2.

(a) Suppose we have nth observations.

If we are not including the jth obs. then no. of obs. left $= n-1$

To Probability [First bootstrap is not jth] $= \frac{n-1}{n}$

(b). Probability that 2nd bootstrap ob is not the jth from original.

Using same concept, it would be $\frac{n-1}{n}$.

(c). For all the bootstrap sample, any observation not being in it would be same.

Since there are n observation.

so, $\left(\frac{n-1}{n}\right) \times \left(\frac{n-1}{n}\right) \times \ldots \left(\frac{n-1}{n}\right)$ n times

$$\left(\frac{n-1}{n}\right)^n = \left(1-\frac{1}{n}\right)^n$$

(d) for $n = 5$, Probability that jth observation is in the bootstrap is probability of not having it in any of the sample.

$$= 1 - \left(1-\frac{1}{5}\right)^5 \simeq 0.67232$$

(e) Similarly for, $n = 100$.

$$= 1 - \left(1 - \frac{1}{100}\right)^{100} \approx 0.63397$$

(f) for $n = 10000$

$$= \left(1 - \left(1 - \frac{1}{10000}\right)\right)^{10000} \approx 0.63231$$

Chapter 6

3 We estimate the regression coefficient in a linear regression by minimizing

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq s$$

(a) As we increases s from 0, the training error will monotonically decreases and will try to include all the variables in the ~~sub~~ model.

(b). Test RSS will decrease initially and then eventually start increasing in U shape. As s increases, model is getting more flexible so, after optimality, test error starts decreasing again.

(c). Variance will ~~initially~~ increase monotonically as more and more variables are included.

(e). squared Bias will decrease steadily as model is getting more flexible.

(f) Irreducible error can never be reduced or increased. It remains constant because its a trait of true function.

ECG HW 3

Code

This is an R Markdown (http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.

Hide

```
plot(cars)
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.

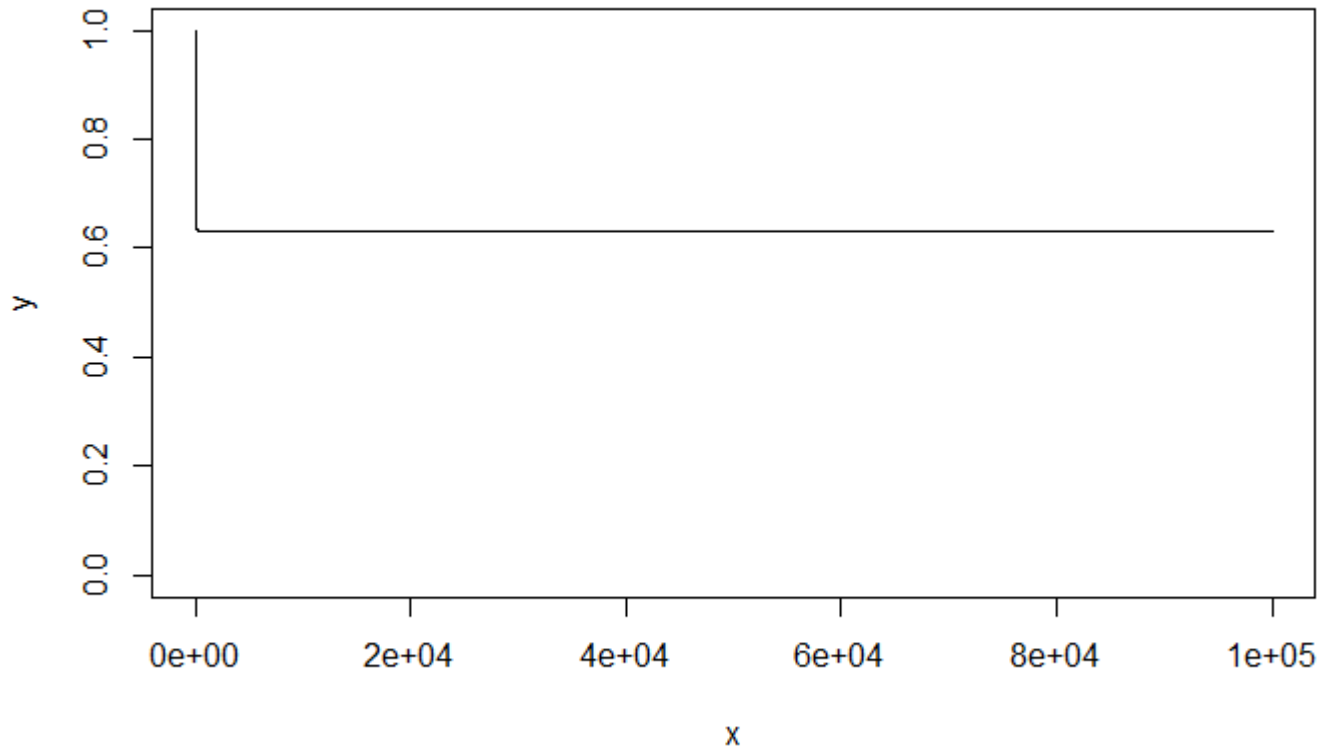
When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

2.g. Create a plot that displays, for each integer value of n from 1 to 100, 000, the probability that the jth observation is in the bootstrap sample. Comment on what you observe.

Hide

