DS502- HW3

Mahdi Alouane and Rahul Pande

1.(10 points) Section 6.8, page 259, question 1

a. When performing best subset selection, the algorithm will consider all the possible models with `k` predictors and choose the one that performs the best based on the training dataset. Hence, the best subset selection approach will provide us with the model resulting in the smallest training RSS.

b. As explained above, the best subset selection approach will consider all the possible models and choose the best among them, so, we would expect it to perform better than the forward and backward stepwise selection even on the testing dataset (smallest test RSS). However, since it is based on the training set, in some cases, we might see the forward or backward stepwise selection outperform the best subset approach on the testing dataset by pure chance.

c.

(i) True

In each step, the forward stepwise selection approach augments the k-variable model by one variable in order to obtain the (k+1)-variable model, so, all the predicors contained in the k-variable model are contained in the (k+1)-variable model.

(ii) True

In each step, the backward stepwise selection removes from the (k+1)-variable model one variable in order to obtain the k-variable model, so, all the predicors contained in the k-variable model are contained in the (k+1)-variable model.

(iii) False

The models obtained by the backward and forward stepwise selection are completely different and are not related, hence, there is no guarantee that the predictors of the k-variable model identified by backward stepwise are a subset of the predictors in the (k + 1)-variable model identified by forward stepwise selection.

(iv) False

The models obtained by the backward and forward stepwise selection are completely different and are not related, hence, there is no guarantee that the predictors of the k-variable model identified by forward stepwise are a subset of the predictors in the (k + 1)-variable model identified by backward stepwise selection.

(v) False

For each `k` as number of predictors, the best subset selection approach will consider all the possible models and choose the best among them. For `k+1` predictors, the model obtained does not necessarily contain all the predictors contained in the k-variable model since it is going to choose the best subset of predictors among all the possible (k+1)-variable models.

2.(10 points) Section 6.8, page 261, question 6

a.

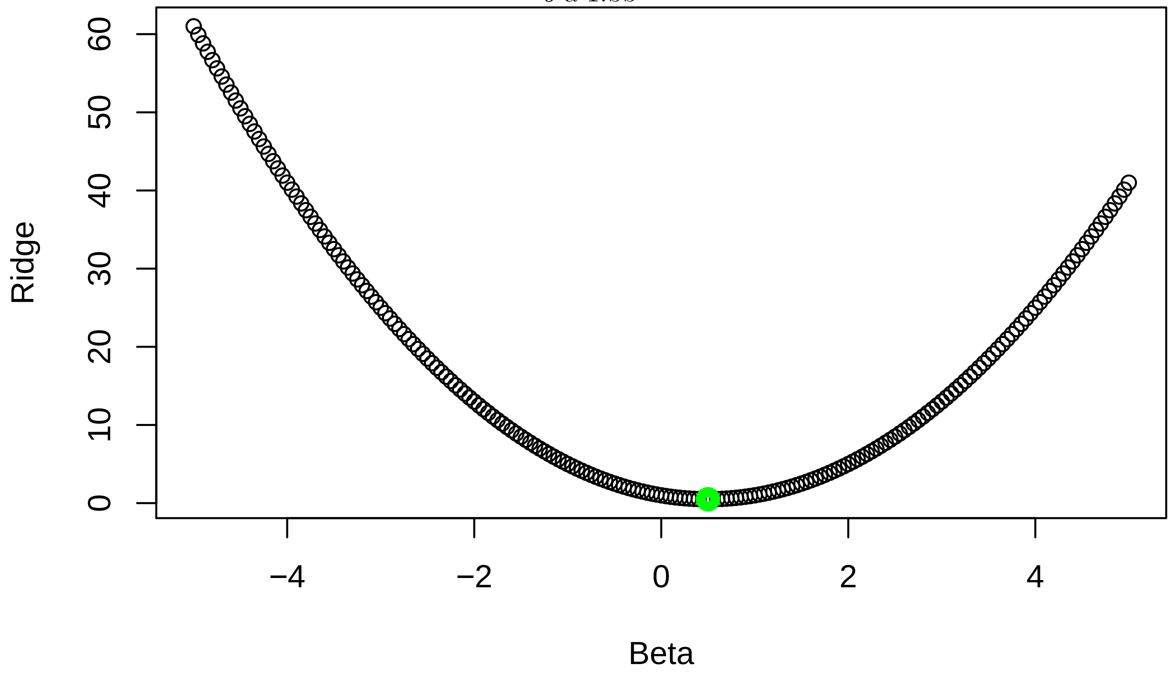
```r
# Initialize values
y = 1
lambda = 1
beta = seq(-5, 5, .05)
```

```r
# Plot the ridge optimization vs beta
plot(beta, (y - beta)^2 + lambda * beta^2, xlab = "Beta", ylab = "Ridge")

# Calculate the ridge regression estimation of Beta (equation 6.14)
betaEst = y / (1 + lambda)
ridgeBetaEst = (y - betaEst)^2 + lambda * betaEst^2

# Add the point of BetaEst to the plot
points(betaEst, ridgeBetaEst, col = "green", lwd = 5)
```

6 a-1.bb



As displayed on the plot, we could observe that the function in (6.12) is minimal at the point (0.5,0.5).

b.

