PH.140.644_HW2

Kunyu An

Chapter 5

Q1

$$\begin{aligned} Var(\alpha X, (1-\alpha)Y) &= \alpha^2\sigma_X^2 + 2\alpha(1-\alpha)\sigma_{XY} + (1-\alpha)^2\sigma_Y^2 \\ &= (\sigma_X^2 - 2\sigma_{XY} + \sigma_Y^2)\alpha^2 + 2(\sigma_{XY} - \sigma_Y^2)\alpha + \sigma_Y^2 \end{aligned} \tag{1}$$

Let

$$\frac{\partial Var(\alpha X, (1-\alpha)Y)}{\partial \alpha} = 2(\sigma_X^2 - 2\sigma_{XY} + \sigma_Y^2)\alpha + 2(\sigma_{XY} - \sigma_Y^2) = 0 \tag{2}$$

We have,

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}} \tag{3}$$

Since we also have

$$\frac{\partial^2 Var(\alpha X, (1-\alpha)Y)}{\partial \alpha^2} = 2(\sigma_X^2 - 2\sigma_{XY} + \sigma_Y^2) \geq 0 \tag{4}$$

Therefore, when $\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$, $Var(\alpha X, (1-\alpha)Y)$ reaches its minimum.

Q3

We now review k-fold cross-validation.

a.Explain how k-fold cross-validation is implemented.

For k-fold cross validation, we first need to divide the sample randomly into k equal-size and non-overlapping groups. Then, every time we use one group as the validation set and the rest as a whole to be the training set. Finally we can compute the testing error by averaging over the statistics of interest in k experiments.

b.what are the advantages and disadvantages of k-fold cross- validation relative to:

i. The validation set approach?

1. The k-fold cross validation has a much lower variability than the validation set approach.
2. All the data is used to both train and test model performance for k-fold CV.
3. The validation set approach is much easier to understand and to preform. A model is only trained once and tested once. In k-fold CV, k models will be trained and tested.

ii. LOOCV?

1. k-fold cross validation is less computationally demanding then LOOCV.
2. LOOCV has higher variance than k-fold CV.

Q5

In Chapter 4, we used logistic regression to predict the probability of `default` using `income` and `balance` on the `Default` data set. We will now estimate the test error of this logistic regression model using the validation set approach. Do not forget to set a random seed before beginning your analysis.

a. Fit a logistic regression model that uses `income` and `balance` to predict `default`.

```
library(ISLR)
attach(Default)
set.seed(1)
fit.glm = glm(default ~ income + balance, data = Default, family = "binomial")
summary(fit.glm)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

b. Using the validation set approach, estimate the test error of this model. In order to do this, you must perform the following steps:

i. Split the sample set into a training set and a validation set.

```
smp_siz <- floor(0.7*nrow(Default))
idx <- sample(seq_len(nrow(Default)),size = smp_siz)
train<-Default[idx,]
test<-Default[-idx,]
```

ii. Fit a multiple logistic regression model using only the training observations.

```
fit.glm = glm(default ~ income + balance, data = train, family = "binomial")
summary(fit.glm)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4481  -0.1402  -0.0561  -0.0211   3.3484
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.167e+01  5.214e-01 -22.379  < 2e-16 ***
## income       2.560e-05  6.012e-06   4.258 2.06e-05 ***
## balance      5.574e-03  2.678e-04  20.816  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2030.3  on 6999  degrees of freedom
## Residual deviance: 1079.6  on 6997  degrees of freedom
## AIC: 1085.6
##
## Number of Fisher Scoring iterations: 8
```

iii. Obtain a prediction of default status for each individual in the validation set by computing the posterior probability of default for that individual, and classifying the individual to the `default` *category if the posterior probability is greater than 0.5.*

```
probs.glm=predict(fit.glm,newdata = test, type='response')
pred.glm=rep('Yes',length(probs.glm))
pred.glm[probs.glm<0.5]='No'
```

iv. Compute the validation set error, which is the fraction of the observations in the validation set that are misclassified.

```
mean(pred.glm != test$default)
```

```
## [1] 0.02666667
```

c. Repeat the process in (b) three times, using three different splits of the observations into a training set and a validation set. Comment on the results obtained.

```
errors <- c()
set.seed(1)
for(i in 1:3){
  smp_siz <- floor(0.7*nrow(Default))
  idx <- sample(seq_len(nrow(Default)),size = smp_siz)
  train <- Default[idx,]
  test <- Default[-idx,]
  fit.glm = glm(default ~ income + balance, data = train, family = "binomial")
  probs.glm = predict(fit.glm,newdata = test, type='response')
```

```
  pred.glm = rep('Yes',length(probs.glm))
  pred.glm[probs.glm<0.5]='No'
  errors[i] <- mean(pred.glm != test$default)
}
errors
```

```
## [1] 0.02666667 0.02800000 0.02300000
```

We can see that the validation set error varies as the training set varies but they are not very far away from each other.

d. Now consider a logistic regression model that predicts the prob- ability of `default` using `income`, `balance`, and a dummy variable for `student`. Estimate the test error for this model using the validation set approach. Comment on whether or not including a dummy variable for `student` leads to a reduction in the test error rate.

```
smp_siz <- floor(0.7*nrow(Default))
idx <- sample(seq_len(nrow(Default)),size = smp_siz)
train <- Default[idx,]
test <- Default[-idx,]
fit.glm = glm(default ~ income + balance + student, data = train, family = "binomial")
probs.glm = predict(fit.glm,newdata = test, type='response')
pred.glm = rep('Yes',length(probs.glm))
pred.glm[probs.glm<0.5]='No'
mean(pred.glm != test$default)
```

```
## [1] 0.028
```

The error is very close to previous ones. Thus we can say that including a dummy variable for student does not lead to a reduction in the test error rate.

Q6

We continue to consider the use of a logistic regression model to predict the probability of `default` using `income` and `balance` on the `Default` data set. In particular, we will now compute estimates for the standard errors of the `income` and `balance` logistic regression coefficients in two different ways: (1) using the bootstrap, and (2) using the standard formula for computing the standard errors in the `glm()` function. Do not forget to set a random seed before beginning your analysis.

(a). Using the summary() and glm() functions, determine the estimated standard errors for the coefficients associated with income and balance in a multiple logistic regression model that uses both predictors.

```
set.seed(1)
attach(Default)
fit.glm <- glm(default ~ income + balance, data = Default, family = "binomial")
summary(fit.glm)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = "binomial",
##     data = Default)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

The estimated standard errors for the coefficients associated with income and balance are $4.985e-06$ and $2.274e-04$.

(b). Write a function, `boot.fn()`, that takes as input the `Default` data set as well as an index of the observations, and that outputs the coefficient estimates for `income` and `balance` in the multiple logistic regression model.

```
boot.fn <- function(data,index){
    coef(glm(default ~ income + balance, family = "binomial", data = Default, subset = index))
}
```

(c). Use the `boot()` function together with your `boot.fn()` function to estimate the standard errors of the logistic regression coefficients for `income` and `balance`.

```
library(boot)
boot(Default,boot.fn,1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01 -3.945460e-02 4.344722e-01
## t2*  2.080898e-05  1.680317e-07 4.866284e-06
## t3*  5.647103e-03  1.855765e-05 2.298949e-04
```

The estimated standard errors for the coefficients associated with income and balance are $4.946e-06$ and $2.277e-04$.

