Linear Model Selection and Regularization in R

Mathematics for Big Data (R Coding Assignment)

Jeremy Williams

April 30, 2018

Contents

Introduction to Statistical Learning (Chapter 6)

Exercise 3 (Answers)

3a

(iv) Steadily decreases:

As we increase s from 0, all β 's increase from 0 to their least square estimate values.

With that said, all training error for 0 β s is maximumized and steadily decreases to the Ordinary Least Square RSS

3b

(ii) Decrease initially, and then eventually start increasing in a U shape:

If $s = 0$, all β s are 0, the model is extremely simple with a high test RSS.

So, if we increase s, *beta* s assume non-zero values where the model starts to fitt well on test data. We can say that the test RSS decreases. This in turns helps the *beta* s approach their full OLS values, start overfitting to the training data and increase the test RSS.

3c

(iii) Steadily increase:

If $s = 0$, the model effectively predicts a constant and has almost no variance.

So, if we increase s, the models includes more β s and their values then start to increase.

When this happens, the values of β s become highly dependent on training data and then increasing the variance value.

3d

(iv) Steadily decrease:

If $s = 0$, the model adequately finds a constant value with it´s prediction far from actual value. This make the bias position tremendous and unrealistic.

When this happens, as s increases, more β s become non-zero with the model continuing to fit training data almost perfectly and thus reducing the present bias position.

3e

(v) Remains constant:

Based on reseach and theoretical information, irreducible error is model independent.

With that said, irrespective of the choice of s, it remains constant and/or homogeneous.

Exercise 5 (Answers)

5a

A general form of Ridge regression optimization is a followed

Minimize:

$\sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \sum_{j=1}^{p} \hat{\beta}_j x_j)^2 + \lambda \sum_{i=1}^{p} \hat{\beta}_i^2$

In this case, $\hat{\beta}_0 = 0$ and $n = p = 2$.

So, the optimization:

Minimize:

$(y_1 - \hat{\beta}_1 x_{11} - \hat{\beta}_2 x_{12})^2 + (y_2 - \hat{\beta}_1 x_{21} - \hat{\beta}_2 x_{22})^2 + \lambda(\hat{\beta}_1^2 + \hat{\beta}_2^2)$

5b

Now given: $x_{11} = x_{12} = x_1$ and $x_{21} = x_{22} = x_2$.

Take the derivatives of above expression with respect to both $\hat{\beta}_1$

and $\hat{\beta}_2$ by setting them equal to zero,

the following is established:

$\hat{\beta^*}_1 = \frac{x_1 y_1 + x_2 y_2 - \hat{\beta^*}_2 (x_1^2 + x_2^2)}{\lambda + x_1^2 + x_2^2}$

and

$\hat{\beta^*}_2 = \frac{x_1 y_1 + x_2 y_2 - \hat{\beta^*}_1 (x_1^2 + x_2^2)}{\lambda + x_1^2 + x_2^2}$

The above shows:

$\hat{\beta^*}_1 = \hat{\beta^*}_2$

5c

According to the Ridge regression,

Minimize:

$(y_1 - \hat{\beta}_1 x_{11} - \hat{\beta}_2 x_{12})^2 + (y_2 - \hat{\beta}_1 x_{21} - \hat{\beta}_2 x_{22})^2 + \lambda(|\hat{\beta}_1| + |\hat{\beta}_2|)$

5d

The following is an interpretation of the solutions for the equation in *5c* above:

If we use the alternate form of Lasso constraints

$|\hat{\beta}_1| + |\hat{\beta}_2| < s$.

The Lasso constraint take the form

$|\hat{\beta}_1| + |\hat{\beta}_2| < s$,

which when is plotted to take the familiar shape of a diamond centered at origin $(0, 0)$.

Now, we consider the squared optimization constraint

$(y_1 - \hat{\beta}_1 x_{11} - \hat{\beta}_2 x_{12})^2 + (y_2 - \hat{\beta}_1 x_{21} - \hat{\beta}_2 x_{22})^2$,

It is best to use the following

$x_{11} = x_{12}$, $x_{21} = x_{22}$, $x_{11} + x_{21} = 0$, $x_{12} + x_{22} = 0$

and

$y_1 + y_2 = 0$

to simplify it to

Minimize:

$2.(y_1 - (\hat{\beta}_1 + \hat{\beta}_2)x_{11})^2$.

Providing the following solution,

$\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_{11}}$.

We now have parallel line to the edge of Lasso-diamond

$\hat{\beta}_1 + \hat{\beta}_2 = s$.

We can say the original Lasso optimization problem are contours of the function

$(y_1 - (\hat{\beta}_1 + \hat{\beta}_2)x_{11})^2$

that touches the Lasso-diamond

$\hat{\beta}_1 + \hat{\beta}_2 = s$.

Therefore, as $\hat{\beta}_1$ and $\hat{\beta}_2$

changes along the line by $\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_{11}}$,

these contours touch the Lasso-diamond edge $\hat{\beta}_1 + \hat{\beta}_2 = s$ at different points.

So, the entire edge $\hat{\beta}_1 + \hat{\beta}_2 = s$ is a possible solution.

We can also say for the opposite Lasso-diamond edge given by

$\hat{\beta}_1 + \hat{\beta}_2 = -s$.

Showing us that the Lasso problem does not have a unique solution.

With that said, the solution is given by two line segments below:

$\hat{\beta}_1 + \hat{\beta}_2 = s; \hat{\beta}_1 \geq 0; \hat{\beta}_2 \geq 0$

and

$\hat{\beta}_1 + \hat{\beta}_2 = -s; \hat{\beta}_1 \leq 0; \hat{\beta}_2 \leq 0$

Exercise 8 (Answers)

8a

Create 100 X and ϵ variables

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

8b

Selecting $\beta_0 = 3$, $\beta_1 = 2$, $\beta_2 = -2$ and $\beta_3 = 0.2$.

```
b0 = 3
b1 = 2
b2 = -2
b3 = 0.2
Y = b0 + b1 * X + b2 * X^2 + b3 * X^3 + er
```

8c

Use *regsubsets* to select best model having polynomial of X of degree 10

```
suppressMessages(suppressWarnings(library(leaps)))
data.fl = data.frame("y" = Y, "x" = X)
mod.fl = regsubsets(y~poly(x, 10, raw=T), data=data.fl, nvmax=10)
mod.sum = summary(mod.fl)

# Locate model size for best Cp, BIC and Adjr2

which.min(mod.sum$cp)
```

```
## [1] 3
```

