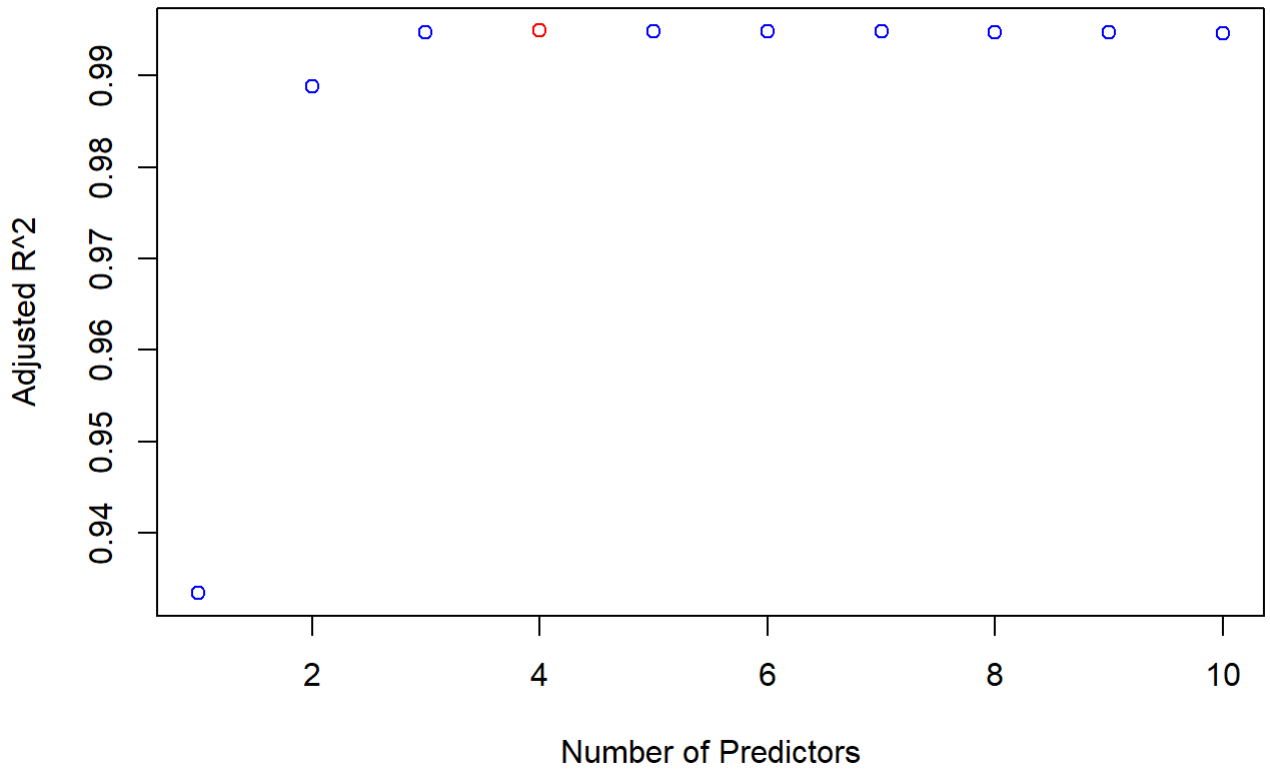


```
coef(regfit.full, idx)
```

```
##              (Intercept) poly(x, 10, raw = TRUE)1 poly(x, 10, raw = TRUE)2
##               1.07200775               2.38745596               2.84575641
## poly(x, 10, raw = TRUE)3 poly(x, 10, raw = TRUE)5
##               3.55797426               0.08072292
```

*Based on $C_p$, the best model selected by best subset selection is the 4 predictor fit, given by*: $y = 1.07 + 2.39x + 2.85x^2 + 3.56x^3 + 0.08x^5$. *This model seems to have overfit the epsilon a bit, but the values of $\beta_0, \beta_1, \beta_2$, and $\beta_3$ are all pretty close to their true values. Technically, the predicted $\beta_5$ value is close to its true value of 0 as well.*

```
idx = which.max(reg.summary$adjr2)
plot(reg.summary$adjr2, xlab = 'Number of Predictors', ylab = 'Adjusted R^2', col = 'blu
e')
points(idx, reg.summary$adjr2[idx], col = 'red')
```




*Based on adjusted $R^2$, the best model using best subset selection is also the 4 predictor model that was yielded by observing $C_p$.*

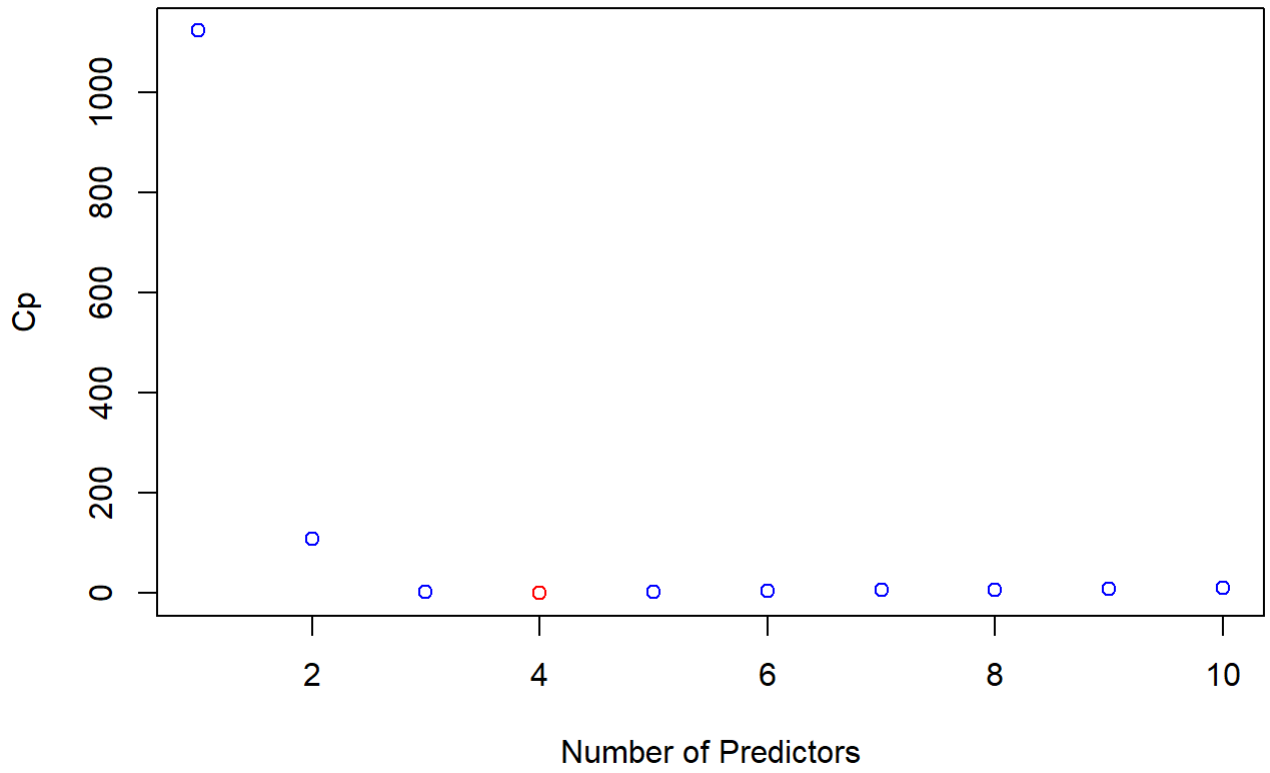
```
coef(regfit.full, 3)
```

```
##              (Intercept) poly(x, 10, raw = TRUE)1 poly(x, 10, raw = TRUE)2
##                 1.061507                 1.975280                 2.876209
## poly(x, 10, raw = TRUE)3
##                 4.017639
```

*Examining the* 3 *predictor model yielded by best subset selection, we can see that it is much closer to the true relationship of* X *and* Y.

d. Repeat (c), using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in (c)?

```
regfit.fwd = regsubsets(y ~ poly(x, 10, raw = TRUE), data = data, nvmax = 10, method =
'forward')
reg.summary = summary(regfit.fwd)
idx = which.min(reg.summary$cp)
plot(reg.summary$cp, xlab = 'Number of Predictors', ylab = 'Cp', col = 'blue')
points(idx, reg.summary$cp[idx], col = 'red')
```