(d). Comment on the estimated standard errors obtained using the `glm()` function and using your bootstrap function.

The standard errors obtained from glm() and bootstrap are very close,so the bootstrap method is efficient.

Chapter 6

Q1

We perform best subset, forward stepwise, and backward stepwise selection on a single data set. For each approach, we obtain p + 1 models, containing 0, 1, 2, . . . , p predictors. Explain your answers:

(a). Which of the three models with k predictors has the smallest training RSS?

- The smallest training RSS will be for the model with best subset approach.

(b). Which of the three models with k predictors has the smallest test RSS?

- We can not say which one is absolutely the best among three models since they all based on the combination for minimizing the training RSS.

(c). True or False:

i. True. For forward stepwise selection, the predictors were added everytime.

ii. True. For the backwards stepwise selection, we're reducing preidctor.

iii. False. The algorithms are different, we can not compare them.

iv. False. Same as part iii.

v. False. The subset with lowest RSS on training set is selected.

Q8

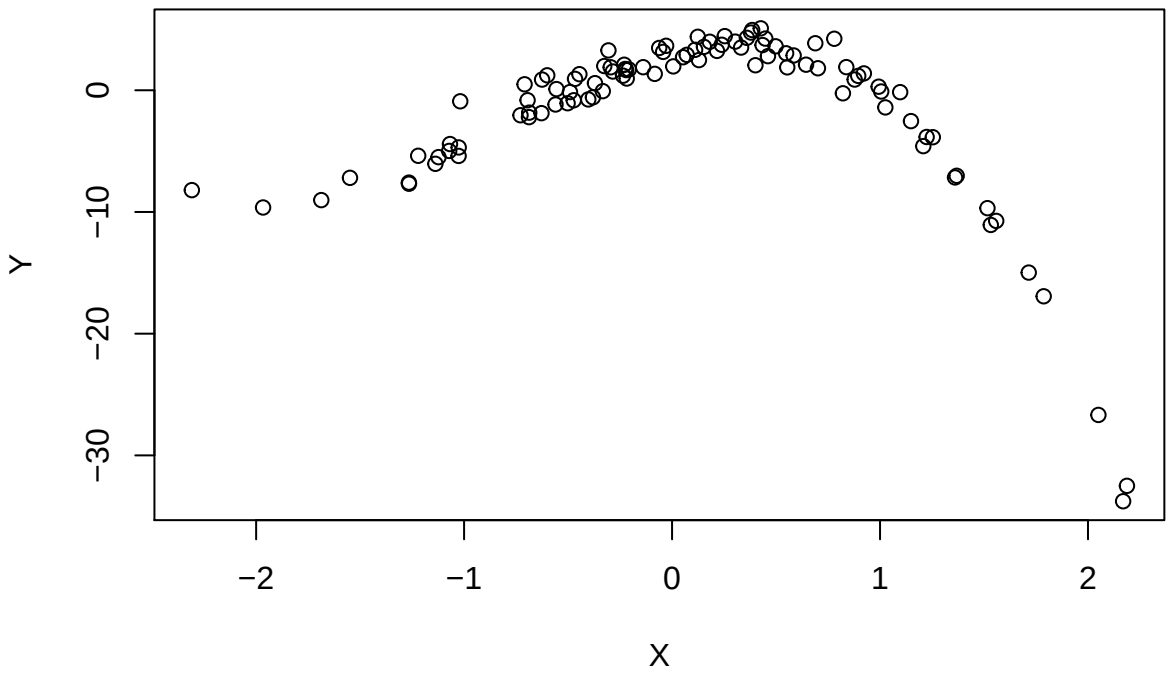
In this exercise, we will generate simulated data, and will then use this data to perform best subset selection.

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

```
set.seed(123)
X = rnorm(100)
eps = 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 $\beta_0, \beta_1, \beta_2, \beta_3$, are constants of your choice. We select $\beta_0 = 2, \beta_1 = 4, \beta_2 = -5, \beta_3 = -1$

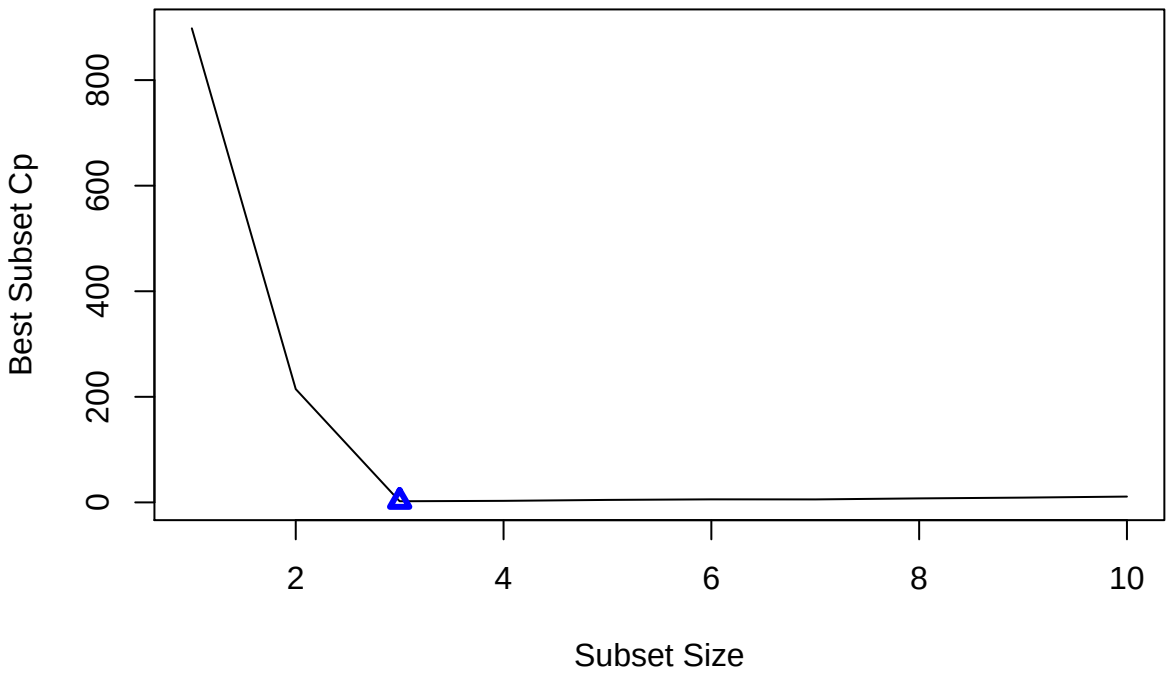
```
Y = 3 + 4 * X + -5 * X^2 + -2* X^3 + eps
plot(X, Y)
```



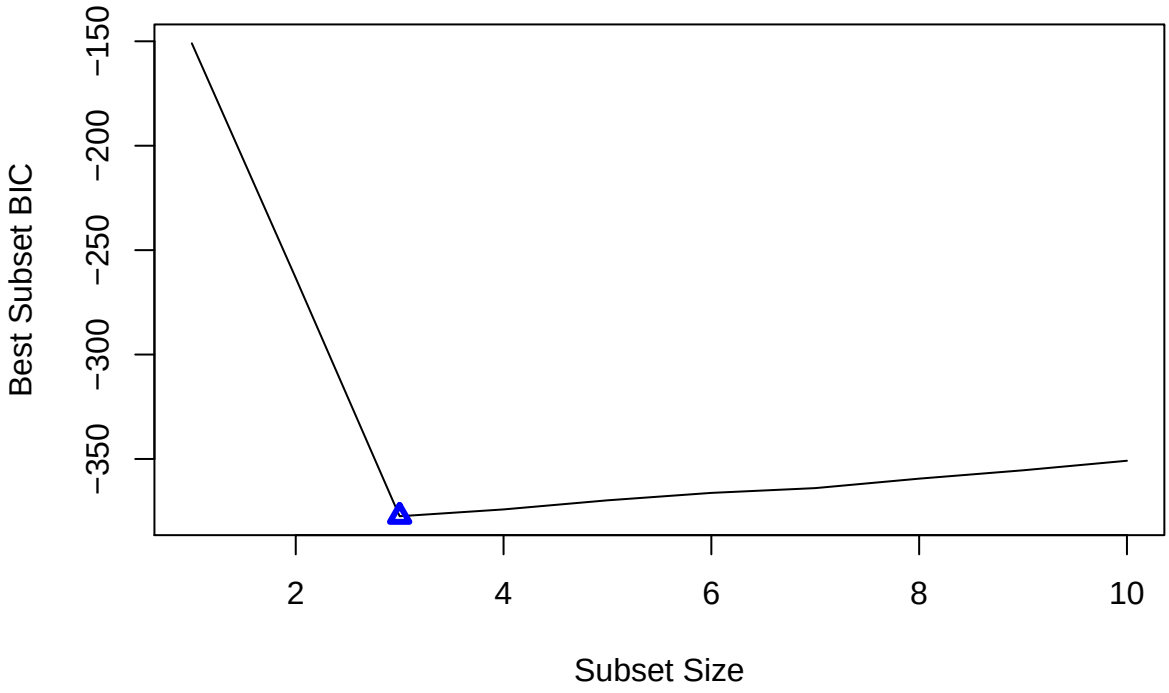
(c) Use the regsubsets() function to perform best subset selection in order to choose the best model containing the predictors X,X2, . . .,X10. What is the best model obtained according to Cp, BIC, and adjusted R2? Show some plots to provide evidence for your answer, and report the coefficients of the 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)
data.full = data.frame(y = Y, x = X)
model = regsubsets(y = Y,x = poly(X, 10, raw = T), nvmax = 10, method = "exhaustive")
model_summary = summary(model)
```