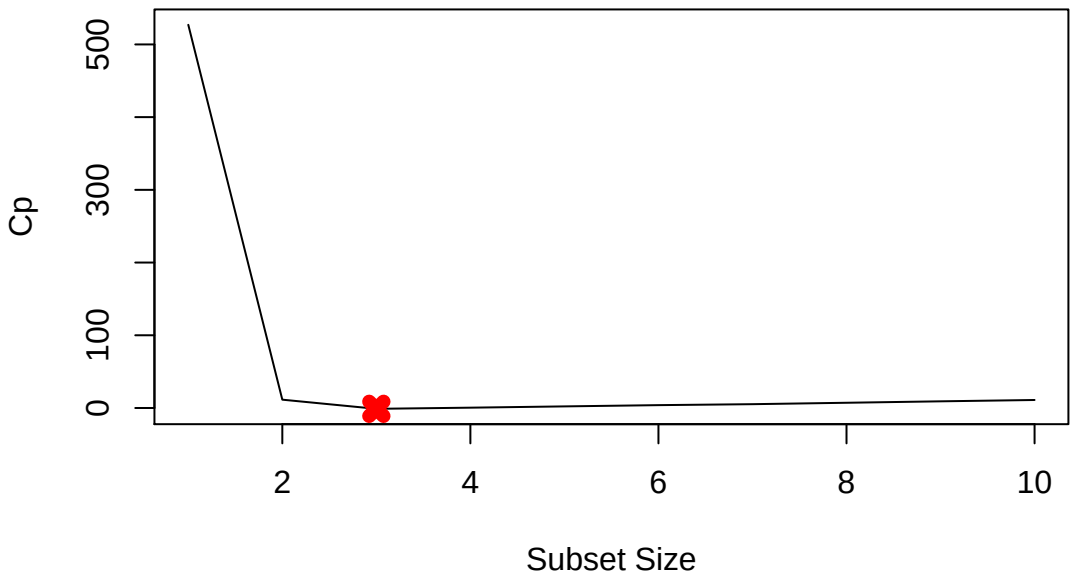
```
which.min(mod.sum$bic)
```

```
## [1] 3
```

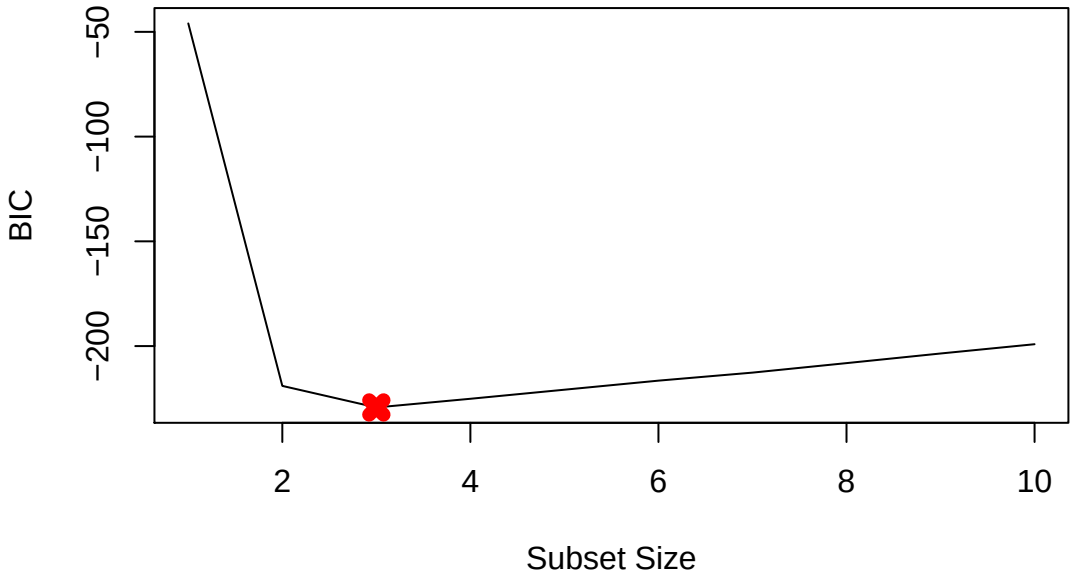
```
which.max(mod.sum$adjr2)
```

```
## [1] 3
```

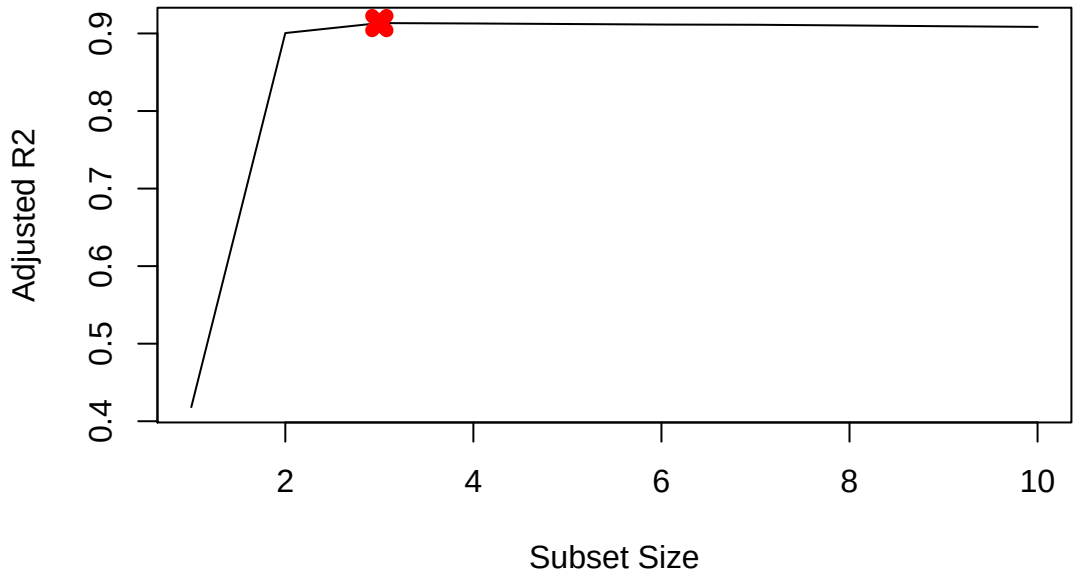
```
# Plot Cp, BIC and Adjr2

par(mfrow=c(1,1))
plot(mod.sum$cp, xlab="Subset Size", ylab="Cp", pch=20, type="l")
points(3, mod.sum$cp[3], pch=4, col="red", lwd=7)
```




```
par(mfrow=c(1,1))
plot(mod.sum$bic, xlab="Subset Size", ylab="BIC", pch=20, type="l")
points(3, mod.sum$bic[3], pch=4, col="red", lwd=7)
```



```
par(mfrow=c(1,1))
plot(mod.sum$adjr2, xlab="Subset Size", ylab="Adjusted R2", pch=20, type="l")
points(3, mod.sum$adjr2[3], pch=4, col="red", lwd=7)
```



Locate with Cp, BIC and Adjusted R2 criteria, 3, 3, and 3 variable models.

```
coefficients(mod.fl, id=3)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           3.076228351           2.219302447          -2.158006460
## poly(x, 10, raw = T)7
##           0.007649528
```

All statistics pick X^7 over X^3.