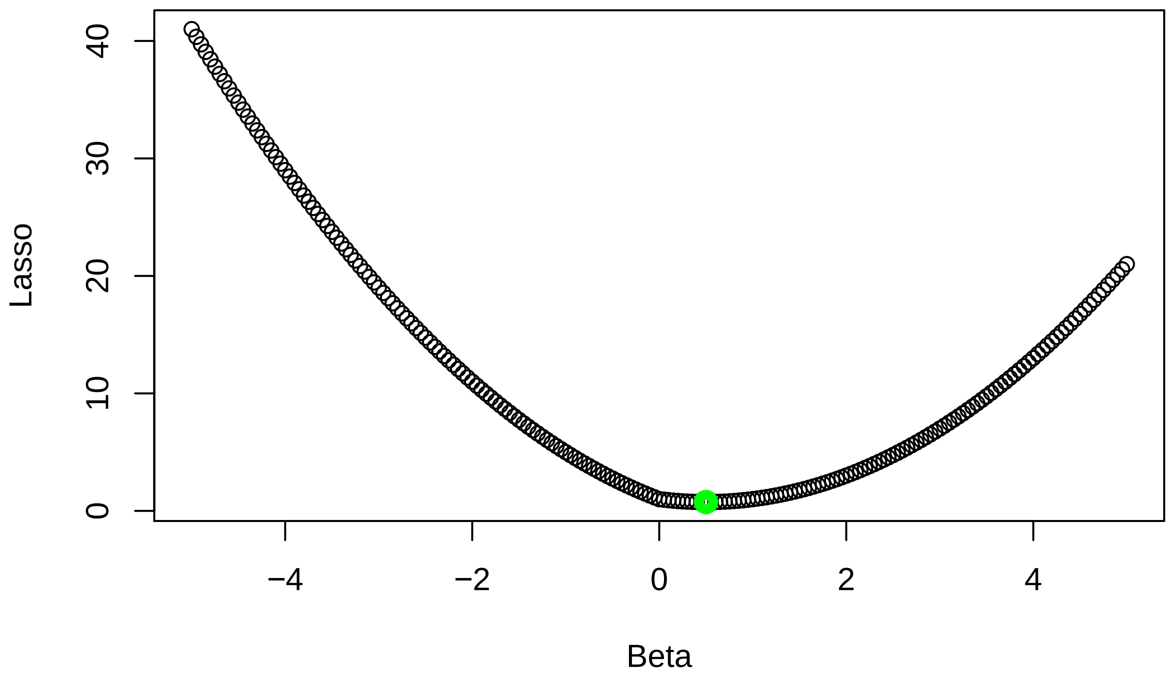
```r
# Initialize values
y = 1
lambda = 1
beta = seq(-5, 5, .05)

# Plot the lasso optimization vs beta
plot(beta, (y - beta)^2 + lambda * abs(beta), xlab = "Beta", ylab = "Lasso")

# Calculate the lasso regression estimation of Beta (equation 6.15)
betaEst = y  - lambda / 2
ridgeBetaEst = (y - betaEst)^2 + lambda * abs(betaEst)

# Add the point of BetaEst to the plot
points(betaEst, ridgeBetaEst, col = "green", lwd = 5)
```

6 b-1.bb



We know that $y > \frac{\lambda}{2}$, then, the lasso estimation is $\beta_e = y - \frac{\lambda}{2}$. As displayed on the plot, we could observe that the function in (6.14) is minimal at the point (0.5,0.75).

3.(15 points) Section 6.8, page 262-263, question 8

```r
set.seed(123)
x = rnorm(100)
noise = rnorm(100)
```

```r
beta0 = 4
beta1 = 0.3
beta2 = 1
beta3 = 3
Y = beta0 * 1 + beta1 * x + beta2 * x^2 + beta3 * x^3 + noise
```

```r
library(leaps)
data_ <- data.frame(y = Y, x = x)
reg.fit <- regsubsets(Y ~ poly(x, 10, raw = TRUE), data = data_, nvmax = 100)

summary(reg.fit)
```

```
## Subset selection object
## Call: regsubsets.formula(Y ~ poly(x, 10, raw = TRUE), data = data_,
##     nvmax = 100)
## 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 ) "*"                      "*"
```

```
names(summary(reg.fit))
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```
summary.as.data.frame <- function(df, vars){
  return(data.frame(lapply(vars, function(x){df[x]})))
}

reg.fit.summary <- summary(reg.fit)

vars <- c("cp", "bic", "adjr2")
plot.data <- summary.as.data.frame(reg.fit.summary, vars)
plot.data$n <- as.numeric(rownames(plot.data))

lapply(plot.data, function(x){which(x == min(x))})
```

```
## $cp
## [1] 2
##
## $bic
## [1] 2
##
## $adjr2
## [1] 1
##
## $n
## [1] 1
```

```
par(mfrow=c(2,2))
lapply(vars, function(m){
  plot(formula(stringr::str_interp("${m} ~ n")), plot.data)
  })
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```