```
prob_jth = function(n){ (1-(1-1/n)^n) }
x = 1:100000
y = sapply(x,prob_jth)
plot(x,y,type='l',ylim=c(0.0,1.00))
```



We can see that probability is almost constant slightly above 0.6

h. We will now investigate numerically the probability that a bootstrap sample of size n = 100 contains the jth observation. Here j = 4. We repeatedly create bootstrap samples, and each time we record whether or not the fourth observation is contained in the bootstrap sample.

Hide

```
store=rep (NA , 10000)
for (i in 1:10000) {
store[i]=sum(sample (1:100 , rep =TRUE)==4) >0
}
mean(store)
```

```
[1] 0.6434
```

We can see that mean is 0.6434 and so, our values above were intuitive and were somewhat above 0.6

8. We will now perform cross-validation on a simulated data set.

a. Generate a simulated data set as follows:

Hide

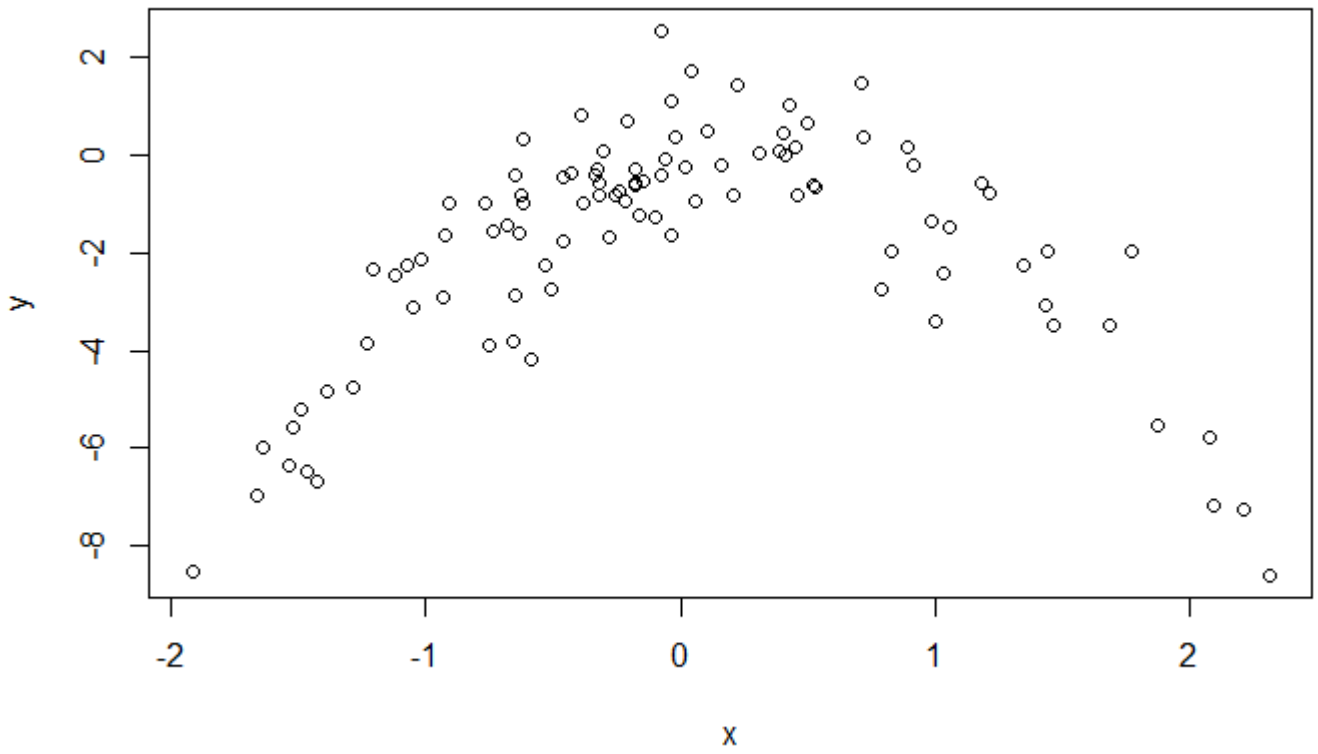
```
set.seed (1)
y=rnorm (100)
x=rnorm (100)
y=x-2* x^2+ rnorm (100)
```

In this example, n is genrated y values, p is generated x values. the equation is y=x-x^2+error

b.Create a scatterplot of X against Y . Comment on what you find.

Hide