8d

Fit the forward and backward stepwise models to the data.

```
mod.fwd = regsubsets(y~poly(x, 10, raw=T), data=data.fl, nvmax=10, method="forward")
mod.bwd = regsubsets(y~poly(x, 10, raw=T), data=data.fl, nvmax=10, method="backward")
fwd.sum = summary(mod.fwd)
bwd.sum = summary(mod.bwd)
which.min(fwd.sum$cp)
```

```
## [1] 3
```

```
which.min(bwd.sum$cp)
```

```
## [1] 3
```

```
which.min(fwd.sum$bic)
```

```
## [1] 3
```

```
which.min(bwd.sum$bic)
```

```
## [1] 3
```

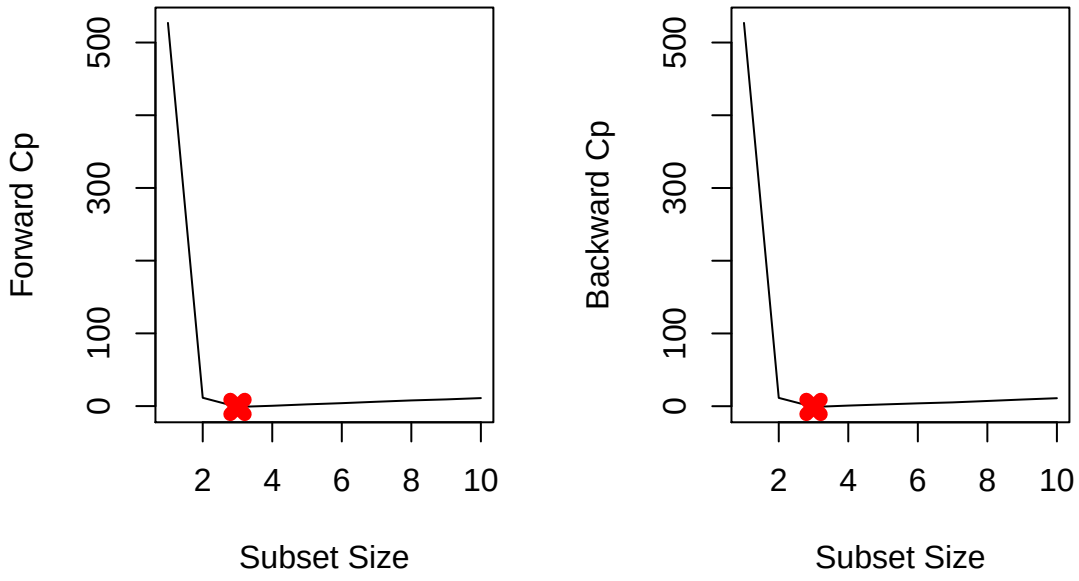
```
which.max(fwd.sum$adjr2)
```

```
## [1] 3
```

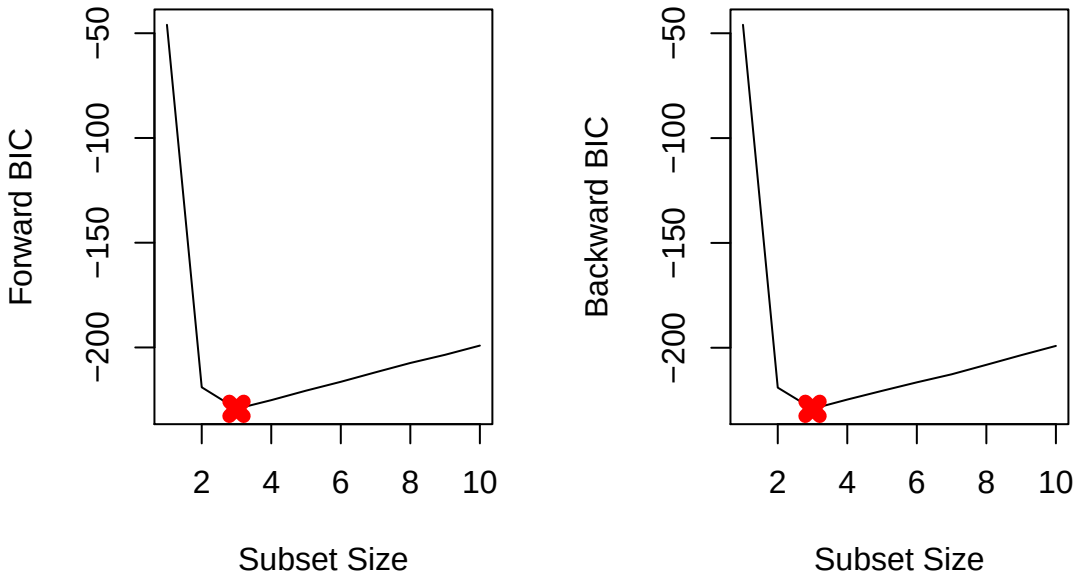
```
which.max(bwd.sum$adjr2)
```

```
## [1] 3
```

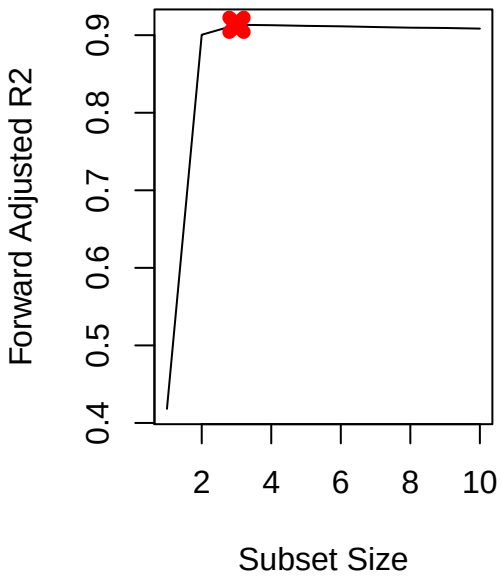
```
# Plot the statistics

par(mfrow=c(1,2))
plot(fwd.sum$cp, xlab="Subset Size", ylab="Forward Cp", pch=20, type="l")
points(3, fwd.sum$cp[3], pch=4, col="red", lwd=7)
plot(bwd.sum$cp, xlab="Subset Size", ylab="Backward Cp", pch=20, type="l")
points(3, bwd.sum$cp[3], pch=4, col="red", lwd=7)
```