```
coefficients(reg.fit, id = 5)
```

```
##              (Intercept) poly(x, 10, raw = TRUE)1 poly(x, 10, raw = TRUE)2
##              4.000021450              0.396621422              0.857550204
## poly(x, 10, raw = TRUE)3 poly(x, 10, raw = TRUE)7 poly(x, 10, raw = TRUE)9
##              2.749174139              0.055863160             -0.009356419
```

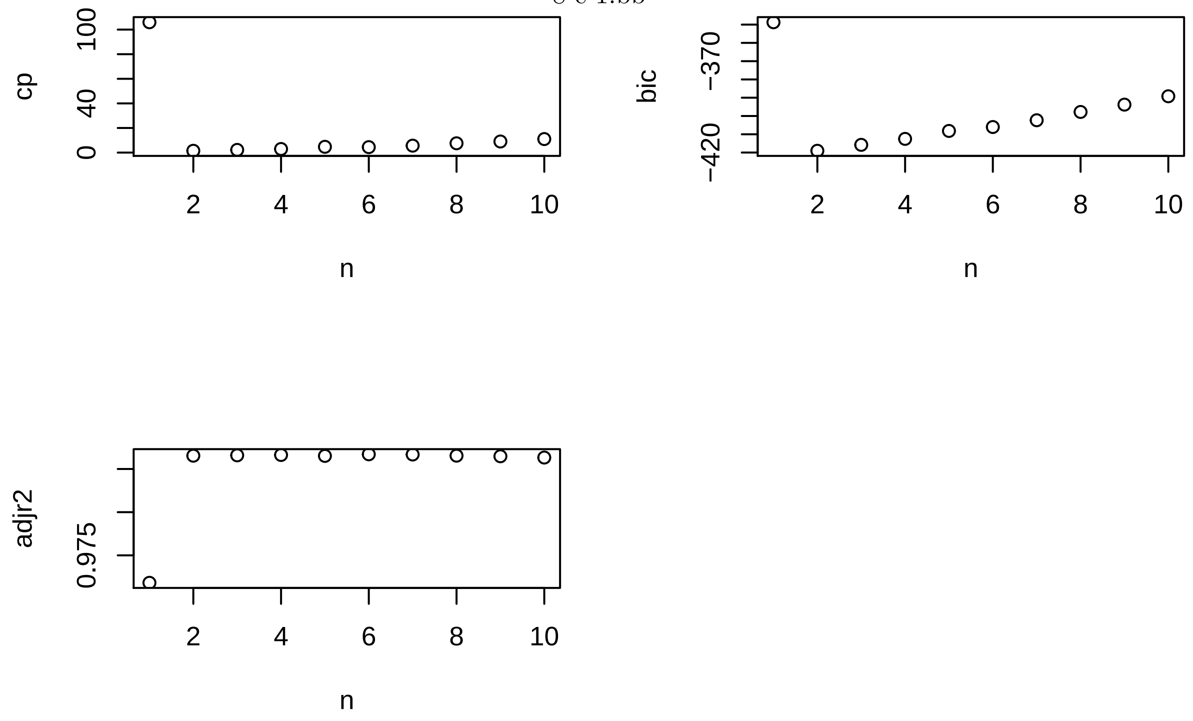
```r
coefficients(reg.fit, id = 2)
```

```
##             (Intercept) poly(x, 10, raw = TRUE)2 poly(x, 10, raw = TRUE)3
##               3.9703537                0.9119294                3.0837330
```

```r
coefficients(reg.fit, id = 1)
```

```
##             (Intercept) poly(x, 10, raw = TRUE)3
##                4.708147                 3.164946
```

8 c-1.bb



We see from bic optimization we get only x3 as best subset (2 coefficient). For cp we have 5 significant coefficients and this appears to be the most close model to the actual function.