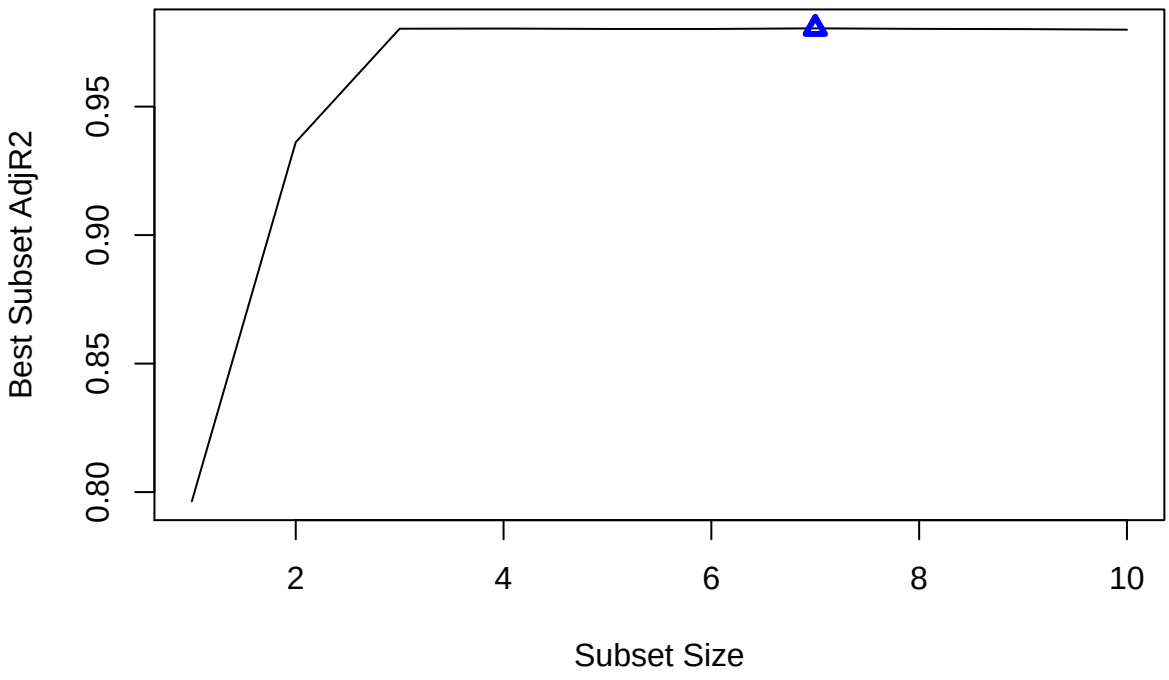
```
plot(model_summary$cp, xlab="Subset Size", ylab="Best Subset Cp", type="l")
points(which.min(model_summary$cp), model_summary$cp[which.min(model_summary$cp)], col="blue", pch=2, l
```




```
plot(model_summary$bic, xlab="Subset Size", ylab="Best Subset BIC", type="l")
points(which.min(model_summary$bic), model_summary$bic[which.min(model_summary$bic)], col="blue", pch=2
```




```
plot(model_summary$adjr2, xlab="Subset Size", ylab="Best Subset AdjR2", type="l")
points(which.max(model_summary$adjr2), model_summary$adjr2[which.max(model_summary$adjr2)], col="blue",
```




```
model_summary$outmat
```

```
##          1   2   3   4   5   6   7   8   9   10
## 1  ( 1 )  " " "*" " " " " " " " " " " " " " " " "
## 2  ( 1 )  " " "*" " " " " " " " " " " "*" " " " "
## 3  ( 1 )  "*" "*" "*" " " " " " " " " " " " " " "
## 4  ( 1 )  "*" "*" "*" " " " " " " "*" " " " " " "
## 5  ( 1 )  "*" "*" "*" " " " " " " " " "*" " " "*"
## 6  ( 1 )  "*" "*" "*" "*" " " " " "*" " " "*" " "
## 7  ( 1 )  "*" "*" "*" "*" " " " " "*" " " "*" " "
## 8  ( 1 )  "*" "*" "*" "*" "*" "*" " " " " "*" " "
## 9  ( 1 )  "*" "*" "*" "*" "*" "*" "*" "*" " " "*"
## 10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

Using Cp and BIC, the best size of subset is 3.