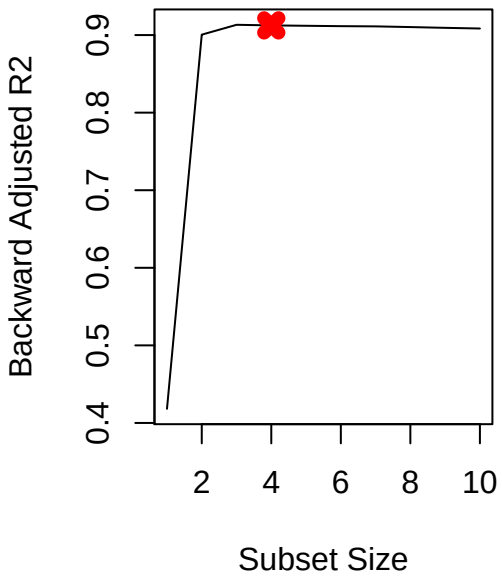



```
par(mfrow=c(1,2))
plot(fwd.sum$bic, xlab="Subset Size", ylab="Forward BIC", pch=20, type="l")
points(3, fwd.sum$bic[3], pch=4, col="red", lwd=7)
plot(bwd.sum$bic, xlab="Subset Size", ylab="Backward BIC", pch=20, type="l")
points(3, bwd.sum$bic[3], pch=4, col="red", lwd=7)
```



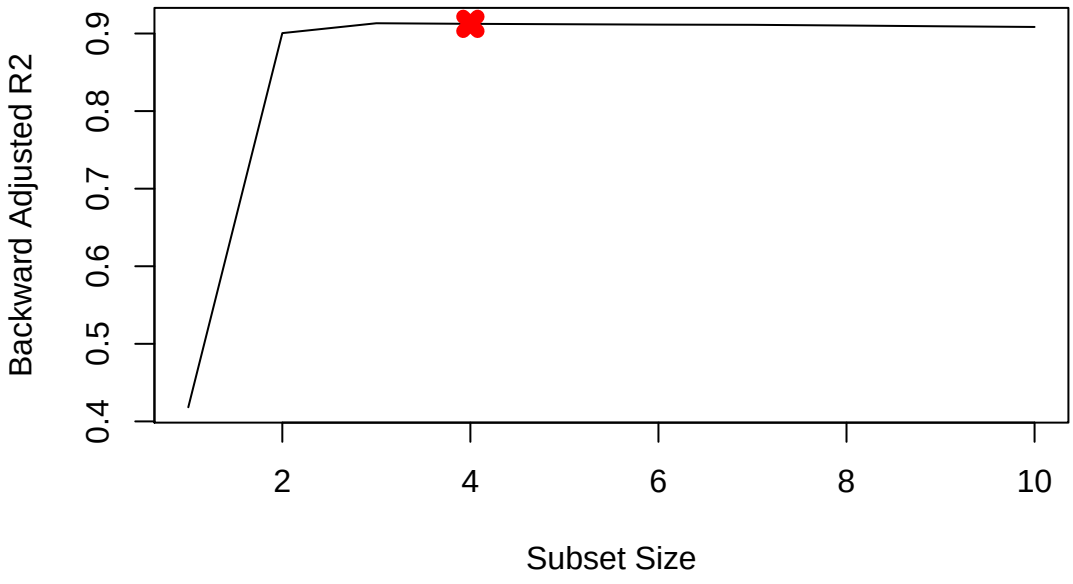

```
par(mfrow=c(1,2))
plot(fwd.sum$adjr2, xlab="Subset Size", ylab="Forward Adjusted R2", pch=20, type="l")
points(3, fwd.sum$adjr2[3], pch=4, col="red", lwd=7)
plot(bwd.sum$adjr2, xlab="Subset Size", ylab="Backward Adjusted R2", pch=20, type="l")
points(4, bwd.sum$adjr2[4], pch=4, col="red", lwd=7)
```





It can be seen that all statistics pick 3 variable models except backward stepwise with adjusted R2.

```
par(mfrow=c(1,1))
plot(bwd.sum$adjr2, xlab="Subset Size", ylab="Backward Adjusted R2", pch=20, type="l")
points(4, bwd.sum$adjr2[4], pch=4, col="red", lwd=7)
```



```
coefficients(mod.fwd, id=4)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           3.113527918           2.239099688          -2.248612840
## poly(x, 10, raw = T)6 poly(x, 10, raw = T)7
##           0.005280229           0.006883952
```

The coefficients values are:

```
coefficients(mod.fwd, id=3)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           3.076228351           2.219302447          -2.158006460
## poly(x, 10, raw = T)7
##           0.007649528
```

```
coefficients(mod.bwd, id=3)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           3.079178293           2.262642715          -2.168260899
## poly(x, 10, raw = T)9
##           0.001385637
```

```
coefficients(mod.fwd, id=4)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##           3.113527918           2.239099688          -2.248612840
## poly(x, 10, raw = T)6 poly(x, 10, raw = T)7
##           0.005280229           0.006883952
```

We can see that the forward stepwise selects the X^7 over X^3 while the Backward stepwise doesn´t. However, all other coefficients are close to β s.