```
plot(x,y)
```



c. Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

i. Y = ??0 + ??1X + error
ii. Y = ??0 + ??1X + ??2X2 + error
iii. Y = ??0 + ??1X + ??2X2 + ??3X3 + error
iv. Y = ??0 + ??1X + ??2X2 + ??3X3 + ??4X4 + error Note you may find it helpful to use the data.frame() function to create a single data set containing both X and Y

Since, these are polynomial models, we can use for loop do this task at once

Hide

```
library(boot)
data=data.frame(y,x)

set.seed(34)

for(i in 1:4)
  print(cv.glm(data = data,glm(y~poly(x,i),data = data))$delta[1])
```

So, unsing cross validation, our error are more for models having degree of polynomial different than quadratic giving a sense that, true fucntion is closer to quadratic

d. Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?

Hide

```
library(boot)
data=data.frame(y,x)

set.seed(50)

for(i in 1:4)
  print(cv.glm(data = data,glm(y~poly(x,i),data = data))$delta[1])
```

Yes, my result is same as in c because there is no sampling in LOOCV. so, cross validations are trained using the same observations.

e. Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.

The model with the smallest LOOCV error is the quadratic model. This is to be expected from the equation that defines Y as a second degree polynomial dependent on X.

f. Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

Hide

```
for(i in 1:4) {
  print(summary(glm(y~poly(x,i),data = data)))
}
```

```
Call:
glm(formula = y ~ poly(x, i), data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-7.3469  -0.9275   0.8028   1.5608   4.3974

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.8277     0.2362  -7.737 9.18e-12 ***
poly(x, i)    2.3164     2.3622   0.981    0.329
---
Signif. codes:  0 □***□ 0.001 □**□ 0.01 □*□ 0.05 □.□ 0.1 □ □ 1

(Dispersion parameter for gaussian family taken to be 5.580018)

    Null deviance: 552.21  on 99  degrees of freedom
Residual deviance: 546.84  on 98  degrees of freedom
AIC: 459.69

Number of Fisher Scoring iterations: 2


Call:
glm(formula = y ~ poly(x, i), data = data)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.89884  -0.53765   0.04135   0.61490   2.73607

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.8277     0.1032 -17.704   <2e-16 ***
poly(x, i)1   2.3164     1.0324   2.244   0.0271 *
poly(x, i)2 -21.0586     1.0324 -20.399   <2e-16 ***
---
Signif. codes:  0 □***□ 0.001 □**□ 0.01 □*□ 0.05 □.□ 0.1 □ □ 1

(Dispersion parameter for gaussian family taken to be 1.06575)

    Null deviance: 552.21  on 99  degrees of freedom
Residual deviance: 103.38  on 97  degrees of freedom
AIC: 295.11

Number of Fisher Scoring iterations: 2


Call:
glm(formula = y ~ poly(x, i), data = data)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
```