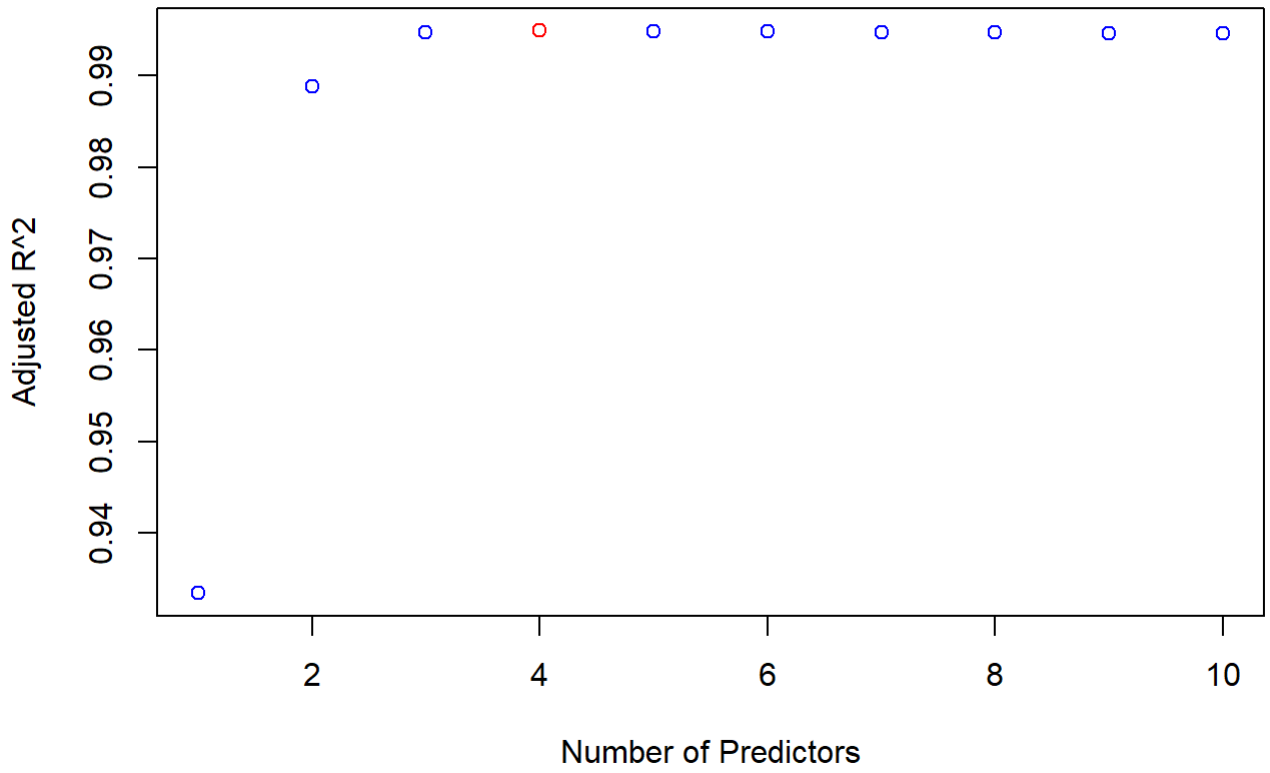



```
coef(regfit.fwd, idx)
```

```
##              (Intercept) poly(x, 10, raw = TRUE)1 poly(x, 10, raw = TRUE)2
##                1.07200775               2.38745596               2.84575641
## poly(x, 10, raw = TRUE)3 poly(x, 10, raw = TRUE)5
##                3.55797426               0.08072292
```

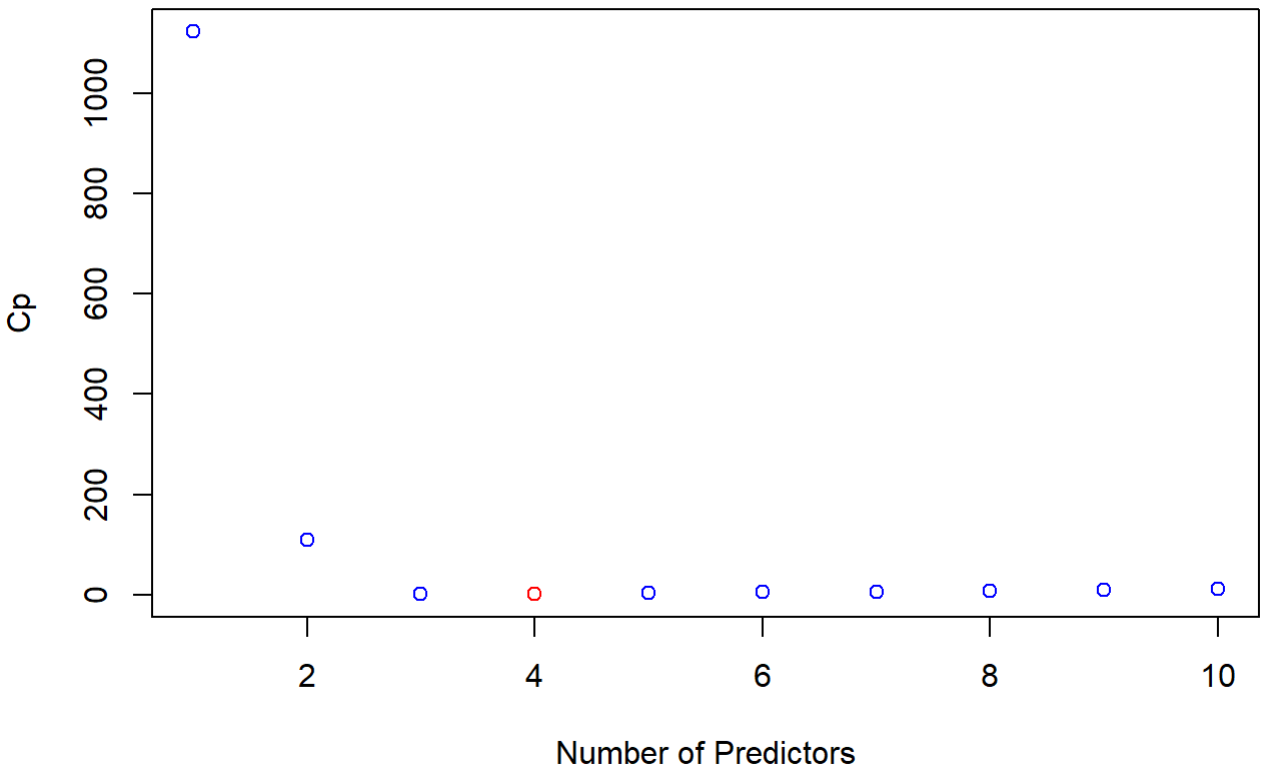
*Interestingly, the forward selection model with the lowest $C_p$ value is the exact same model yielded by the best subset method.*

```
idx = which.max(reg.summary$adjr2)
plot(reg.summary$adjr2, xlab = 'Number of Predictors', ylab = 'Adjusted R^2', col = 'blu
e')
points(idx, reg.summary$adjr2[idx], col = 'red')
```

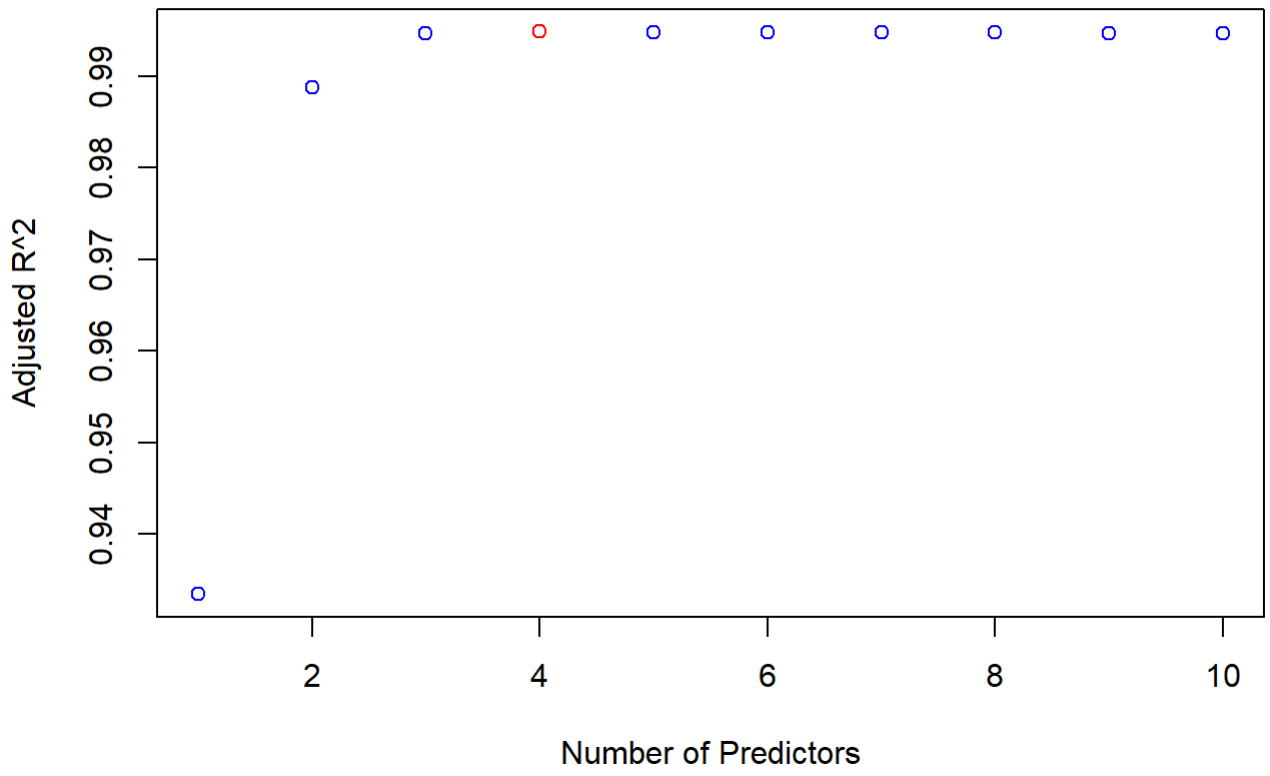



*Again, forward selection yields the exact same model based on adjusted $R^2$.*

```
regfit.bwd = regsubsets(y ~ poly(x, 10, raw = TRUE), data = data, nvmax = 10, method =
'backward')
reg.summary = summary(regfit.bwd)
idx = which.min(reg.summary$cp)
plot(reg.summary$cp, xlab = 'Number of Predictors', ylab = 'Cp', col = 'blue')
points(idx, reg.summary$cp[idx], col = 'red')
```




```
idx = which.max(reg.summary$adjr2)
plot(reg.summary$adjr2, xlab = 'Number of Predictors', ylab = 'Adjusted R^2', col = 'blu
e')
points(idx, reg.summary$adjr2[idx], col = 'red')
```