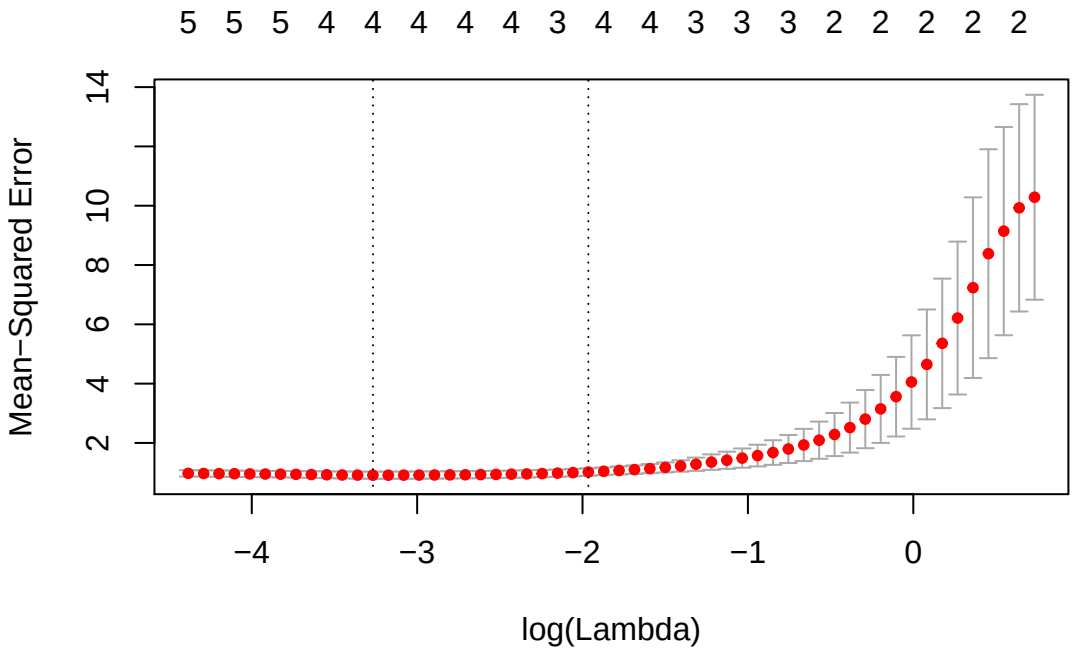
8e

We now conduct a training process Lasso on the data

```
suppressMessages(suppressWarnings(library(glmnet)))
xmat = model.matrix(y~poly(x, 10, raw=T), data=data.fl)[, -1]
mod.lasso = cv.glmnet(xmat, Y, alpha=1)
best.lambda = mod.lasso$lambda.min
best.lambda
```

```
## [1] 0.03810876
```

```
par(mfrow=c(1,1))
plot(mod.lasso)
```




```
# Next fit the model on entire data using best lambda

best.model = glmnet(xmat, Y, alpha=1)
predict(best.model, s=best.lambda, type="coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)             3.043568365
## poly(x, 10, raw = T)1   2.179290718
## poly(x, 10, raw = T)2  -2.109349656
## poly(x, 10, raw = T)3   .
## poly(x, 10, raw = T)4   .
## poly(x, 10, raw = T)5   0.007917102
## poly(x, 10, raw = T)6   .
## poly(x, 10, raw = T)7   0.005538967
## poly(x, 10, raw = T)8   .
## poly(x, 10, raw = T)9   .
## poly(x, 10, raw = T)10  .
```

Lasso also picks X^5 over X^3. It also picks X^7 with negligible coefficient.

8f

Now, let´s make a new Y with $\beta_7 = 7$.

```
b7 = 7
Y = b0 + b7 * X^7 + er

# Predict using regsubsets
```

```
data.fl = data.frame("y" = Y, "x" = X)
mod.fl = regsubsets(y~poly(x, 10, raw=T), data=data.fl, nvmax=10)
mod.sum = summary(mod.fl)

# Find the model size for best Cp, BIC and Adjr2

which.min(mod.sum$cp)
```

```
## [1] 2
```

```
which.min(mod.sum$bic)
```

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

```
which.max(mod.sum$adjr2)
```

```
## [1] 4
```

```
coefficients(mod.fl, id=1)
```

```
##           (Intercept) poly(x, 10, raw = T)7
##               2.95894              7.00077
```

```
coefficients(mod.fl, id=2)
```

```
##           (Intercept) poly(x, 10, raw = T)2 poly(x, 10, raw = T)7
##             3.0704904            -0.1417084             7.0015552
```

```
coefficients(mod.fl, id=4)
```

```
##           (Intercept) poly(x, 10, raw = T)1 poly(x, 10, raw = T)2
##             3.0762524             0.2914016            -0.1617671
## poly(x, 10, raw = T)3 poly(x, 10, raw = T)7
##            -0.2526527             7.0091338
```

We see that BIC picks the most accurate 1-variable model with matching coefficients.

The Other criterias pick different variables.

```
xmat = model.matrix(y~poly(x, 10, raw=T), data=data.fl)[, -1]
mod.lasso = cv.glmnet(xmat, Y, alpha=1)
best.lambda = mod.lasso$lambda.min
best.lambda
```

```
## [1] 13.57478
```

```
best.model = glmnet(xmat, Y, alpha=1)
predict(best.model, s=best.lambda, type="coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                1
## (Intercept)            3.904188
## poly(x, 10, raw = T)1  .
## poly(x, 10, raw = T)2  .
## poly(x, 10, raw = T)3  .
## poly(x, 10, raw = T)4  .
## poly(x, 10, raw = T)5  .
## poly(x, 10, raw = T)6  .
## poly(x, 10, raw = T)7  6.776797
## poly(x, 10, raw = T)8  .
```

```
## poly(x, 10, raw = T)9  .
## poly(x, 10, raw = T)10 .
```

The Lasso also picks the best 1-variable model.

However, we can see that the intercept is quite off (3.9 vs 3).

Exercise 9 (Answers)

9a

First we load and split the College data

```
suppressMessages(suppressWarnings(library(ISLR)))
set.seed(10)
sum(is.na(College))
```

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

```
train.size = dim(College)[1] / 2
train = sample(1:dim(College)[1], train.size)
test = -train
College.train = College[train, ]
College.test = College[test, ]
```

9b

Fit a linear model using the Apps variable.

```
lm.fit = lm(Apps~., data=College.train)
lm.pred = predict(lm.fit, College.test)
mean((College.test[, "Apps"] - lm.pred)^2)
```

```
## [1] 1016996
```

9c

Now pick λ using College.train and report error on College.test

```
library(glmnet)
train.mat = model.matrix(Apps~., data=College.train)
test.mat = model.matrix(Apps~., data=College.test)
grid = 10 ^ seq(4, -2, length=100)
mod.ridge = cv.glmnet(train.mat, College.train[, "Apps"], alpha=0, lambda=grid, thresh=1e-12)
lambda.best = mod.ridge$lambda.min
lambda.best
```

```
## [1] 0.01
```

```
ridge.pred = predict(mod.ridge, newx=test.mat, s=lambda.best)
mean((College.test[, "Apps"] - ridge.pred)^2)
```

```
## [1] 1016980
```

9d

Pick λ using College.train and report error on College.test

```
mod.lasso = cv.glmnet(train.mat, College.train[, "Apps"], alpha=1, lambda=grid, thresh=1e-12)
lambda.best = mod.lasso$lambda.min
lambda.best
```

```
## [1] 24.77076
```

```
lasso.pred = predict(mod.lasso, newx=test.mat, s=lambda.best)
mean((College.test[, "Apps"] - lasso.pred)^2)
```

```
## [1] 925601.4
```

The coefficients are:

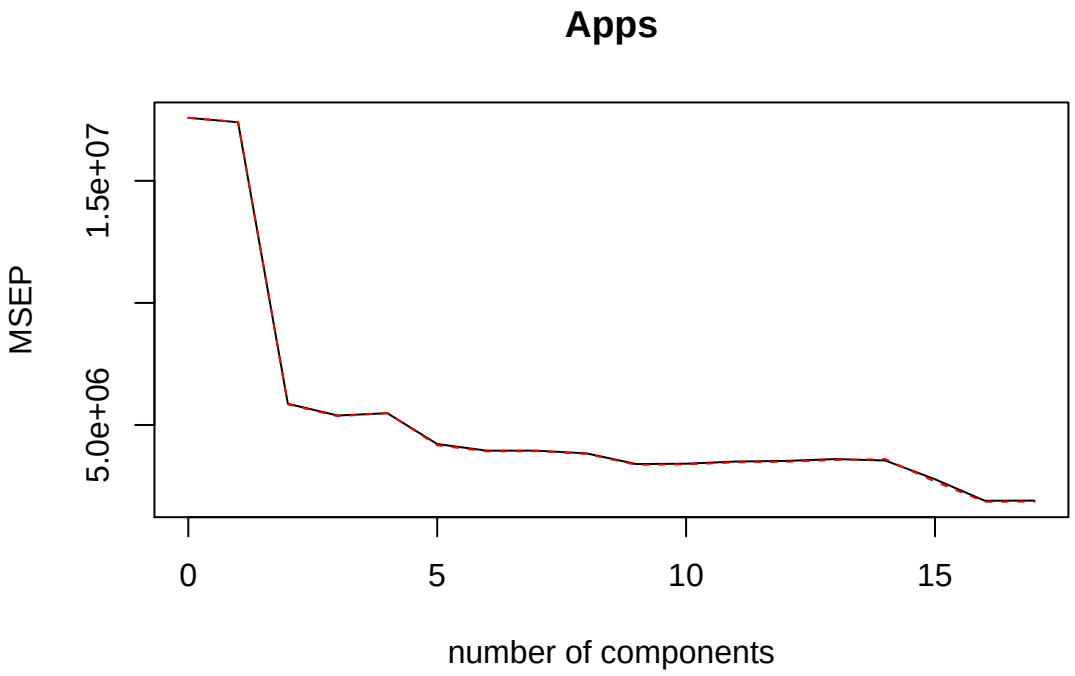
```
mod.lasso = glmnet(model.matrix(Apps~., data=College), College[, "Apps"], alpha=1)
predict(mod.lasso, s=lambda.best, type="coefficients")
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -635.23797264
## (Intercept)    .
## PrivateYes  -408.11931578
## Accept         1.43592777
## Enroll        -0.13872799
## Top10perc     31.34261852
## Top25perc     -0.79389706
## F.Undergrad    .
## P.Undergrad    0.01482400
## Outstate      -0.05320031
## Room.Board     0.12039828
## Books          .
## Personal       .
## PhD           -5.12091278
## Terminal      -3.36566878
## S.F.Ratio      2.73523904
## perc.alumni   -1.05561449
## Expend         0.06836020
## Grad.Rate      4.69112249
```

9g

Now using the validation process to fit pcr

```
suppressMessages(suppressWarnings(library(pls)))
pcr.fit = pcr(Apps~., data=College.train, scale=T, validation="CV")
validationplot(pcr.fit, val.type="MSEP")
```

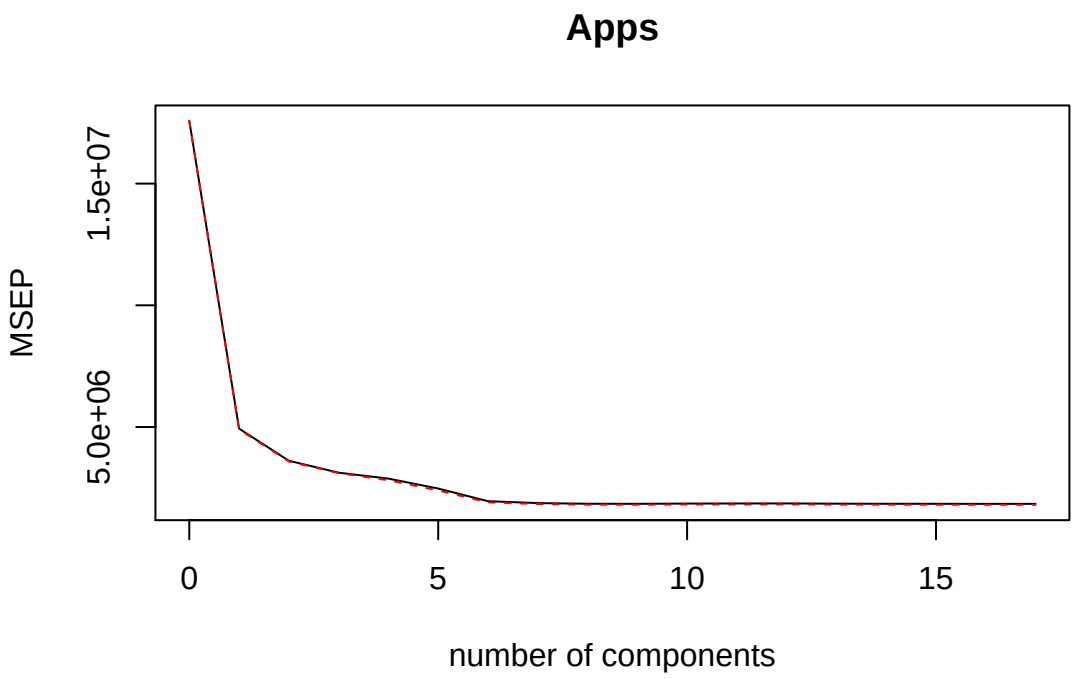



```
pcr.pred = predict(pcr.fit, College.test, ncomp=10)
mean((College.test[, "Apps"] - data.frame(pcr.pred))^2)
```

```
## [1] 1371446
```

Now, we can fit pls by

```
pls.fit = plsr(Apps~., data=College.train, scale=T, validation="CV")
validationplot(pls.fit, val.type="MSEP")
```



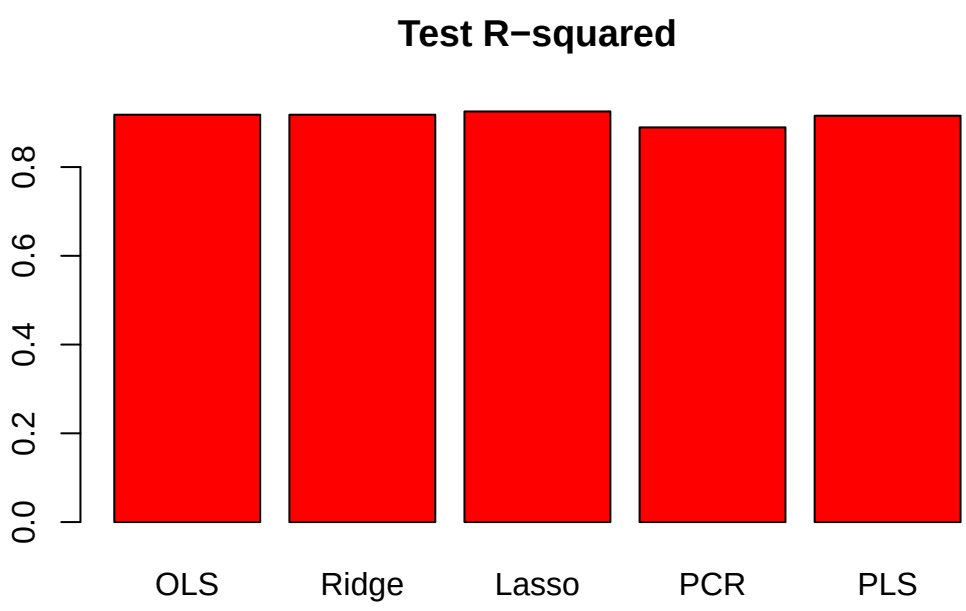

```
pls.pred = predict(pls.fit, College.test, ncomp=10)
mean((College.test[, "Apps"] - data.frame(pls.pred))^2)
```

```
## [1] 1044772
```

Results for OLS, Lasso, Ridge are comparable.