```r
## forward selection
reg.fit.forward <- regsubsets(Y ~ poly(x, 10, raw = TRUE), data = data_, nvmax = 100,
                              method = "forward")

reg.fit.forward.summary <- summary(reg.fit.forward)

plot.data <- summary.as.data.frame(reg.fit.forward.summary, vars)
plot.data$n <- as.numeric(rownames(plot.data))
lapply(plot.data, function(x){which(x == min(x))})
```

```
## $cp
## [1] 2
##
## $bic
## [1] 2
##
## $adjr2
## [1] 1
##
```

```
## $n
## [1] 1
```

```
par(mfrow=c(2,2))
lapply(vars, function(m){
  plot(formula(stringr::str_interp("${m} ~ n")), plot.data)
  })
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```

```
coefficients(reg.fit.forward, id = 3)
```

```
##             (Intercept) poly(x, 10, raw = TRUE)1 poly(x, 10, raw = TRUE)2
##               3.9703939                0.2204462                0.9084569
## poly(x, 10, raw = TRUE)3
##               3.0204363
```

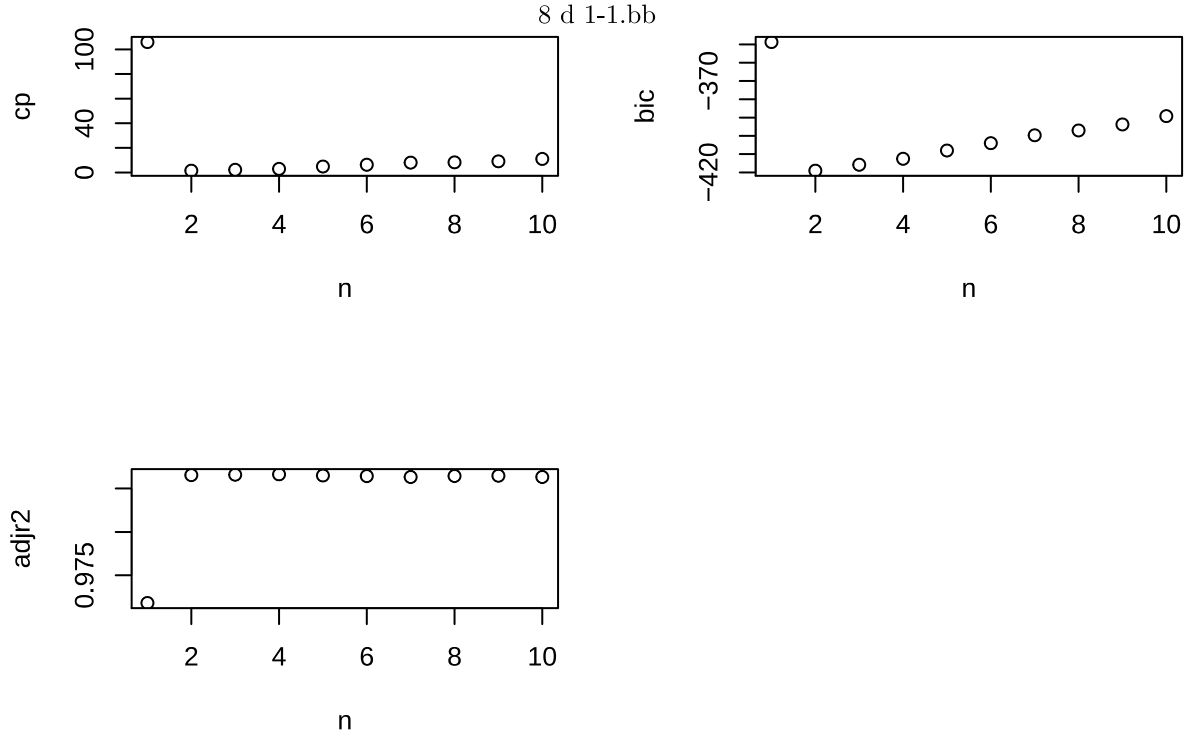
```
coefficients(reg.fit.forward, id = 2)
```

```
##             (Intercept) poly(x, 10, raw = TRUE)2 poly(x, 10, raw = TRUE)3
##               3.9703537                0.9119294                3.0837330
```

```
coefficients(reg.fit.forward, id = 1)
```

```
##             (Intercept) poly(x, 10, raw = TRUE)3
##                4.708147                 3.164946
```