*The backwards stepwise selection also yields the same model when examining both $C_p$ and adjusted $R^2$.*

e. Now fit a lasso model to the simulated data, again using $X, X^2, \ldots, X^{10}$ as predictors. Use cross-validation to select the optimal value of $\lambda$. Create plots of the cross-validation error as a function of $\lambda$. Report the resulting coefficient estimates, and discuss the results obtained.

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.4.4
```

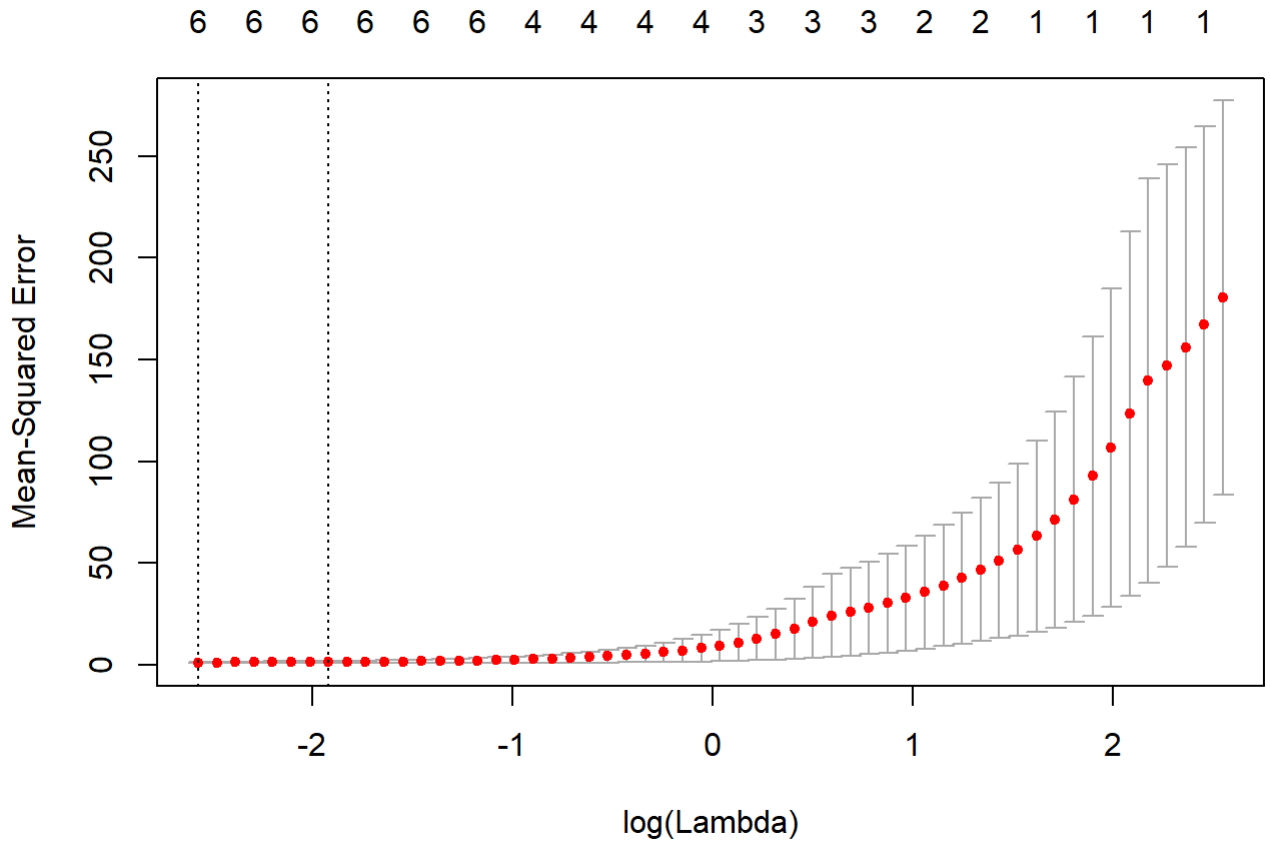
```
## Loading required package: Matrix
```

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

```
## Warning: package 'foreach' was built under R version 3.4.4
```

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

```
set.seed(1)
cv.out = cv.glmnet(poly(x, 10, raw = TRUE), y, alpha = 1)
plot(cv.out)
```




```
bestlam = cv.out$lambda.min
bestlam
```

```
## [1] 0.07660225
```

```
predict(glmnet(poly(x, 10, raw = TRUE), y, alpha = 1), type = 'coefficients', s = bestla
m)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) 1.182646169
## 1           2.137739131
## 2           2.623547995
## 3           3.813195738
## 4           0.042303133
## 5           0.012404464
## 6           .
## 7           0.003849104
## 8           .
## 9           .
## 10          .
```

*The cross-validation with the lasso yields the following sparse model with* 6 *predictors:* $y = 1.183 + 2.138x + 2.624x^2 + 3.813x^3 + 0.042x^4 + 0.012x^5 + 0.004x^7$. *This model is not too far off the true relationship, but it has* 2 *more predictors than the models yielded by subset selection techniques, so it is somewhat less interpretable.*

f. Now generate a response vector *Y* according to the model

$$Y = \beta_0 + \beta_7 X^7 + \epsilon$$

and perform best subset selection and the lasso. Discuss the results obtained.

```
y = 1 + 8 * x^7 + epsilon
data = data.frame(x = x, y = y)
regfit.full = regsubsets(y ~ poly(x, 10, raw = T), data = data, nvmax = 10)
coef(regfit.full, which.max(summary(regfit.full)$adjr2))
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##             1.0762524             0.2914016            -0.1617671
## poly(x, 10, raw = T)3 poly(x, 10, raw = T)7
##            -0.2526527             8.0091338
```

*The true relationship is* $y = 1 + 8x^7$. *Using best subset selection and observing the adjusted* $R^2$ *value to select the best model, we obtain the following* 4 *predictor model:* $y = 1.076 + 0.291x - 0.162x^2 - 0.253x^3 + 8.009x^7$. *This model is pretty close to the true relationship. The coefficients of the* $x^3$, $x^2$, *and* $x$ *predictors are pretty small, while the* $x^7$ *and intercept coefficients are close to the true values.*

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

```
##           (Intercept) poly(x, 10, raw = T)7
##             0.9589402             8.0007705
```

*Like before, we see that the model with the* correct *number of predictors fit by best subset selection is very close to the true relationship.*

```
set.seed(1)
bestlam = cv.glmnet(poly(x, 10, raw = T), y, alpha = 1)$lambda.min
predict(glmnet(poly(x, 10, raw = T), y, alpha = 1), type = 'coefficients', s = bestlam)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     1
## (Intercept) 2.039208
## 1           .
## 2           .
## 3           .
## 4           .
## 5           .
## 6           .
## 7           7.744804
## 8           .
## 9           .
## 10          .
```

*Using* 10 *fold cross-validation to select a lambda value for the lasso, we obtain the following* 1 *predictor model:* $y = 2.039 + 7.745x^7$. *This model does not have coefficients that are as close to the true relationship as the best subset model did, but it does have the only one predictor, which drastically improves interpretability.*

---

# Exercise 9

In this exercise, we will predict the number of applications received using the other variables in the **College** data set.

a. Split the data set into a training set and a test set.

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.4.4
```

```
set.seed(1)
train = sample(c(F,T), dim(College)[1], replace = T)
test = !train
```

b. Fit a linear model using least squares on the training set, and report the test error obtained.

```
lm.fit = lm(Apps ~ ., data = College, subset = train)
lm.pred = predict(lm.fit, College[test,], type = 'response')
lm.mse = mean((College[test, 'Apps'] - lm.pred)^2)
lm.mse
```

```
## [1] 1383033
```

*The least squares model has a test MSE of* 1,383,033.