Here are the test R^2 for all models.

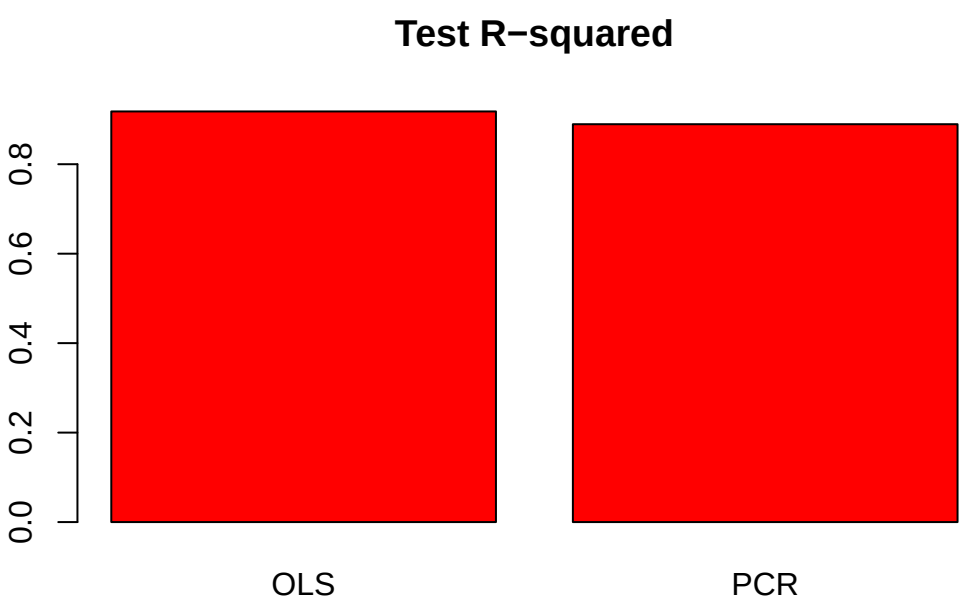
```
test.avg = mean(College.test[, "Apps"])
lm.test.r2 = 1 - mean((College.test[, "Apps"] - lm.pred)^2) /mean((College.test[, "Apps"] - test.avg)^2
ridge.test.r2 = 1 - mean((College.test[, "Apps"] - ridge.pred)^2) /mean((College.test[, "Apps"] - test.a
lasso.test.r2 = 1 - mean((College.test[, "Apps"] - lasso.pred)^2) /mean((College.test[, "Apps"] - test.a
pcr.test.r2 = 1 - mean((College.test[, "Apps"] - data.frame(pcr.pred))^2) /mean((College.test[, "Apps"]
pls.test.r2 = 1 - mean((College.test[, "Apps"] - data.frame(pls.pred))^2) /mean((College.test[, "Apps"]
par(mfrow=c(1,1))
barplot(c(lm.test.r2, ridge.test.r2, lasso.test.r2, pcr.test.r2, pls.test.r2), col="red", names.arg=c("
```



The plot shows that all models except PCR predicts with great accuracy.

See OLS vs PCR plot:

```
par(mfrow=c(1,1))
barplot(c(lm.test.r2,pcr.test.r2), col="red", names.arg=c("OLS","PCR"), main="Test R-squared")
```


Test R−squared
0.0
0.2
0.4
0.6
0.8
OLS
PCR

Exercise 10 (Answers)

10a

Generate a data set with p = 20 features, n = 1,000 observations and β s = 0.

```
set.seed(1)
p = 20
n = 1000
x = matrix(rnorm(n*p), n, p)
B = rnorm(p)
B[3] = 0
B[4] = 0
B[9] = 0
B[19] = 0
B[10] = 0
er = rnorm(p)
y = x %*% B + er
```

10b

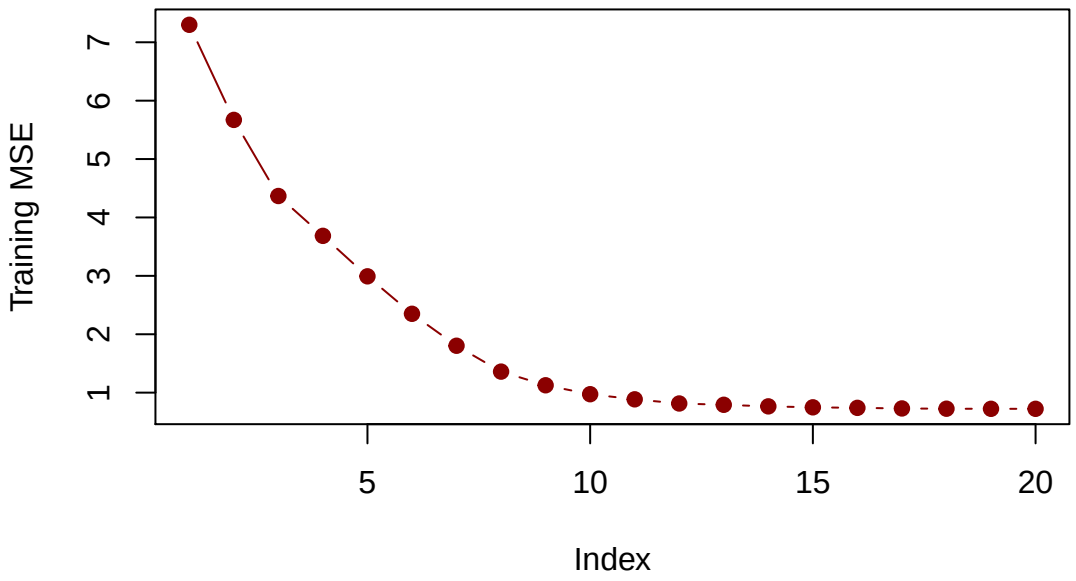
We now split the data set into a training sets.

```
train = sample(seq(1000), 100, replace = FALSE)
y.train = y[train,]
y.test = y[-train,]
x.train = x[train,]
x.test = x[-train,]
```

10c

We now perform best subset selection on the training set and plot the training set MSE.