```
coef(model, id = 3)
```

```
## (Intercept)           1           2           3
##    2.970394    3.920446   -5.091543   -1.979564
```

Using Adjusted R-Squared, the best size of subset is 7.

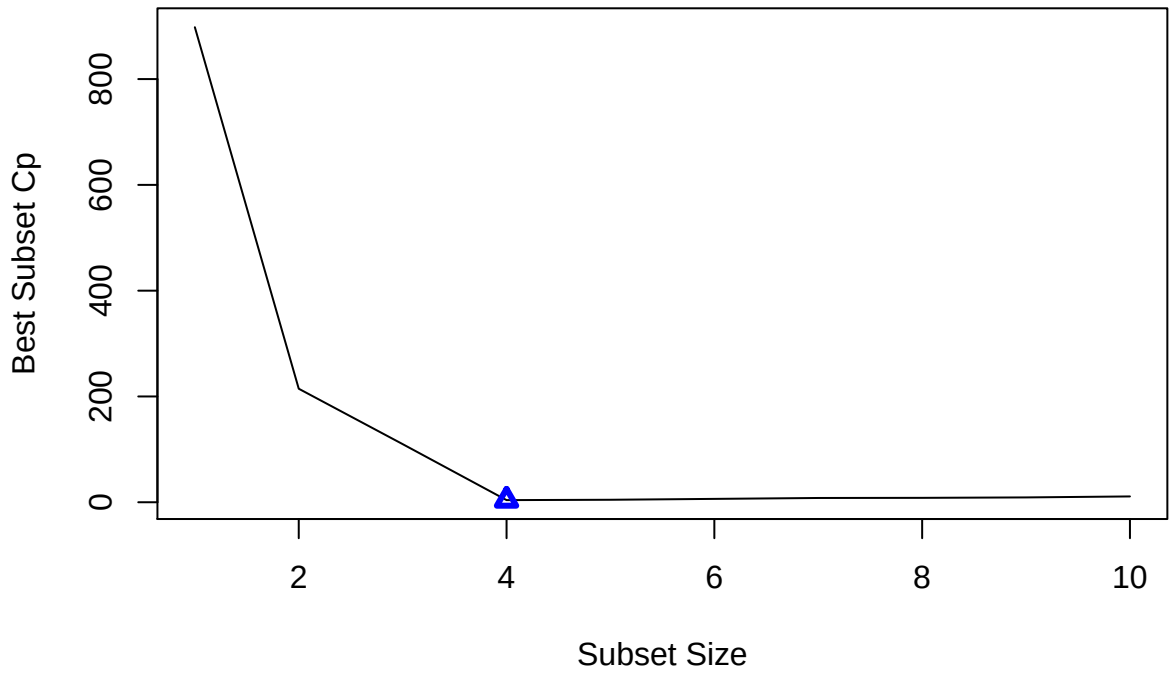
```
coef(model, id = 7)
```

```
## (Intercept)           1           2           3           4           6
##   2.7931750   3.8907484  -2.7241152  -1.9469486  -4.0243410   2.2138377
##           8          10
##  -0.4870719   0.0373538
```

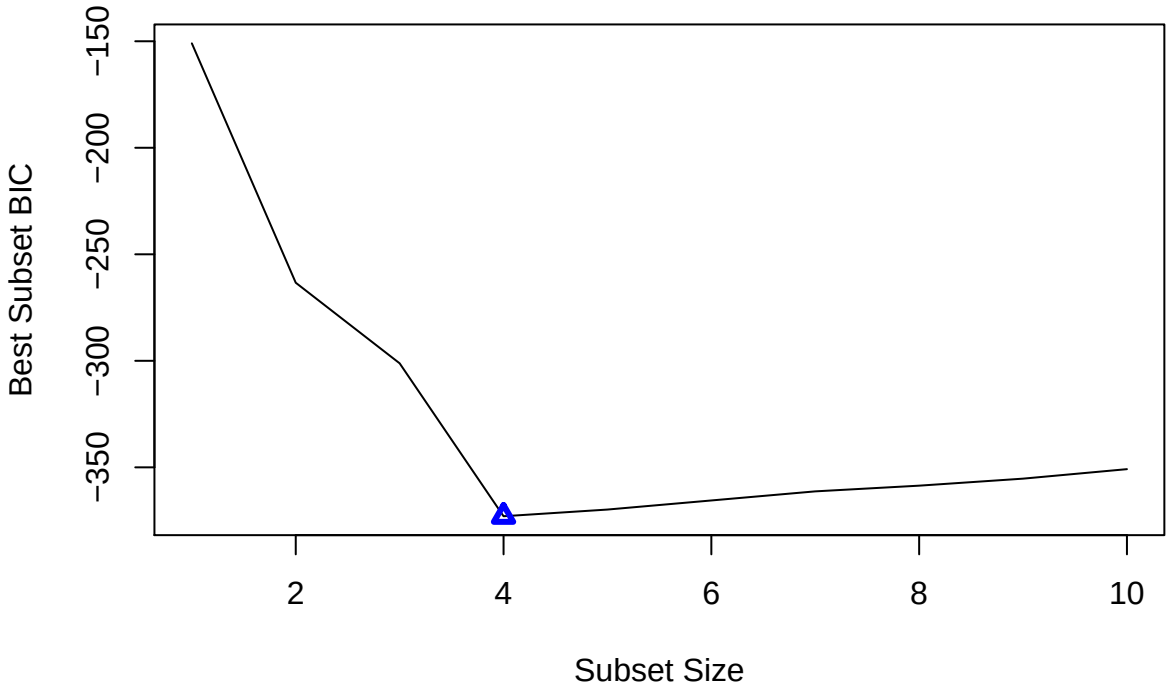
(d) Repeat (c), using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in (c)? Forward stepwise selection