c. Fit a ridge regression model on the training set, with $\lambda$ chosen by cross-validation. Report the test error obtained.

```
x = model.matrix(Apps ~ ., College)
y = College[,'Apps']
set.seed(1)
bestlambda = cv.glmnet(x[train,], y[train], alpha = 0)$lambda.min
ridge.fit = glmnet(x[train,], y[train], alpha = 0)
ridge.pred = predict(ridge.fit, s = bestlambda, newx = x[test,])
ridge.mse = mean((y[test] - ridge.pred)^2)
ridge.mse
```

```
## [1] 1152487
```

```
(ridge.mse - lm.mse) * 100 / lm.mse
```

```
## [1] -16.66964
```

*The ridge regression model has a test MSE of* 1,152,487, *which is* 16.67% *less than the least squares model.*

d. Fit a lasso model on the training set, with $\lambda$ chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```
set.seed(1)
bestlambda = cv.glmnet(x[train,], y[train], alpha = 1)$lambda.min
lasso.fit = glmnet(x[train,], y[train], alpha = 1)
lasso.pred = predict(lasso.fit, s = bestlambda, newx = x[test,])
lasso.mse = mean((y[test] - lasso.pred)^2)
lasso.mse
```

```
## [1] 1329293
```

```
(lasso.mse - lm.mse) * 100 / lm.mse
```

```
## [1] -3.885668
```

*The lasso model has a test MSE of* 1,329,293, *which is* 3.89% *less than the least squares model.*

```
predict(lasso.fit, s = bestlambda, type = 'coefficients')
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -536.28301112
## (Intercept)    .
## PrivateYes  -365.26347117
## Accept         1.65900471
## Enroll        -0.44569793
## Top10perc     30.32506600
## Top25perc      .
## F.Undergrad    .
## P.Undergrad    .
## Outstate      -0.10313991
## Room.Board     0.14995025
## Books          .
## Personal      -0.06607212
## PhD           -4.98394219
## Terminal      -8.06160448
## S.F.Ratio     12.58019177
## perc.alumni   -0.97154708
## Expend         0.09491440
## Grad.Rate      6.14442103
```

*The lasso model uses* 13 *coefficients, which is* 4 *less than the total number of predictors.*

e. Fit a PCR model on the training set, with *M* chosen by cross-validation. Report the test error obtained, along with the value of *M* selected by cross-validation.

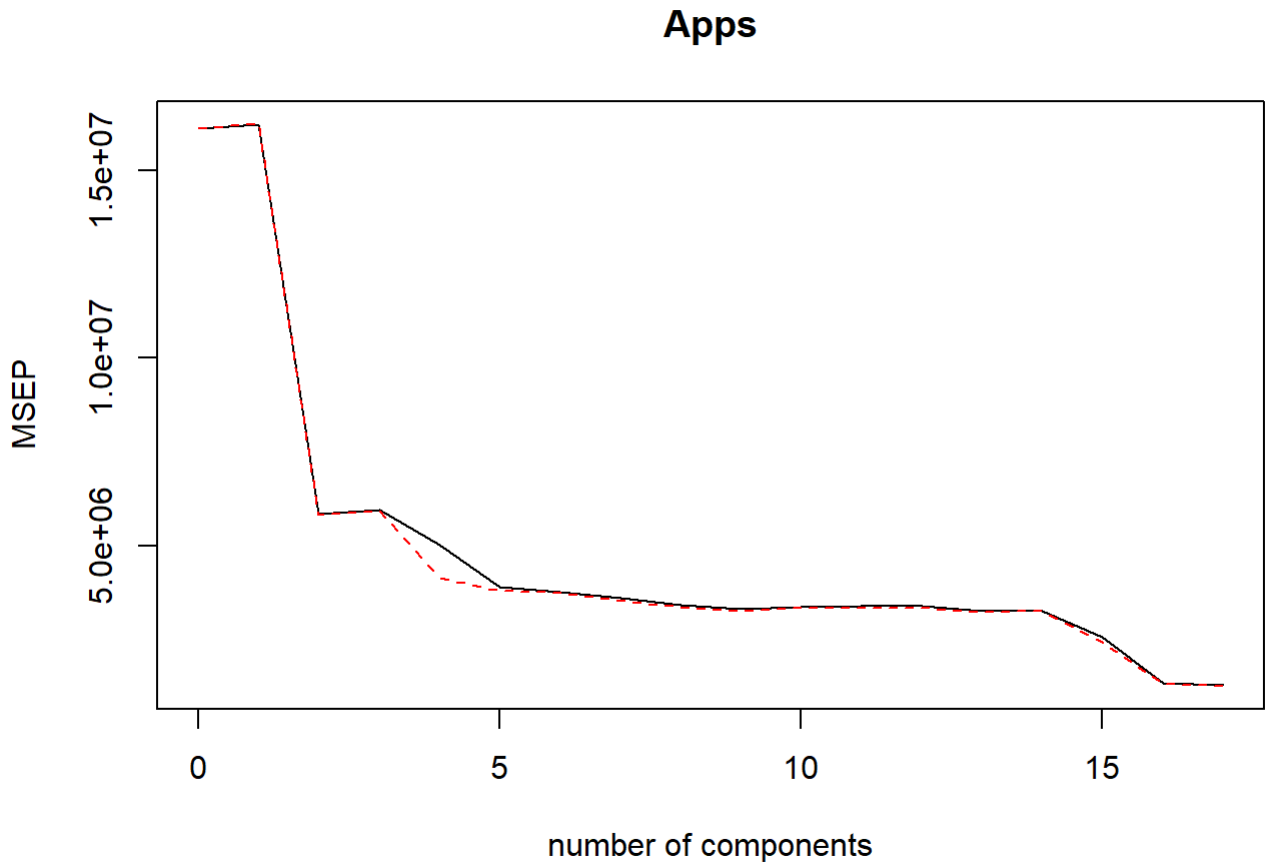
```
library(pls)
```

```
## Warning: package 'pls' was built under R version 3.4.4
```

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

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

```
set.seed(1)
pcr.fit = pcr(Apps ~ ., data = College, subset = train, scale = T, validation = 'CV')
validationplot(pcr.fit, val.type = 'MSEP')
```




```
summary(pcr.fit)
```

```
## Data:    X dimension: 377 17
##  Y dimension: 377 1
## Fit method: svdpc
## 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            4015     4027     2418     2439     2242     1978     1945
## adjCV         4015     4029     2412     2434     2034     1948     1935
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1902     1853     1820      1840      1842      1841      1805
## adjCV     1878     1835     1808      1829      1832      1831      1798
##        14 comps  15 comps  16 comps  17 comps
## CV         1811      1604      1161      1134
## adjCV      1805      1559      1151      1124
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X     31.6744    56.59    64.17    69.82    75.39    80.36    83.90
## Apps   0.1051    66.43    66.43    80.05    80.06    80.12    81.38
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X       87.32    90.25     92.67     94.65     96.49     97.59     98.56
## Apps    82.16    82.77     83.08     83.08     83.24     84.54     84.62
##       15 comps  16 comps  17 comps
## X        99.36     99.90     100.0
## Apps     93.07     94.03      94.2
```

*We can see that the best model chosen by cross-validation uses all* 17 *components, which does not actually yield any dimension reduction.*

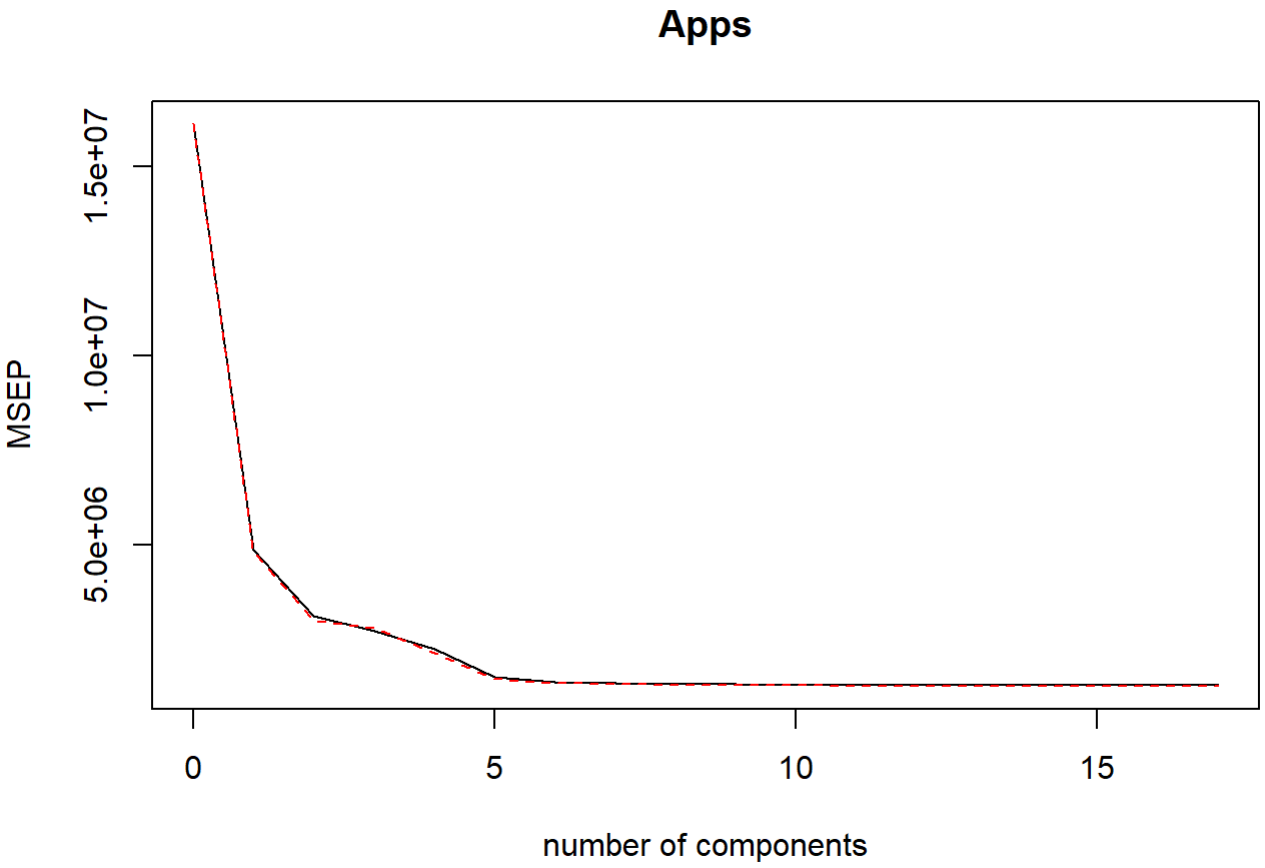
```
pcr.pred = predict(pcr.fit, College[test,], ncomps = 17)
mean((College[test, 'Apps'] - pcr.pred)^2)
```