8 d 1-1.bb



We see that with forward selection with cp (3 feature subset) we have a close approximation of the actual function. With bic, beta1, which is the smallest coefficient, is suppressed.

```
## backward selection
reg.fit.backward <- regsubsets(Y ~ poly(x, 10, raw = TRUE), data = data_, nvmax = 100,
                              method = "backward")

reg.fit.backward.summary <- summary(reg.fit.backward)

plot.data <- summary.as.data.frame(reg.fit.backward.summary, vars)
plot.data$n <- as.numeric(rownames(plot.data))

lapply(plot.data, function(x){which(x == min(x))})
```

```
## $cp
## [1] 2
##
## $bic
## [1] 2
##
## $adjr2
## [1] 1
##
## $n
## [1] 1
```

```
par(mfrow=c(2,2))
lapply(vars, function(m){
  plot(formula(stringr::str_interp("${m} ~ n")), plot.data)
  })
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```

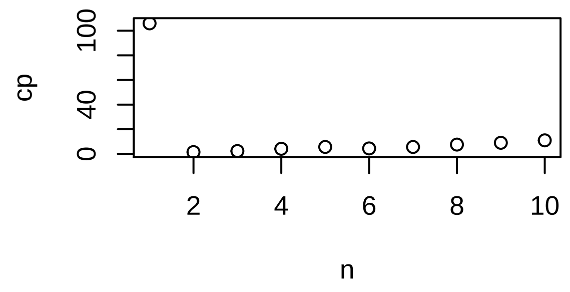
```
coefficients(reg.fit.backward, id = 5)
```

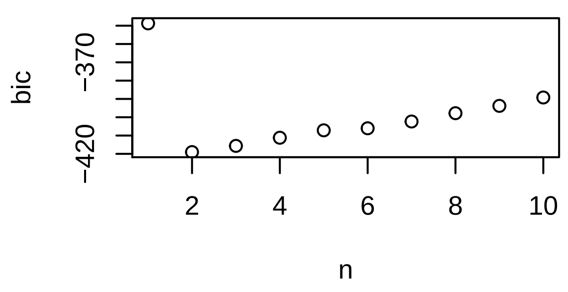
```
##               (Intercept) poly(x, 10, raw = TRUE)2 poly(x, 10, raw = TRUE)3
##                3.94238785               1.47105455               3.08057008
## poly(x, 10, raw = TRUE)4 poly(x, 10, raw = TRUE)6 poly(x, 10, raw = TRUE)8
##               -0.75906400               0.25632444              -0.02459802
```

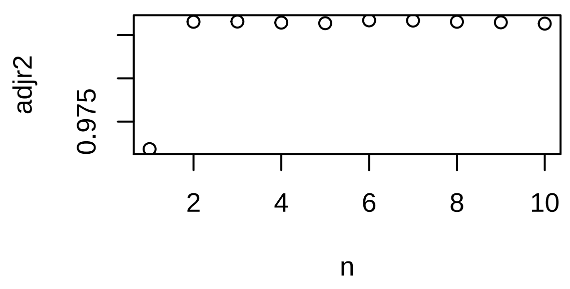
```
coefficients(reg.fit.backward, id = 1)
```

```
##               (Intercept) poly(x, 10, raw = TRUE)3
##                  4.708147                 3.164946
```

8 d 2-1.bb







With backward selection we have 5 feature set as the best set with bic and cp criteria. With adjr2 we only have one, which is incorrect.

```
library(glmnet)
```

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

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

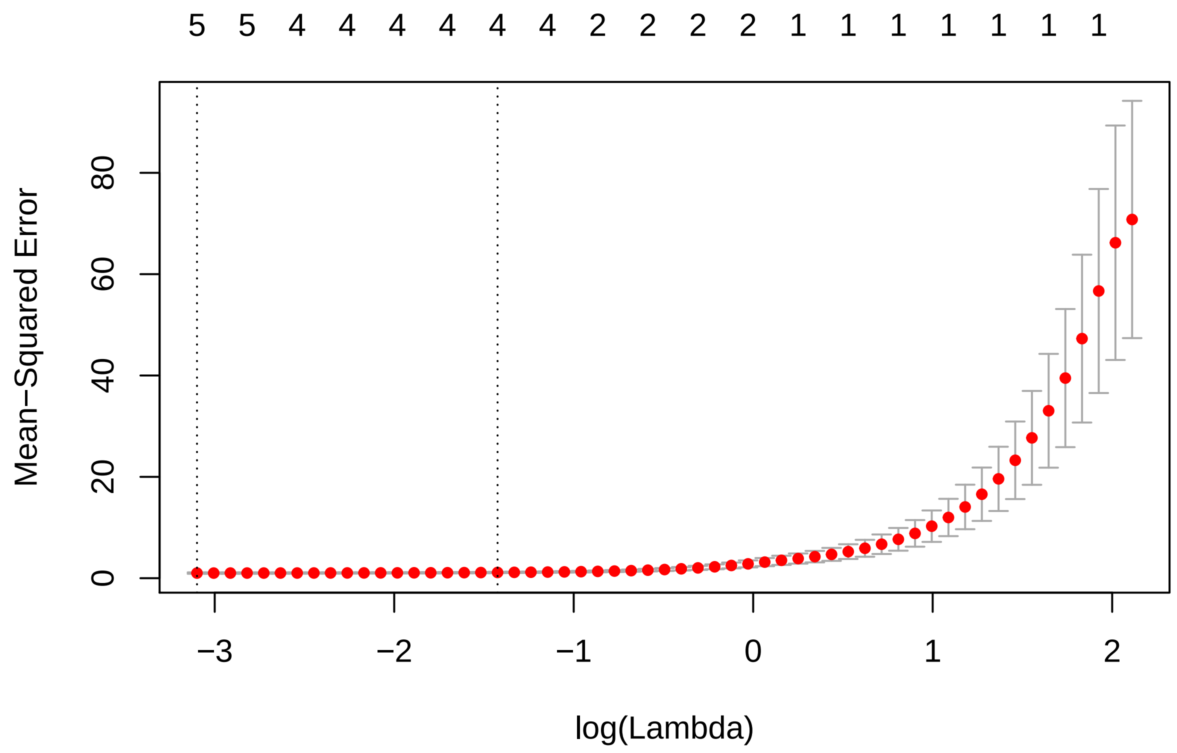
```
## Loaded glmnet 2.0-16
```