```
model = regsubsets(y = Y,x = poly(X, 10, raw = T), nvmax = 10, method = "forward")
model_summary = summary(model)
```

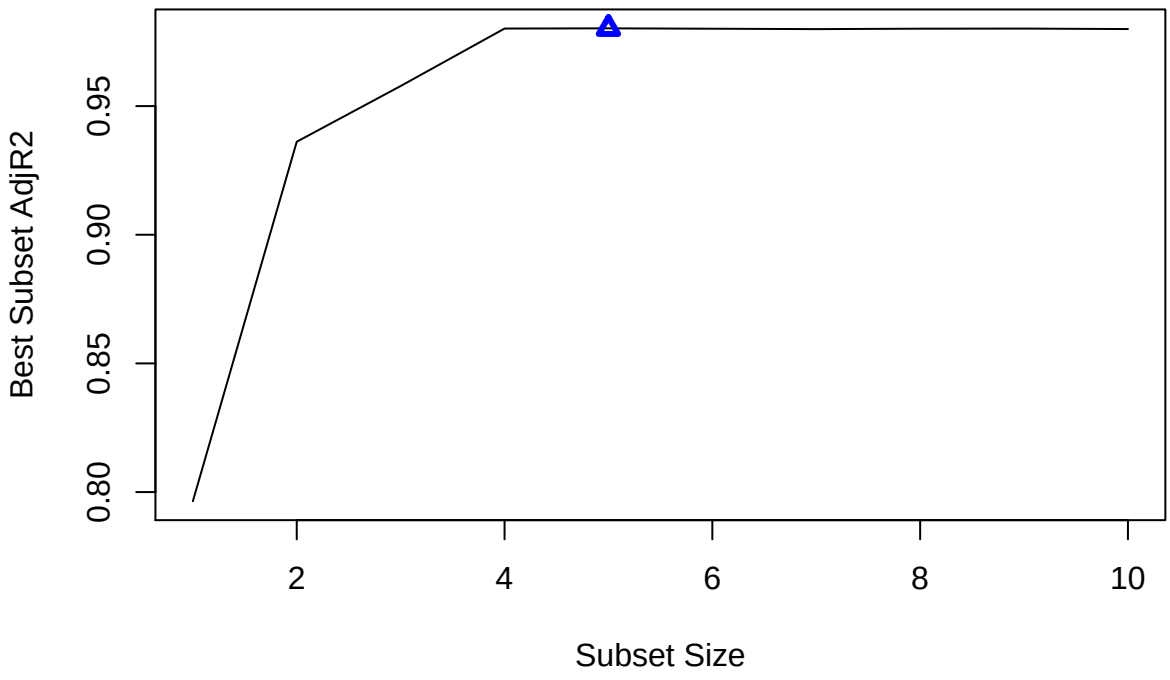
```
plot(model_summary$cp, xlab="Subset Size", ylab="Best Subset Cp", type="l")
points(which.min(model_summary$cp), model_summary$cp[which.min(model_summary$cp)], col="blue", pch=2, lw
```




```
plot(model_summary$bic, xlab="Subset Size", ylab="Best Subset BIC", type="l")
points(which.min(model_summary$bic), model_summary$bic[which.min(model_summary$bic)], col="blue", pch=2
```




```
plot(model_summary$adjr2, xlab="Subset Size", ylab="Best Subset AdjR2", type="l")
points(which.max(model_summary$adjr2), model_summary$adjr2[which.max(model_summary$adjr2)], col="blue",
```




```
model_summary$outmat
```

```
##          1   2   3   4   5   6   7   8   9   10
## 1  ( 1 )  " " "*" " " " " " " " " " " " " " " " "
## 2  ( 1 )  " " "*" " " " " " " " " " " "*" " " " " "
## 3  ( 1 )  "*" "*" " " " " " " " " " " "*" " " " " "
## 4  ( 1 )  "*" "*" "*" " " " " " " " " "*" " " " " "
## 5  ( 1 )  "*" "*" "*" " " " " " " " " "*" " " "*" " "
## 6  ( 1 )  "*" "*" "*" " " " " " " "*" "*" " " "*" " "
## 7  ( 1 )  "*" "*" "*" " " " " " " "*" "*" " " "*" "*"
## 8  ( 1 )  "*" "*" "*" "*" " " " " "*" "*" " " "*" "*"
## 9  ( 1 )  "*" "*" "*" "*" " " " " "*" "*" "*" "*" "*"
## 10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

Using Cp and BIC, the best size of subset is 4.

```
coef(model, id = 4)
```

```
##  (Intercept)            1            2            3            7
##  2.972216767  3.852611672 -5.098852110 -1.918623281 -0.002237796
```

Using Adjusted R-Squared, the best size of subset is 5.

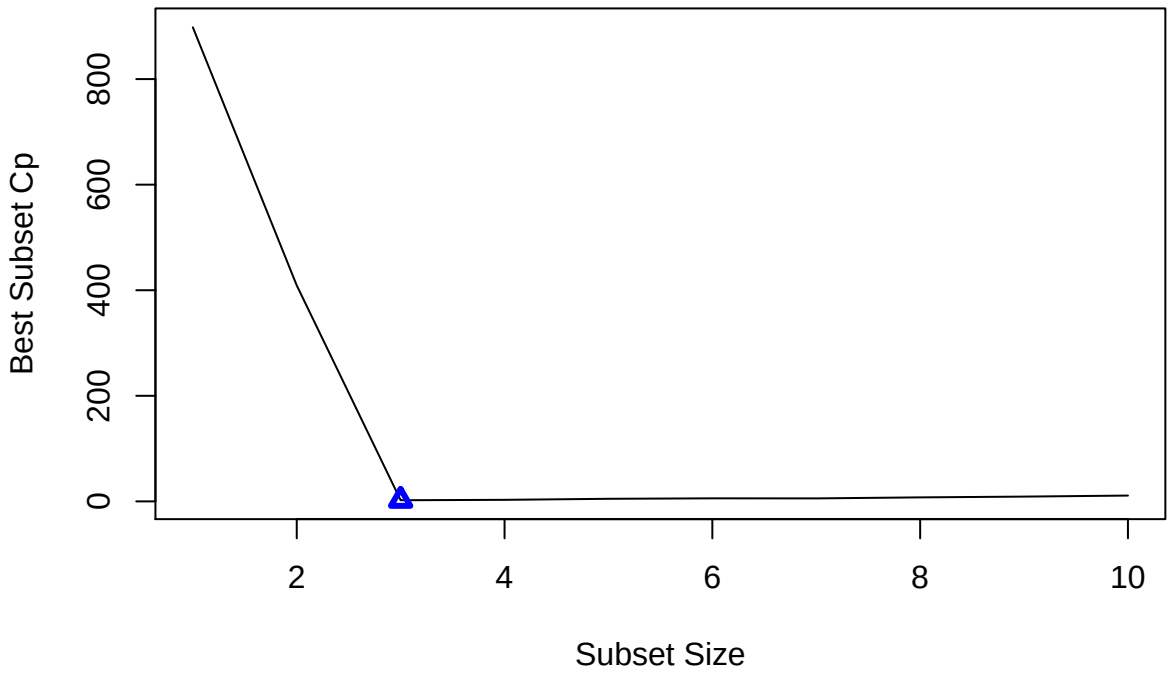
```
coef(model, id = 5)
```

```
##  (Intercept)            1            2            3            7
##  3.000021450  4.096621422 -5.142449796 -2.250825861  0.055863160
##            9
## -0.009356419
```