```
## [1] 2553624
```

*The resulting test MSE is* 2,553,624, *which, surprisingly, is much more than it was using least squares.*

f. Fit a PLS model on the training set, with *M* chosen by cross-validation. Report the test error obtained, along with the value of *M* selected by cross-validation.

```
set.seed(1)
pls.fit = plsr(Apps ~ ., data = College, subset = train, scale = T, validation = 'CV')
validationplot(pls.fit, val.type = 'MSEP')
```




```
summary(pls.fit)
```

```
## Data:    X dimension: 377 17
##  Y dimension: 377 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            4015     2206     1768     1648     1503     1229     1174
## adjCV         4015     2196     1735     1677     1467     1201     1160
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1159     1152     1149      1143      1138      1135      1133
## adjCV     1147     1141     1138      1133      1127      1125      1124
##        14 comps  15 comps  16 comps  17 comps
## CV         1135      1134      1134      1134
## adjCV      1125      1124      1125      1124
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       24.71    30.97    60.46    64.03    66.50    71.40    75.96
## Apps    73.50    86.26    87.27    92.32    93.93    94.02    94.07
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X       78.60    82.08     84.12     86.01     89.66     92.36     94.37
## Apps    94.11    94.13     94.15     94.17     94.18     94.19     94.20
##       15 comps  16 comps  17 comps
## X        97.04     98.02     100.0
## Apps     94.20     94.20      94.2
```

*The best model with the smallest number of dimensions comes from using* 13 *components, reducing the number of dimensions by* 4.

```
pls.pred = predict(pls.fit, College[test,], ncomp = 13)
mean((College[test, 'Apps'] - pls.pred)^2)
```

```
## [1] 1377696
```

*The resulting test MSE is* 1,377,696, *which is a hair less than the least squares test MSE of* 1,383,033.

g. Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

*We saw a decent spread of test MSE from the five approaches. The lowest test MSE came from the ridge regression approach and was* 1,152,487. *The largest test MSE came from the PCR approach and was* 2,553,624 - *almost double. If you only cared about test MSE, ridge regression would probably be the way to go. If model interpretability was important, then the lasso approach might be more appealing. It reduced the problem by the same number of dimensions as the PLS approach while mainting a better test MSE. However, based just on the validation plot for the PLS, it may actually be possible to reduce the number of dimensions to only* 5, *while still maintaining similar predictive performance. This could still be less interpretable than the lasso model with* 13 *predictors, since the PLS uses linear transforms, which can be more difficult to interpret.*

---

# Exercise 10

We have seen that as the number of features used in a model increases, the training error will necessarily decrease, but the test error may not. We will now explore this in a simulated data set.

a. Generate a data set with $p = 20$ features, $n = 1,000$ observations, and an associated quantitative response vector generated according to the model

$$Y = X\beta + \epsilon$$

where $\beta$ has some elements that are exactly equal to zero.

```
p = 20
n = 1000

# Randomly generate the X matrix and the beta array
set.seed(1)
X = matrix(rnorm(n*p, mean = 0, sd = 1), n, p)
beta = sample(1:5, size = p, replace = T)

# Set some beta values to exactly zero
beta[c(3,5,7)] = 0

Y = rep(0, n)
for (i in 1:p) Y = Y + beta[i] * X[,i]

# Add noise to Y
Y = Y + rnorm(n, mean = 0, sd = 1)

beta
```

```
##  [1] 3 3 0 2 0 4 0 5 5 1 2 5 1 2 4 2 5 1 4 5
```

b. Split your data set into a training set containing 100 observations and a test set containing 900 observations.

```
set.seed(1)
train = sample(1:n, size = 100, replace = F)
Y.train = Y[train]
X.train = X[train,]
Y.test = Y[-train]
X.test = X[-train,]
```

c. Perform best subset selection on the training set and plot the training set MSE associated with the best model of each size.

```
# Put X and Y into a dataframe to use with regsubsets
df = data.frame(X)
X.names = rep('', p)
for (i in 1:p) X.names[i] = paste('X', toString(i), sep = "")
colnames(df) = X.names
df$Y = Y
```

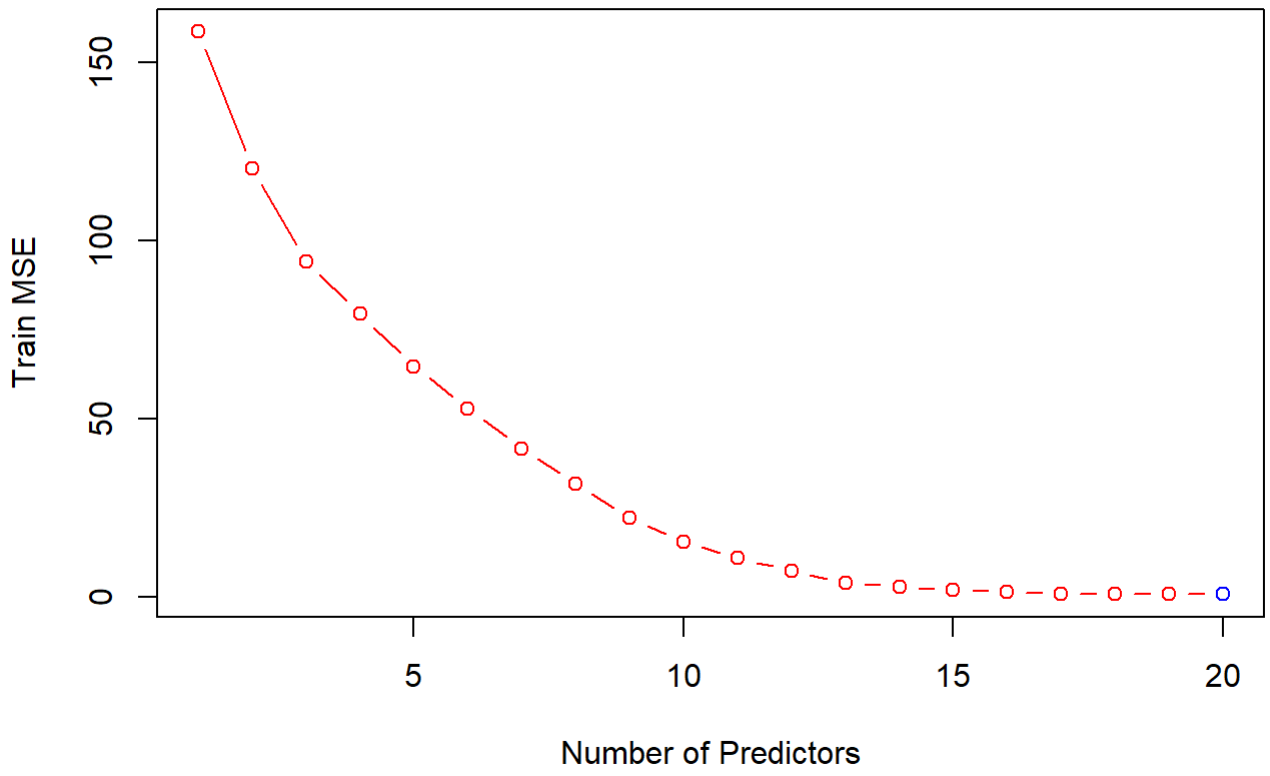
```
set.seed(1)
regfit.best = regsubsets(Y ~ ., data = df[train,], nvmax = p)
summary(regfit.best)
```