```
-2.87250  -0.53881   0.02862   0.59383   2.74350

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.8277     0.1037 -17.621   <2e-16 ***
poly(x, i)1   2.3164     1.0372   2.233   0.0279 *
poly(x, i)2 -21.0586     1.0372 -20.302   <2e-16 ***
poly(x, i)3  -0.3048     1.0372  -0.294   0.7695
---
Signif. codes:  0 □***□ 0.001 □**□ 0.01 □*□ 0.05 □.□ 0.1 □ □ 1

(Dispersion parameter for gaussian family taken to be 1.075883)

    Null deviance: 552.21  on 99  degrees of freedom
Residual deviance: 103.28  on 96  degrees of freedom
AIC: 297.02

Number of Fisher Scoring iterations: 2


Call:
glm(formula = y ~ poly(x, i), data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.8914  -0.5244   0.0749   0.5932   2.7796

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.8277     0.1041 -17.549   <2e-16 ***
poly(x, i)1   2.3164     1.0415   2.224   0.0285 *
poly(x, i)2 -21.0586     1.0415 -20.220   <2e-16 ***
poly(x, i)3  -0.3048     1.0415  -0.293   0.7704
poly(x, i)4  -0.4926     1.0415  -0.473   0.6373
---
Signif. codes:  0 □***□ 0.001 □**□ 0.01 □*□ 0.05 □.□ 0.1 □ □ 1

(Dispersion parameter for gaussian family taken to be 1.084654)

    Null deviance: 552.21  on 99  degrees of freedom
Residual deviance: 103.04  on 95  degrees of freedom
AIC: 298.78

Number of Fisher Scoring iterations: 2
```

All the models agree, on the 5% confidence level; The squared term is seen as significant in all the equation it is present, and the remaining terms are not seem as significant at this confidence level in any model.

9. We will now consider the Boston housing data set, from the MASS library.

a. Based on this data set, provide an estimate for the population mean of medv. Call this estimate ^??.

Hide

```
library(MASS)
attach(Boston)
mu=mean(medv)
print(mu)
```

```
[1] 22.53281
```

b. Provide an estimate of the standard error of ^??. Interpret this result. Hint: We can compute the standard error of the sample mean by dividing the sample standard deviation by the square root of the number of observations.

Hide

```
sterr = function(x,index){ sd(x[index])/sqrt(length(x[index])) }  # estimate of standard error.
print(sterr(medv))
```

```
[1] 0.4088611
```

So, our standard error is 0.4088611. On medv deviates from mean by this many standard deviation.

c. Now estimate the standard error of ^?? using the bootstrap. How does this compare to your answer from (b)?

Hide

```
library(boot)
set.seed(456)
boot(medv,function(x,index){mean(x[index])},R=1000)
```

```

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = medv, statistic = function(x, index) {
    mean(x[index])
}, R = 1000)


Bootstrap Statistics :
    original      bias    std. error
t1* 22.53281 -0.04298696    0.404878
```

Using Bootstrap is giving little different value as 0.404878 than sample estimate of 0.4088611

d. Based on your bootstrap estimate from (c), provide a 95% confidence interval for the mean of medv. Compare it to the results obtained using t.test(Boston$medv). Hint: You can approximate a 95% confidence interval using the formula [^?? ??? 2SE(^??), ^?? + 2SE(^??)].

We know that 95% Confidence interval is within 2 standard deviation of mean. Let's check the lower bound

Hide

```
mu-2*0.404878
```

```
[1] 21.72305
```

Let's check upper bound

Hide

```
mu+2*0.404878
```

```
[1] 23.34256
```

e. Based on this data set, provide an estimate, ^??med, for the median value of medv in the population.

Hide

```
print(median(medv))
```

```
[1] 21.2
```

f. We now would like to estimate the standard error of ^??med. Unfortunately, there is no simple formula for computing the standard error of the median. Instead, estimate the standard error of the median using the bootstrap. Comment on your findings.

Hide

```
boot(medv,function(data,index){median(data[index])},1000)
```

```

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = medv, statistic = function(data, index) {
    median(data[index])
}, R = 1000)