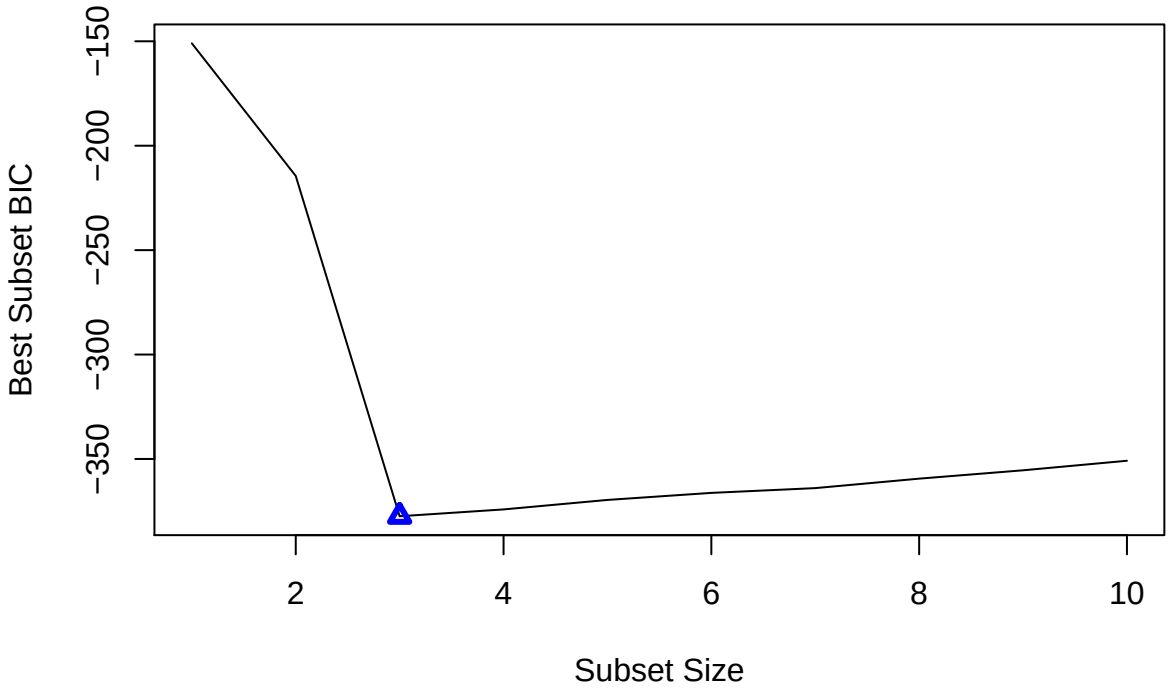
Backwards stepwise selection

```
model = regsubsets(y = Y,x = poly(X, 10, raw = T), nvmax = 10, method = "backward")
model_summary = summary(model)
```

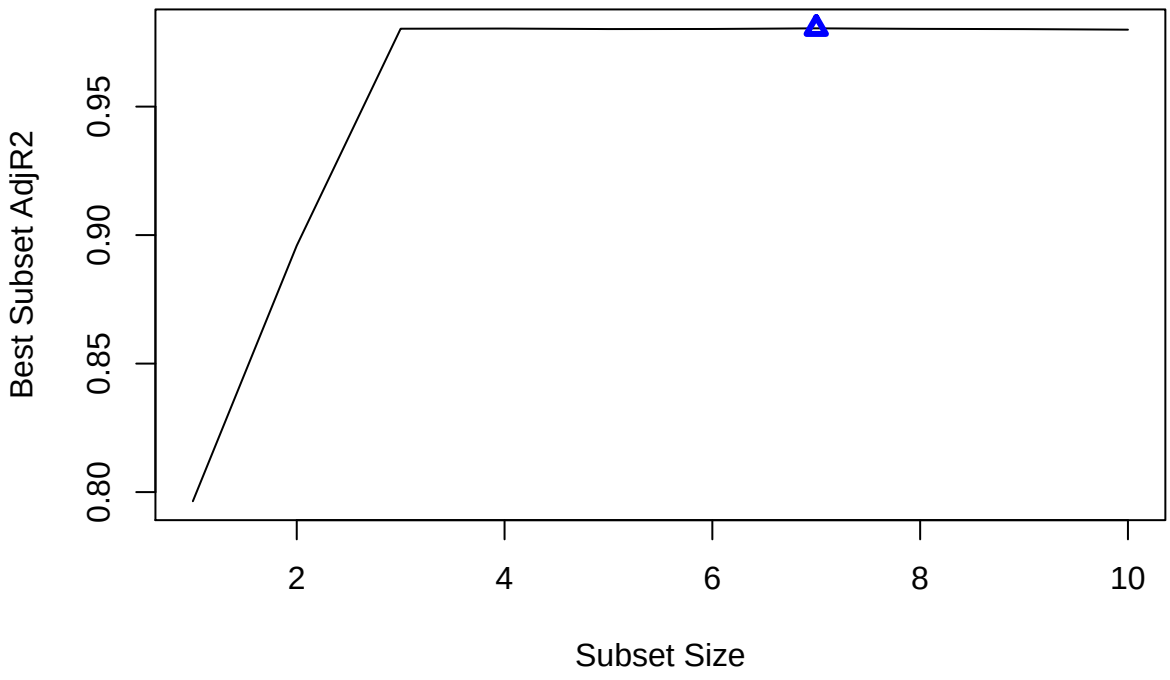
```
plot(model_summary$cp, xlab="Subset Size", ylab="Best Subset Cp", type="l")
points(which.min(model_summary$cp), model_summary$cp[which.min(model_summary$cp)], col="blue", pch=2, lw
```




```
plot(model_summary$bic, xlab="Subset Size", ylab="Best Subset BIC", type="l")
points(which.min(model_summary$bic), model_summary$bic[which.min(model_summary$bic)], col="blue", pch=2
```




```
plot(model_summary$adjr2, xlab="Subset Size", ylab="Best Subset AdjR2", type="l")
points(which.max(model_summary$adjr2), model_summary$adjr2[which.max(model_summary$adjr2)], col="blue",
```




```
model_summary$outmat
```

```
##          1   2   3   4   5   6   7   8   9   10
## 1  ( 1 )  " " "*" " " " " " " " " " " " " " " " "
## 2  ( 1 )  " " "*" "*" " " " " " " " " " " " " " "
## 3  ( 1 )  "*" "*" "*" " " " " " " " " " " " " " "
## 4  ( 1 )  "*" "*" "*" " " " " " " "*" " " " " " "
## 5  ( 1 )  "*" "*" "*" "*" " " " " "*" " " " " " "
## 6  ( 1 )  "*" "*" "*" "*" " " " " "*" " " " " "*"
## 7  ( 1 )  "*" "*" "*" "*" " " " " "*" " " " " "*"
## 8  ( 1 )  "*" "*" "*" "*" "*" "*" "*" " " " " "*"
## 9  ( 1 )  "*" "*" "*" "*" "*" "*" "*" "*" " " "*"
## 10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

Using Cp and BIC, the best size of subset is 3.

```
coef(model, id = 3)
```

```
## (Intercept)           1           2           3
##    2.970394    3.920446   -5.091543   -1.979564
```

Using Adjusted R-Squared, the best size of subset is 7.