```
## Subset selection object
## Call: regsubsets.formula(Y ~ ., data = df[train, ], nvmax = p)
## 20 Variables  (and intercept)
##     Forced in Forced out
## X1      FALSE      FALSE
## X2      FALSE      FALSE
## X3      FALSE      FALSE
## X4      FALSE      FALSE
## X5      FALSE      FALSE
## X6      FALSE      FALSE
## X7      FALSE      FALSE
## X8      FALSE      FALSE
## X9      FALSE      FALSE
## X10     FALSE      FALSE
## X11     FALSE      FALSE
## X12     FALSE      FALSE
## X13     FALSE      FALSE
## X14     FALSE      FALSE
## X15     FALSE      FALSE
## X16     FALSE      FALSE
## X17     FALSE      FALSE
## X18     FALSE      FALSE
## X19     FALSE      FALSE
## X20     FALSE      FALSE
## 1 subsets of each size up to 20
## Selection Algorithm: exhaustive
##           X1  X2  X3  X4  X5  X6  X7  X8  X9  X10 X11 X12 X13 X14 X15 X16
## 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 ) "*" "*" "*" "*" " " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
## 20  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
##           X17 X18 X19 X20
## 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 ) "*" "*" "*" "*"
## 20  ( 1 ) "*" "*" "*" "*"
```

```
# Create function for predicting MSE from subset models
predict.MSE = function(object, newdata, id, ...) {
  form = as.formula(object$call[[2]])
  mat = model.matrix(form, newdata)
  coefi = coef(object, id = id)
  xvars = names(coefi)
  pred = mat[,xvars]%*%coefi
  mse = mean((newdata[,'Y'] - pred)^2)
  return(mse)
}
```

```
train.MSE = rep(0, p)
for (i in 1:p) train.MSE[i] = predict.MSE(regfit.best, df[train,], i)
plot(x = 1:p, y = train.MSE, xlab = 'Number of Predictors', ylab = 'Train MSE', col = 'r
ed', type = 'b')
min.idx = which.min(train.MSE)
points(min.idx, train.MSE[min.idx], col = 'blue')
```




```
train.MSE
```

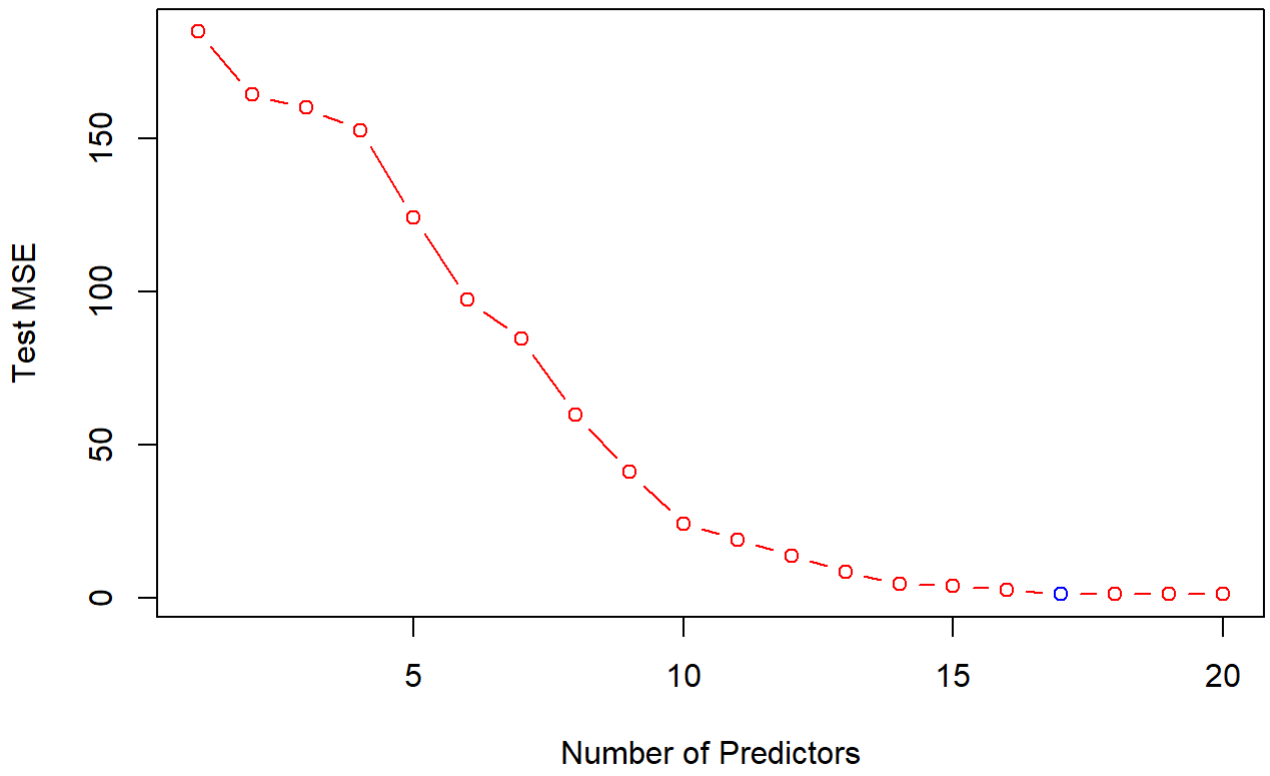
```
##  [1] 158.5500774 120.2631765  94.1827807  79.4017793  64.6386727
##  [6]  52.8078548  41.6278482  31.7267423  22.1224762  15.5087715
## [11]  10.7493021   7.2974079   3.8784549   2.8474477   1.9593517
## [16]   1.4360556   0.7213383   0.7190067   0.7176057   0.7169529
```

```
min.idx
```

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

d. Plot the test set MSE associated with the best model of each size.

```
test.MSE = rep(0, p)
for (i in 1:p) test.MSE[i] = predict.MSE(regfit.best, df[-train,], i)
plot(x = 1:p, y = test.MSE, xlab = 'Number of Predictors', ylab = 'Test MSE', col = 're
d', type = 'b')
min.idx = which.min(test.MSE)
points(min.idx, test.MSE[min.idx], col = 'blue')
```




```
test.MSE
```

```
##  [1] 184.647040 164.180279 159.898218 152.493387 124.020794  97.167947
##  [7]  84.424004  59.700990  40.961639  24.224151  18.907560  13.737290
## [13]   8.486622   4.655916   3.775371   2.707960   1.212911   1.224165
## [19]   1.229481   1.225034
```

```
min.idx
```

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

e. For which model size does the test set MSE take on its minimum value? Comment on your results. If it takes on its minimum value for a model containing only an intercept or a model containing all of the features, then play around with the way you are generating the data in (a) until you come up with a scenario in which the test set MSE is minimized for an intermediate model size.

*The* 17 *predictor model seems to have the best test MSE. This result makes a lot of sense, since we set* 3 *beta values to* 0 *and* 17 *beta values to a non-zero.*

f. How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient values.

```
coef(regfit.best, id = min.idx)
```

```
## (Intercept)          X1          X2          X4          X6          X8
##  0.04541654  3.05234471  3.01846674  2.17815551  4.14289035  4.92019947
##          X9         X10         X11         X12         X13         X14
##  5.09457297  0.90649997  1.81861737  4.96448220  0.92875326  2.15141905
##         X15         X16         X17         X18         X19         X20
##  4.10127468  1.98973394  4.92549756  0.94907196  3.83481076  5.07901129
```