```
xmatrix = model.matrix(Y ~ poly(x, 10, raw = T), data = data_)[, -1]
fit.lasso = cv.glmnet(xmatrix, Y, alpha = 1)
best.lambda = fit.lasso$lambda.min
cat(best.lambda)
```

```
## 0.04510738
```

```
plot(fit.lasso)
```

8 e-1.bb



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

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                                1
## (Intercept)           4.0732782668
## poly(x, 10, raw = T)1  0.2032461427
## poly(x, 10, raw = T)2  0.6650355659
## poly(x, 10, raw = T)3  3.0133709149
## poly(x, 10, raw = T)4  0.0510550101
## poly(x, 10, raw = T)5  .
## poly(x, 10, raw = T)6  0.0002661783
## poly(x, 10, raw = T)7  .
## poly(x, 10, raw = T)8  .
## poly(x, 10, raw = T)9  .
## poly(x, 10, raw = T)10 .
```

We see that X4, X6-10 are not chosen and the corresponding coeff are made zero by lasso.

```
beta7 = 5
Y = beta0 * 1 + beta7 * x^7 + noise

data_ = data.frame(y = Y, x = x)
reg.fit = regsubsets(y ~ poly(x, 10, raw = T), data = data_, nvmax = 10)

reg.fit.summary <- summary(reg.fit)
plot.data <- summary.as.data.frame(reg.fit.summary, vars)
plot.data$n <- as.numeric(rownames(plot.data))

lapply(plot.data, function(x){which(x == min(x))})
```

```
## $cp
## [1] 1
##
## $bic
## [1] 1
##
## $adjr2
## [1] 10
##
## $n
## [1] 1
```

```
coefficients(reg.fit, id = 1)
```

```
##          (Intercept) poly(x, 10, raw = T)7
##             3.893251              4.999704
```

We see that best subset selection algorithm selects x7 as the best predictor. Also from all bic, adjr2, cp we have only one predictor as feature set.

```
# lasso
xmatrix = model.matrix(Y ~ poly(x, 10, raw = T), data = data_)[, -1]
fit.lasso = cv.glmnet(xmatrix, Y, alpha = 1)
best.lambda = fit.lasso$lambda.min
cat(best.lambda)
```

```
## 8.390058
```

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

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept)            4.325165
## 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  4.839751
## poly(x, 10, raw = T)8  .
## poly(x, 10, raw = T)9  .
## poly(x, 10, raw = T)10 .
```

We can see that lasso selects X7 (4.839) and suppresses other coeff. The prediction for intercept (5.6) is off from the actual (4).

4.(15 points) Section 6.8, page 263, question 9

a.

```
# Load the library and the dataset
library(ISLR)
attach(College)

# Set random seed
set.seed(99)
```

```r
# Splitting the data according to a 70/30 ratio
split = sample(nrow(College),nrow(College)*0.7)
train = College[split,]
test = College[-split,]
```

b.

```r
# Fit a linear model on the training set
fitLm <- lm(Apps ~ ., data = train)

# Predict the values of the testing set using the model
predLm <- predict(fitLm, test)

# Calculate and display the MSE
lmMSE = mean((predLm - test$Apps)^2)
lmMSE
```

```
## [1] 766479.5
```

c.

```r
# Create a matrix for the training and testing sets
trainMat = model.matrix(Apps~.,data=train)
testMat = model.matrix(Apps~.,data=test)

# Define a grid to cover the range of lambda
grid = 10^seq(4,-2,length=100)

# Fit the ridge regression
library(glmnet)
ridge = glmnet(trainMat,train$Apps,alpha=0,lambda=grid,thresh = 1e-12)

# Cross validating the model
crossVRidge = cv.glmnet(trainMat,train$Apps,alpha=0,lambda=grid,thresh=1e-12)

# Find the minimum lambda for which the cross validation error is minimal
estRidge = crossVRidge$lambda.min

# Use this lambda value directly on the testing set to predict
predRidge <-predict(ridge, s=estRidge, newx=testMat)

# Calculate and display the MSE
ridgeMSE = mean((test$Apps-predRidge)^2)
ridgeMSE
```

```
## [1] 766464.9
```

d.