```
coef(model, id = 7)
```

```
## (Intercept)           1           2           3           4           6
##   2.7931750   3.8907484  -2.7241152  -1.9469486  -4.0243410   2.2138377
##           8          10
##  -0.4870719   0.0373538
```

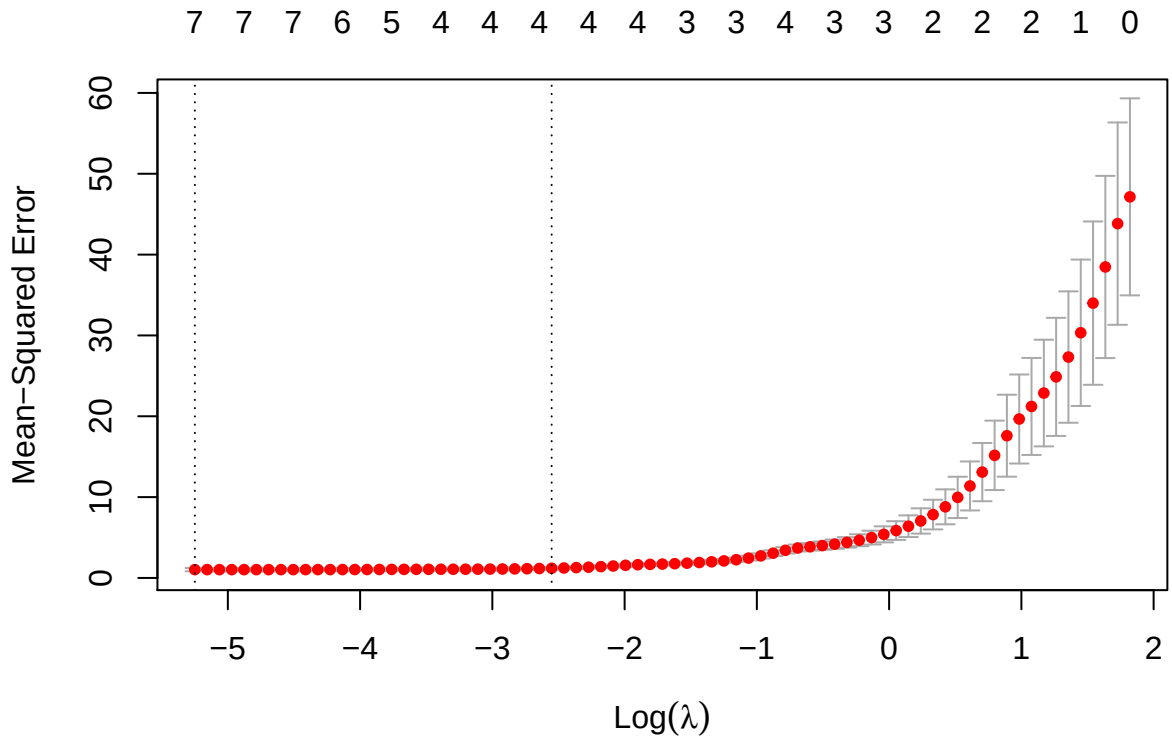
Different selection methods and metrics result in different subsets of the predictors being selected.

(e) Now fit a lasso model to the simulated data, again using X,X2,..., X10 as predictors. Use cross-validation to select the optimal value of . Create plots of the cross-validation error as a function of . Report the resulting coefficient estimates, and discuss the results obtained.

```
library(glmnet)
set.seed(234)
xmat = model.matrix(y ~ poly(x, 10, raw = T), data = data.full)[, -1]
mod.lasso = cv.glmnet(xmat, Y, alpha = 1)
best.lambda = mod.lasso$lambda.min
best.lambda
```

```
## [1] 0.005249866
```

```
plot(mod.lasso)
```



```
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)             3.0161542593
## poly(x, 10, raw = T)1   3.8056266568
## poly(x, 10, raw = T)2  -5.1971657907
## poly(x, 10, raw = T)3  -1.8956227207
## poly(x, 10, raw = T)4   .
## poly(x, 10, raw = T)5   .
## poly(x, 10, raw = T)6   0.0002816093
## poly(x, 10, raw = T)7  -0.0008554854
## poly(x, 10, raw = T)8   0.0011770029
## poly(x, 10, raw = T)9  -0.0003163914
## poly(x, 10, raw = T)10  .
```

Lasso also picks X1, X2, X7. It also picks X6, X7, X8, X9 with negligible coefficient.

(f) Now generate a response vector Y according to the model $Y = 0 + 7X7 + e$, and perform best subset selection and the lasso. Discuss the results obtained. We select $\beta_0 = 2, \beta_7 = 8$

```
Y = 10 + 5*X^7 + eps
```

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

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

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

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

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

```
which.min(mod.summary$adjr2)
```

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

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

```
##           (Intercept) poly(x, 10, raw = T)7
##              9.893251              4.999704
```

We can see cp and BIC have the most accurate 1-variable model, but Adjust R2 pick additional variables.

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

```
## [1] 7.644709
```

```
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)             10.28679
## 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    4.85396
## poly(x, 10, raw = T)8    .
## poly(x, 10, raw = T)9    .
## poly(x, 10, raw = T)10   .
```

We can see lasso also have the most accurate 1-variable model.

Q9

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.

```
set.seed(1)
library(ISLR)
attach(College)
smp_siz <- floor(0.7*nrow(College))
idx <- sample(seq_len(nrow(College)),size = smp_siz)
train <- College[idx,]
test <- College[-idx,]
```

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

```
set.seed(1)
fit.lm <- lm(Apps ~ ., data = train)
lm.pred <- predict(fit.lm, test)
(lm.mse <- mean((lm.pred - test$Apps)^2))
```

```
## [1] 1261630
```

The MSE is 1.2616304×10^6

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

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

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

The best is0.01

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

```
## [1] 1261598
```

The test MSE is 1.2615981×10^6**, which is very close but smaller than the linear regression.**

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

```
mod.lasso = cv.glmnet(train.mat, train[, "Apps"], alpha=1, lambda=grid, thresh=1e-12,nfolds = 5,standar
lambda.best = mod.lasso$lambda.min
print(lambda.best)
```

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

The best is0.01