```
beta
```

```
##  [1] 3 3 0 2 0 4 0 5 5 1 2 5 1 2 4 2 5 1 4 5
```

*As expected, X3, X5, and X7 are the predictors which are omitted from the best performing model. These were the three zero-value predictors (i.e. the noise). All of the other predicted coefficients are very close to their true values.*

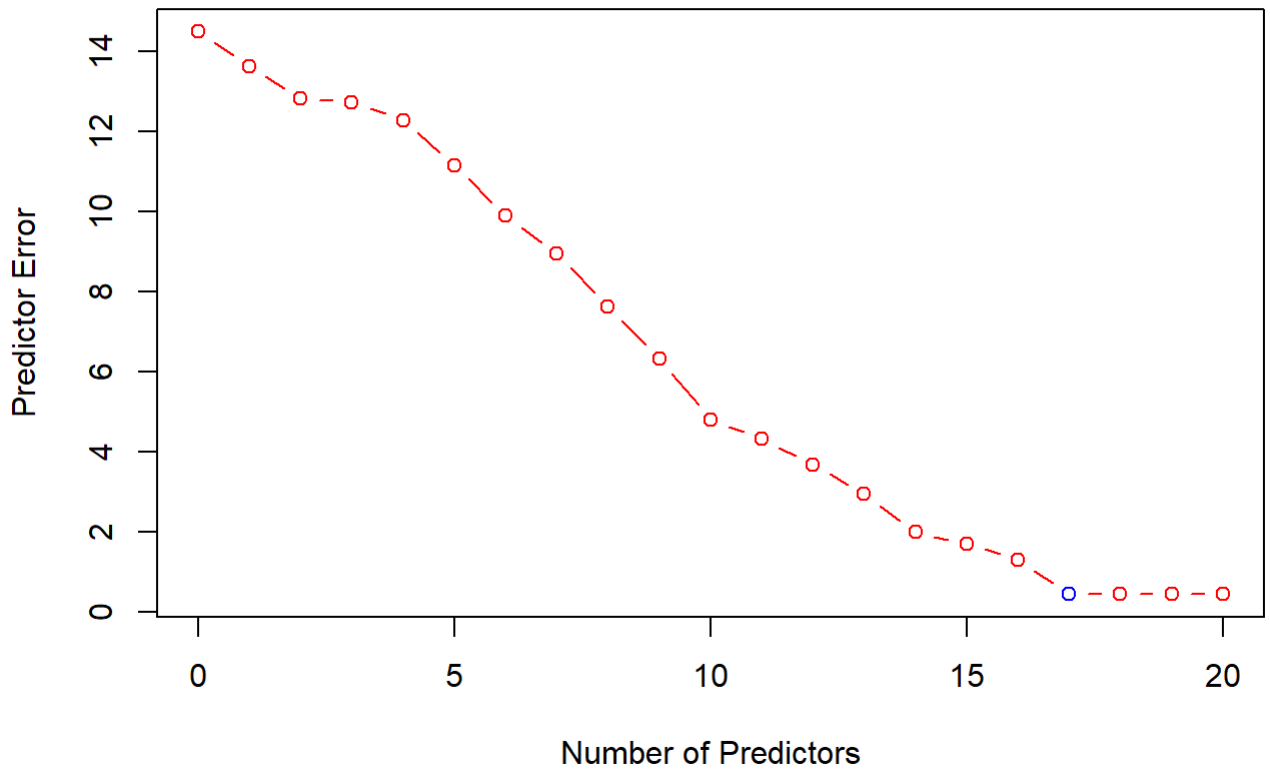
g. Create a plot displaying $\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_j^r)^2}$ for a range of values of *r*, where $\hat{\beta}_j^r$ is the *j*th coefficient estimate for the best model containing *r* coefficients. Comment on what you observe. How does this compare to the test MSE plot from (d)?

```
predictor.error = rep(0, p + 1)

# The 0 predictor model will have a predictor.error equal to the sum of the squares of e
ach beta
predictor.error[1] = sum(beta^2)

for (r in 1:p)
{
  coefi = coef(regfit.best, id = r)
  for (i in 1:p)
  {
    this.error = beta[i]
    col.name = paste('X', toString(i), sep = '')
    if (col.name %in% names(coefi)) this.error = this.error - coefi[col.name]
    predictor.error[r + 1] = predictor.error[r + 1] + this.error^2
  }
}
predictor.error = sqrt(predictor.error)

plot(x = 0:p, y = predictor.error, xlab = 'Number of Predictors', ylab = 'Predictor Erro
r', col = 'red', type = 'b')
min.idx = which.min(predictor.error)
points(min.idx - 1, predictor.error[min.idx], col = 'blue')
```




```
predictor.error
```

```
##  [1] 14.4913767 13.6377423 12.8154821 12.7324352 12.2794685 11.1596199
##  [7]  9.8870229  8.9454802  7.6151795  6.3187484  4.7957359  4.3210618
## [13]  3.6742178  2.9358071  1.9855045  1.7041887  1.2999655  0.4398795
## [19]  0.4483184  0.4514569  0.4498557
```

```
min.idx
```

```
## [1] 18
```

*From the plot, you can see that as we keep adding signal variables, the total error in estimated predictors decreases monotonically. However, as soon as we start to add in noise variables, the total error in estimated predictors begins to increase monotonically.*

---

# Exercise 11

We will now try to predict per capita crime rate in the **Boston** data set.

```
library(MASS)
sum(is.na(Boston))
```

```
## [1] 0
```

*Luckily, the Boston data set has no NaNs in it, so we don't have to worry about that.*

a. Try out some of the regression methods explored in this chapter, such as best subset selection, the lasso, ridge regression, and PCR. Present and discuss results for the approaches that you consider.

*Separate data into training set and test set.*

```
set.seed(1)
train.index = sample(1:nrow(Boston), nrow(Boston)/2)
Boston.train = Boston[train.index,]
Boston.test = Boston[-train.index,]
```

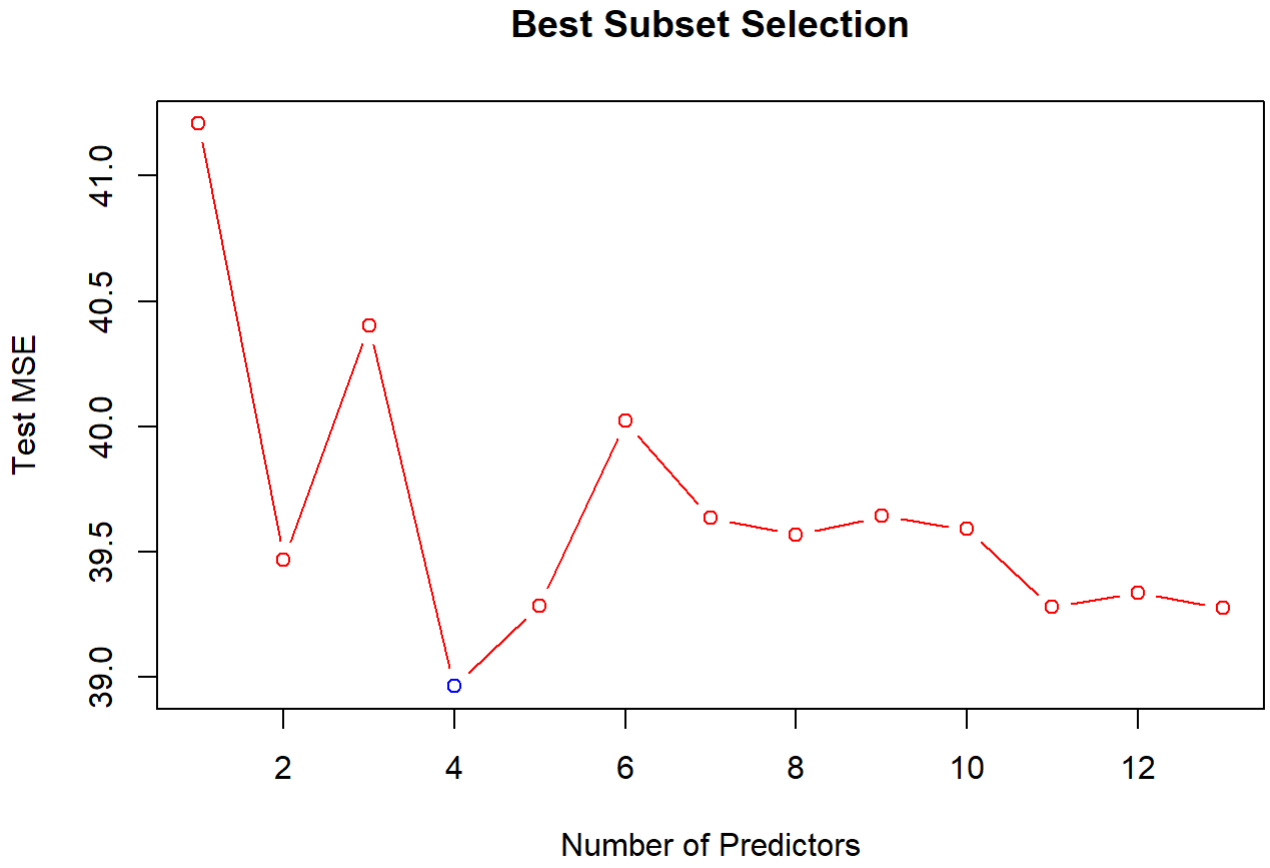
*Try best subset selection.*

```
# Set the maximum number of predictors
p = ncol(Boston) - 1
regfit.best = regsubsets(crim ~ ., data = Boston.train, nvmax = p)

# Need to tweak function from before to point to the 'crim' column
predict.MSE = function(object, newdata, id, ...) {
  form = as.formula(object$call[[2]])
  mat = model.matrix(form, newdata)
  coefi = coef(object, id = id)
  xvars = names(coefi)
  pred = mat[,xvars]%*%coefi
  mse = mean((newdata[,'crim'] - pred)^2)
  return(mse)
}

# Find the model with the lowest test MSE
test.MSE = rep(0, p)
for (i in 1:p) test.MSE[i] = predict.MSE(regfit.best, Boston.test, i)
min.idx = which.min(test.MSE)

# Visually represent the lowest test MSE
plot(1:p, test.MSE, xlab = 'Number of Predictors', ylab = 'Test MSE', main = 'Best Subse
t Selection', col = 'red', type = 'b')
points(min.idx, test.MSE[min.idx], col = 'blue')
```




*We can see that the 4 predictor model has the lowest test MSE of all the models fit via best subset selection.*