```r
# Fit the lasso regression
lasso = glmnet(trainMat, train$Apps, alpha = 1, lambda = grid, thresh = 1e-12)

# Cross validating the model
crossVLasso = cv.glmnet(trainMat, train$Apps, alpha = 1, lambda = grid, thresh = 1e-12)

# Find the minimum lambda for which the cross validation error is minimal
```

```r
estLasso = crossVLasso$lambda.min

# Use this lambda value directly on the testing set to predict
predLasso = predict(lasso, s = estLasso, newx = testMat)

# Calculate and display the MSE
lassoMSE = mean((predLasso - test$Apps)^2)
lassoMSE
```

```
## [1] 725420.5
```

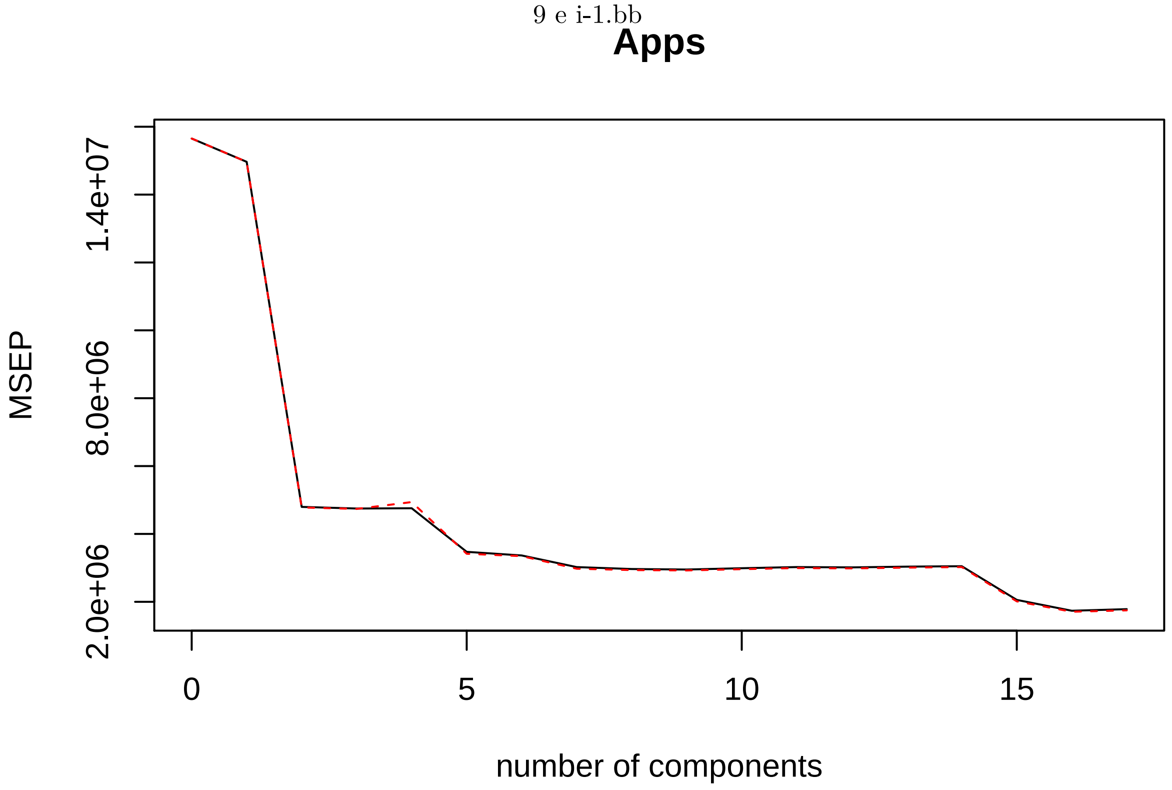
e.

```r
# Import the library
library(pls)

# Fit a pcr model on the training set
pcrFit = pcr(Apps ~ ., data = train, scale = TRUE, validation = "CV")

# Plot MSEP vs the number of predictors for PCR
validationplot(pcrFit, val.type="MSEP")
```

9 e i-1.bb



```r
# Choose the number of components based on the previous plot
predPcr <- predict(pcrFit, test, ncomp = 17)