```
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) -471.39372069
## (Intercept)    .
## PrivateYes  -491.04485135
## Accept         1.57033288
## Enroll        -0.75961467
## Top10perc     48.14698891
## Top25perc    -12.84690694
## F.Undergrad    0.04149116
## P.Undergrad    0.04438973
## Outstate      -0.08328388
```

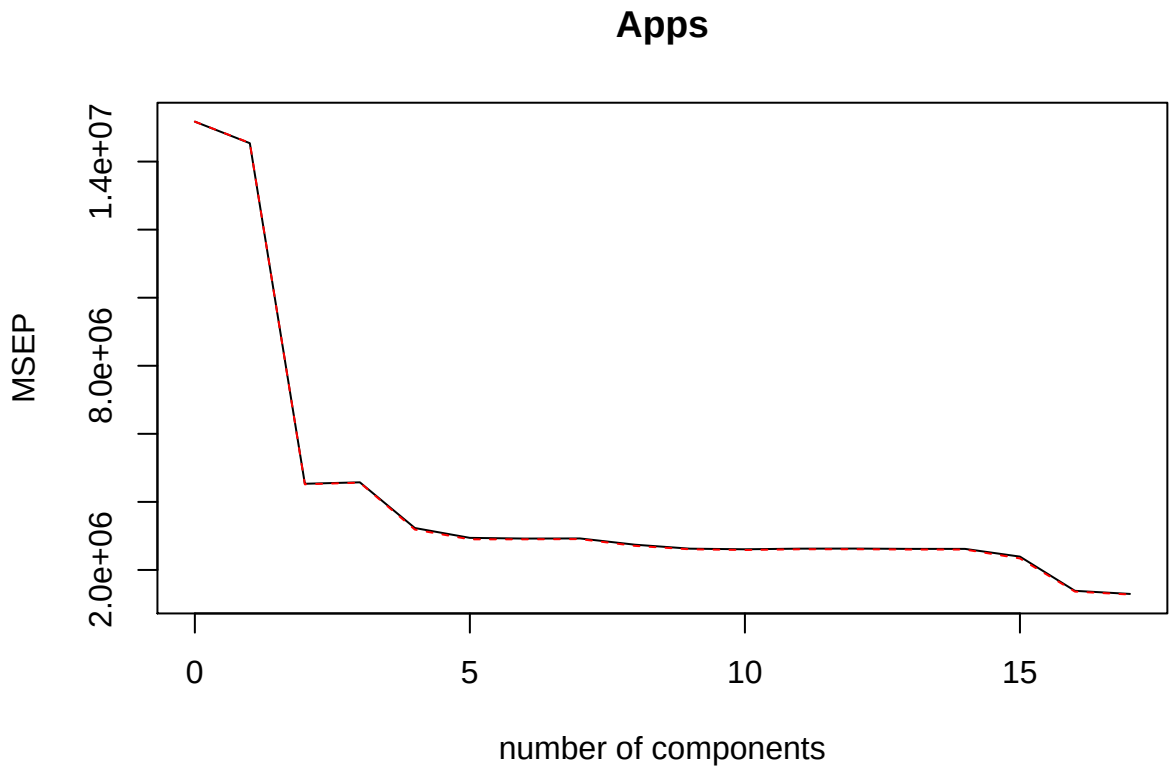
```
## Room.Board     0.14943472
## Books          0.01532293
## Personal       0.02909954
## PhD           -8.39597537
## Terminal      -3.26800340
## S.F.Ratio     14.59298267
## perc.alumni   -0.04404771
## Expend         0.07712632
## Grad.Rate      8.28950241
```

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

The MSE is 1.1078321×10^6, which is smaller than ridge regression and linear regression.

(e).

```
library(pls)
set.seed(1)
pcr.fit = pcr(Apps ~ ., data = train, scale = TRUE, validation = "CV")
validationplot(pcr.fit, val.type="MSEP")
```

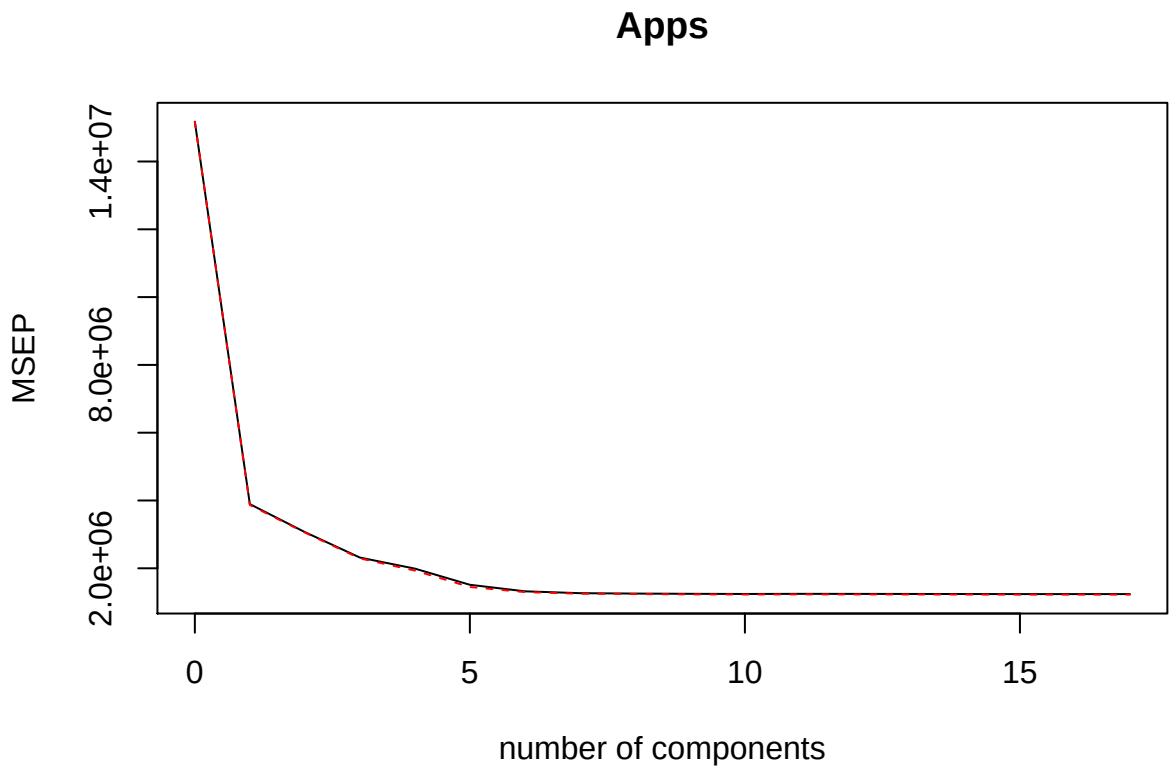



```
pcr.pred = predict(pcr.fit, test, ncomp=17)
pcr.mse <- mean((test[, "Apps"] - c(pcr.pred))^2)
```

M = 17 and the MSE for PCR is 1.2616304×10^6.

(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.

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




```
pls.pred = predict(pls.fit, test, ncomp=15)
pls.mse <- mean((test[, "Apps"] - c(pcr.pred))^2)
```

M = 15 and the MSE for PCR is 1.2616304×10^6.

(g). Comment on the results obtained. How accurately can we pre- dict the number of college applications received? Is there much difference among the test errors resulting from these five ap- proaches?

```
tot  <- sum((test$Apps - mean(test$Apps))^2)
data.frame(method = c("lm", "Ridge", "Lasso", "PCR", "PLS"),
           test_MSE = c(lm.mse, ridge.mse, lasso.mse, pcr.mse, pls.mse),
           test_R2 = c(1 - sum((test$Apps - lm.pred)^2) / tot,
                       1 - sum((test$Apps - ridge.pred)^2) / tot,
                       1 - sum((test$Apps - lasso.pred)^2) / tot,
                       1 - sum((test$Apps - pcr.pred)^2) / tot,
                       1 - sum((test$Apps - pls.pred)^2) / tot))
```

```
##   method test_MSE   test_R2
```

```
## 1     lm  1261630 0.9134458
## 2  Ridge  1261598 0.9134480
## 3  Lasso  1107832 0.9239971
## 4    PCR  1261630 0.9134458
## 5    PLS  1261630 0.9134496
```

The R-square value are all above 0.9 and their MSE are very close to each other