```
test.MSE[min.idx]
```

```
## [1] 38.96427
```

```
coef(regfit.best, id = min.idx)
```

```
## (Intercept)          zn         dis         rad        medv
##  6.42991747  0.06592941 -0.97635387  0.47545876 -0.19873222
```

*Using the proportion of residential land zoned (zn), the weighted mean distances to five Boston employment centres (dis), the index of accessibility to radial highways (rad), and the median value of owner-occupied homes (medv), this 4-predictor model has a test MSE of* 38.96427.

*Try the lasso method.*

```
# Create X and Y matrices to use for lasso and ridge regression
Y.train = Boston.train[,'crim']
X.train = model.matrix(crim ~ ., Boston.train)
Y.test = Boston.test[,'crim']
X.test = model.matrix(crim ~ ., Boston.test)

# Find the best lambda for the lasso based on 10-fold cross-validation with the training
data
set.seed(1)
bestlambda = cv.glmnet(X.train, Y.train, alpha = 1)$lambda.min

# Calculate the test MSE based on that lambda
lasso.fit = glmnet(X.train, Y.train, alpha = 1)
lasso.pred = predict(lasso.fit, s = bestlambda, newx = X.test)
lasso.mse = mean((lasso.pred - Y.test)^2)

# Find the model coefficients which yield that test MSE
lasso.coef = predict(lasso.fit, s = bestlambda, type = 'coefficients')

# Print findings
lasso.mse
```

```
## [1] 38.31114
```

```
lasso.coef
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  5.908377637
## (Intercept)  .
## zn           0.035611733
## indus       -0.027463548
## chas        -0.497602440
## nox         -8.530185182
## rm           1.122646635
## age          .
## dis         -0.905307174
## rad          0.508636003
## tax          .
## ptratio     -0.186754581
## black       -0.002438343
## lstat        0.174656099
## medv        -0.181773482
```

*The lasso method yields an* 11*-predictor model with a test MSE of* 38.31114*.*

*Try ridge regression.*

```
# Find the best lambda for ridge regression based on 10-fold cross-validation with the t
raining data
set.seed(1)
bestlambda = cv.glmnet(X.train, Y.train, alpha = 0)$lambda.min

# Calculate the test MSE based on that lambda
ridge.fit = glmnet(X.train, Y.train, alpha = 0)
ridge.pred = predict(ridge.fit, s = bestlambda, newx = X.test)
ridge.mse = mean((ridge.pred - Y.test)^2)

# Find the model coefficients which yield that test MSE
ridge.coef = predict(ridge.fit, s = bestlambda, type = 'coefficients')

# Print findings
ridge.mse
```

```
## [1] 38.36719
```

```
ridge.coef
```

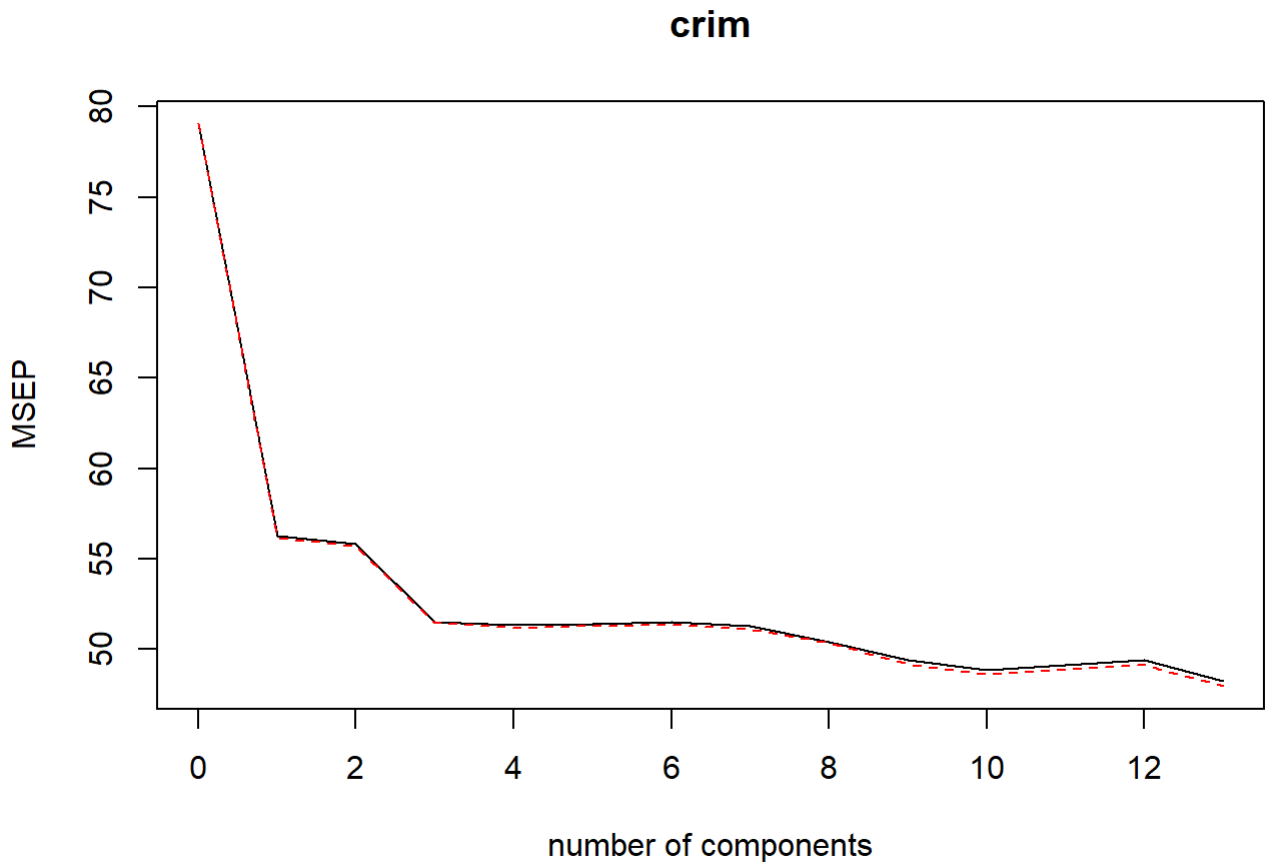
```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  2.861544763
## (Intercept)  .
## zn           0.034393951
## indus       -0.058853509
## chas        -0.785807275
## nox         -6.750896945
## rm           1.108697726
## age          0.005922744
## dis         -0.809673180
## rad          0.398888727
## tax          0.004266196
## ptratio     -0.131627212
## black       -0.004306092
## lstat        0.190997048
## medv        -0.157552605
```

*The ridge regression yeilds a model where 2 of the predictors, age and tax, are very close to zero. This model has a test MSE of* 38.36719.

*Try PCR.*

```
# Determine the best Principal Components Regression model via 10-fold cross-validation
 with the training data
pcr.fit = pcr(crim ~ ., data = Boston.train, scale = T, validation = "CV", segments = 10
)

# Examine the resulting model
validationplot(pcr.fit, val.type = 'MSEP')
```




```
summary(pcr.fit)
```

```
## Data:    X dimension: 253 13
##  Y dimension: 253 1
## Fit method: svdpc
## Number of components considered: 13
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           8.892    7.500    7.470    7.178    7.167    7.168    7.176
## adjCV        8.892    7.494    7.465    7.171    7.152    7.161    7.165
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       7.161    7.099    7.030     6.989     7.009     7.027     6.947
## adjCV    7.148    7.096    7.014     6.972     6.993     7.007     6.925
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       49.04    60.72    69.75    76.49    83.02    88.40    91.73
## crim    30.39    30.93    36.63    37.31    37.35    37.98    38.85
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X       93.77    95.73     97.36     98.62     99.57    100.00
## crim    39.94    41.89     42.73     42.73     43.55     45.48
```

*The best PCR model using all 13 principal components.*

```
# Find the test MSE with the PCR model
pcr.pred = predict(pcr.fit, Boston.test, ncomps = 13)
pcr.mse = mean((pcr.pred - Y.test)^2)
pcr.mse
```

```
## [1] 41.4208
```

*This PCR model has a test MSE of* 41.4208.

b. Propose a model (or set of models) that seem to perform well on this data set, and justify your answer. Make sure that you are evaluating model performance using validation set error, cross-validation, or some other reasonable alternative, as opposed to using training error.

*The best subset model had a test MSE of* 38.96427 *and used* 4 *predictors. The lasso model had a test MSE of* 38.31114 *and used* 11 *predictors. The ridge regression model had a test MSE of* 38.36719. *The PCR model had a test MSE of* 41.4208 *and used* 13 *principal components. The PCR model had the highest test MSE and did not result in any dimension reduction, so it definitely seems like the worst model. The other models all had very close test MSEs. The lasso model had the lowest test MSE, so if we only cared about prediction accuracy, we would probably go with that model. However, the best subset model had a very similar test MSE and uses only* 4 *predictors, so I would choose that model as the best.*

c. Does your chosen model involve all of the features in the data set? Why or why not?

*No, the best subset model uses relatively few predictors. From the results of all of the models, it seems like at least* 2 *or* 3 *of the variables in the Boston set are only noise in terms of predicting the crime rate.*