# Calculate and display the MSE
pcrMSE = mean((predPcr - test$Apps)^2)
pcrMSE
```

```
## [1] 766479.5
```

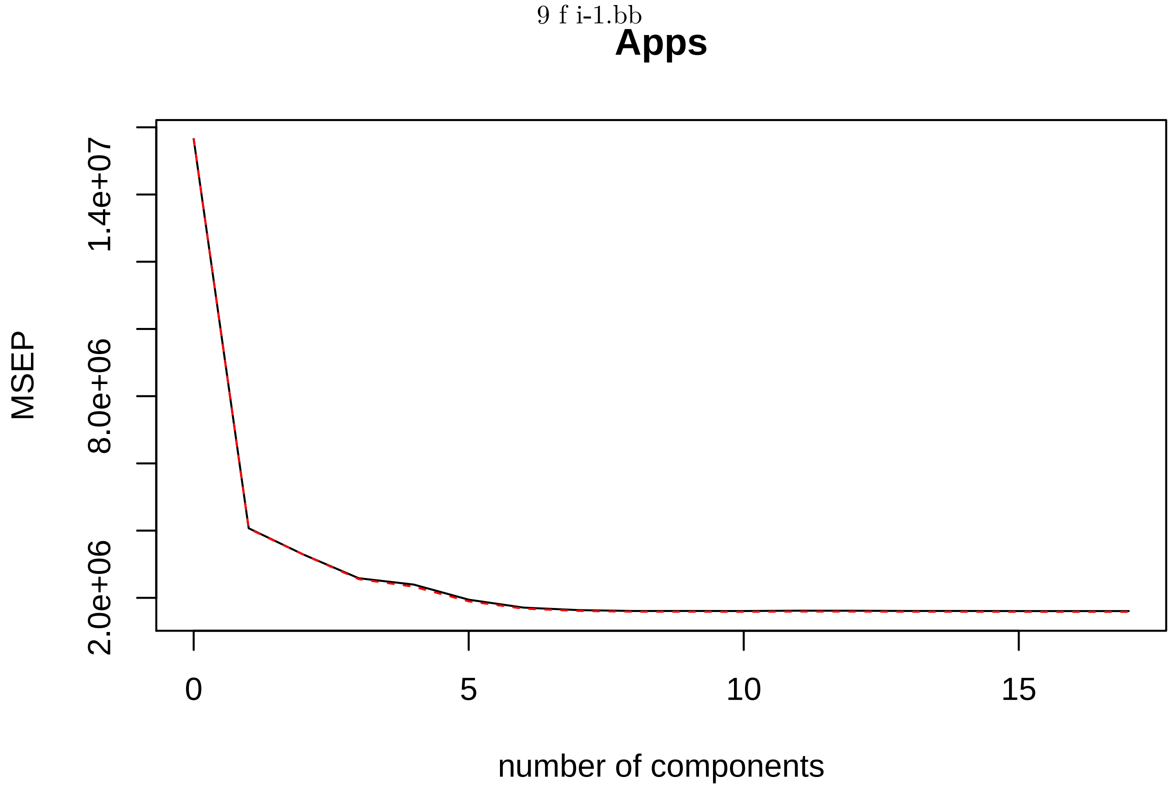
f.

```r
# Import the library
library(pls)

# Fit a pcr model on the training set
plsFit = plsr(Apps ~ ., data = train, scale = TRUE, validation = "CV")

# Plot MSEP vs the number of predictors for PLS
validationplot(plsFit, val.type="MSEP")
```

9 f i-1.bb



```r
# Choose the number of components based on the previous plot
predPls <- predict(plsFit, test, ncomp = 15)

# Calculate and display the MSE
plsMSE = mean((predPls - test$Apps)^2)
plsMSE
```

```
## [1] 766367.4
```

g.

Let's first calculate the test R^2 for all the models fitted.

```r
# Calculate the mean of apps in the testing set
avg = mean(test$Apps)

# Calculate the R2 for the linear model
lmR2 = 1 - mean((predLm - test$Apps)^2) / mean((avg - test$Apps)^2)

# Calculate the R2 for the ridge regression model
```

```r
ridgeR2 <- 1 - mean((predRidge - test$Apps)^2) / mean((avg - test$Apps)^2)

# Calculate the R2 for the lasso regression model
lassoR2 <- 1 - mean((predLasso - test$Apps)^2) / mean((avg - test$Apps)^2)

# Calculate the R2 for the PCR model
pcrR2 <- 1 - mean((predPcr - test$Apps)^2) / mean((avg - test$Apps)^2)

# Calculate the R2 for the PLS model
plsR2 <- 1 - mean((predPls - test$Apps)^2) / mean((avg - test$Apps)^2)
```

In the table below, we can observe a summary of the previously calculated MSE and R^2 for all the five models fitted.

	MSE	R^2
Linear Model	`r lmMSE`	`r lmR2`
Ridge	`r ridgeMSE`	`r ridgeR2`
Lasso	`r lassoMSE`	`r lassoR2`
PCR	`r pcrMSE`	`r pcrR2`
PLS	`r plsMSE`	`r plsR2`

We can note that all the models share the same performance except for the "Lasso regression" which presents a slightly lower MSE and higher R^2. So, in this example, the Lasso regression performs a bit better than the other models in predicting the number of college applications even though the other models are able to predict it with a high accuracy.

However, when we change the random seed, we can observe that this is not always the case (that Lasso performs better) but in almost all the cases, the difference between the models' performances is not huge.