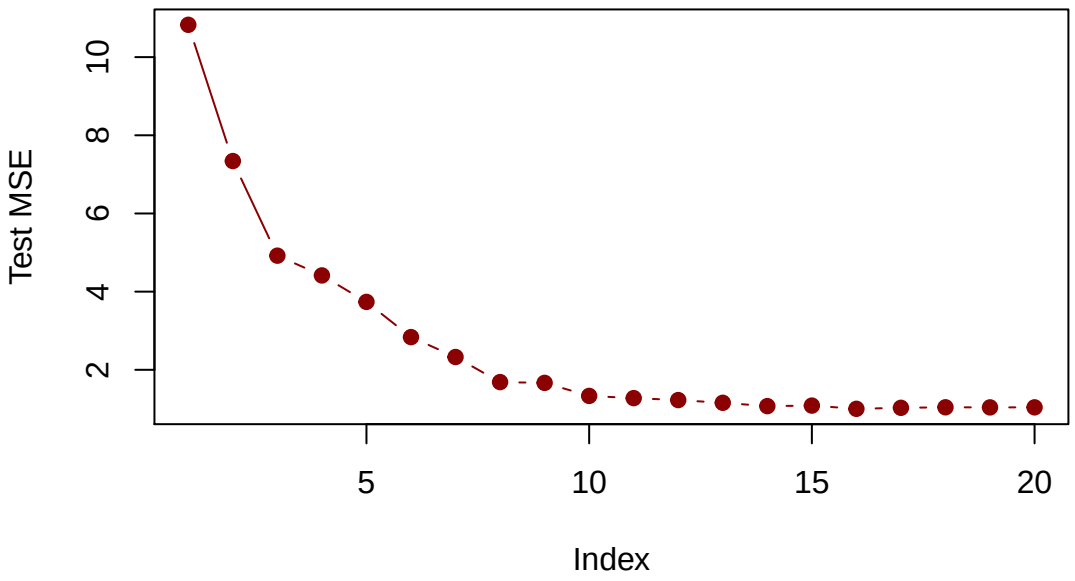
```
suppressMessages(suppressWarnings(library(leaps)))
regfit.full = regsubsets(y~., data=data.frame(x=x.train, y=y.train), nvmax=p)
mse.errors = rep(NA, p)
x_cols = colnames(x, do.NULL=FALSE, prefix="x.")
for (i in 1:p) {
  coefi = coef(regfit.full, id=i)
  pred = as.matrix(x.train[, x_cols %in% names(coefi)]) %*% coefi[names(coefi) %in% x_cols]
  mse.errors[i] = mean((y.train - pred)^2)
}
par(mfrow=c(1,1))
plot(mse.errors, ylab="Training MSE", col = "dark red", pch=19, type="b")
```



10d

Plot the test set MSE.

```
mse.errors = rep(NA, p)
for (i in 1:p) {
  coefi = coef(regfit.full, id=i)
  pred = as.matrix(x.test[, x_cols %in% names(coefi)]) %*% coefi[names(coefi) %in% x_cols]
  mse.errors[i] = mean((y.test - pred)^2)
}
par(mfrow=c(1,1))
plot(mse.errors, ylab="Test MSE", col = "dark red", pch=19, type="b")
```



10e

We now locate which model size does the test set MSE take on its minimum value.

```
which.min(mse.errors)
```

```
## [1] 16
```

We can see 16 parameter model take on this value having the smallest test MSE.

10f

```
coef(regfit.full, id=16)
```

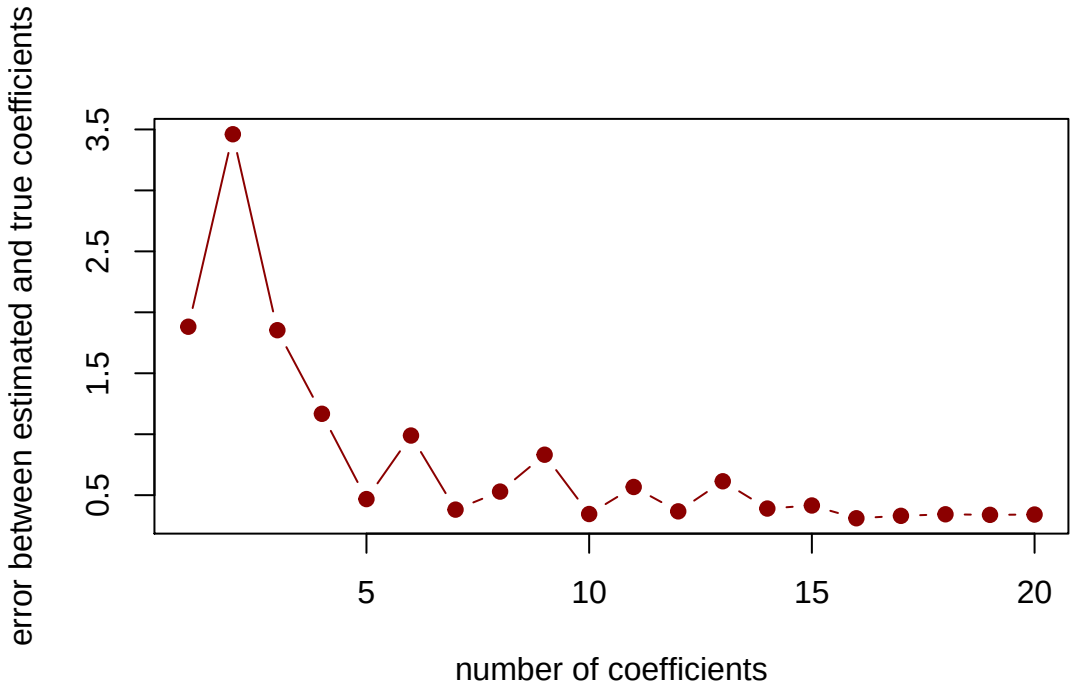
```
## (Intercept)         x.1         x.2         x.5         x.6         x.7
##  0.09613244  0.28256751  0.19385802  0.99994674 -0.28597795 -1.50482273
##         x.8        x.11        x.12        x.13        x.14        x.15
##  0.77817125  0.90815918  0.48477881 -0.19747066 -0.71978955 -0.74282068
##        x.16        x.17        x.18        x.19        x.20
## -0.33900837  0.12234642  1.74270174 -0.12435131 -1.03003019
```

We have caught all but one zeroed out coefficient at x.19.

10g

```
mse.errors = rep(NA, p)
mse1 = rep(NA, p)
mse2 = rep(NA, p)
```

```
for (i in 1:p) {
  coefi = coef(regfit.full, id=i)
  mse1[i] = length(coefi)-1
  mse2[i] = sqrt(
    sum((B[x_cols %in% names(coefi)] - coefi[names(coefi) %in% x_cols])^2) +
    sum(B[!(x_cols %in% names(coefi))])^2)
}
par(mfrow=c(1,1))
plot(x=mse1, y=mse2, xlab="number of coefficients",
     ylab="error between estimated and true coefficients", col = "dark red", pch=19, type="b")
```




```
which.min(mse2)
```

```
## [1] 16
```

According to the model coefficients, the test error is minimized with a 16 parameter model; with a purpose to mininze the error between the estimated and true coefficients.

Further tests and better fit of the true coefficient can be measured in our findings. However, we must be careful not to overstate/understate the model towards having a lower test MSE; which might not effectively become successful.

Reference

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning: With Applications in R. Corrected edition. New York: Springer.