Bootstrap Statistics :
    original  bias    std. error
t1*     21.2  0.0061   0.3758178
```

We have a standard error of 0.3758178 in estimating the median.

g. Based on this data set, provide an estimate for the tenth percentile of medv in Boston suburbs. Call this quantity ^??0.1. (You can use the quantile() function.)

Hide

```
quantile(medv,.1)
```

```
  10%
12.75
```

So, 10th percentile of medv is 12.75

h. Use the bootstrap to estimate the standard error of ^??0.1. Comment on your findings.

Hide

```
boot(medv,function(data,index){quantile(medv[index],.1)},R=1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = medv, statistic = function(data, index) {
    quantile(medv[index], 0.1)
}, R = 1000)


Bootstrap Statistics :
    original  bias    std. error
t1*    12.75  0.0144   0.5025537
```

We can see that std error of 10th percentile is 0.5025537 slightly higher than median

Chapter 6.

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.

Let's import the data first

Hide

```
library(ISLR)
n=nrow(College)
```

Hide

```
x=c()
set.seed(100)
train=sample(1:n,n/2)
test=-train
```

We splitted the data into train test

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

Hide

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

```
     lm
1355557
```

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

Hide

```
grid = 10 ^ seq(4, -2, length=100)
train.mat = model.matrix(Apps~., data=College[train,])
test.mat = model.matrix(Apps~., data=College[test,])
```

Hide

```
library(glmnet)
ridge.cv=cv.glmnet(x=train.mat,y=College[train,"Apps"],alpha=0,thresh=1e-12,lambda = grid)
ridge.fit=glmnet(x=train.mat,y=College[train,"Apps"],alpha=0,lambda = ridge.cv$lambda.min)
ridge.pred=predict(ridge.fit,newx=test.mat )
ridge.coef=predict(ridge.fit,type='coefficients',s=ridge.cv$lambda.min)
ridge.cv$lambda.min
```

```
[1] 14.17474
```

Hide

```
ridge.coef
```

```
19 x 1 sparse Matrix of class "dgCMatrix"
                       1
(Intercept) -5.698205e+02
(Intercept)  .
PrivateYes  -5.936518e+02
Accept       1.280631e+00
Enroll      -4.716949e-01
Top10perc    4.794493e+01
Top25perc   -1.219088e+01
F.Undergrad  9.839294e-02
P.Undergrad  7.080416e-02
Outstate    -7.050049e-02
Room.Board   3.114784e-01
Books        4.586375e-02
Personal     2.040331e-03
PhD         -7.258500e+00
Terminal    -2.507214e+00
S.F.Ratio   -1.510205e+01
perc.alumni -1.442947e+01
Expend       6.324409e-02
Grad.Rate    1.116685e+01
```

Hide

```
x=c(x,ridge=mean( (ridge.pred-College[test,2])^2))
x
```

```
     lm   ridge
1355557 1408910
```

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

Hide

```
lasso.cv=cv.glmnet(train.mat,y=College[train,"Apps"],alpha=1,lambda=grid,thresh=1e-12)
lasso.fit=glmnet(x=as.matrix(College[train,-c(1,2)]),y=College[train,2],alpha=1,lambda = lasso.cv$lambda.min)
lasso.coef=predict(lasso.fit,type='coefficients',s=lasso.cv$lambda.min)
lasso.cv$lambda.min
```

```
[1] 16.29751
```

Hide

```
lasso.coef
```

```
17 x 1 sparse Matrix of class "dgCMatrix"
                       1
(Intercept) -1.364330e+03
Accept       1.264287e+00
Enroll       .
Top10perc    3.850869e+01
Top25perc   -4.595076e+00
F.Undergrad  3.380132e-02
P.Undergrad  7.214142e-02
Outstate    -8.197778e-02
Room.Board   2.618713e-01
Books        .
Personal     .
PhD         -2.243368e+00
Terminal     .
S.F.Ratio    .
perc.alumni -1.614927e+01
Expend       6.392972e-02
Grad.Rate    6.941991e+00
```

Hide

```
lasso.pred=predict(ridge.fit,newx=test.mat)
x=c(x,lasso=mean( (lasso.pred-College[test,2])^2))
x
```

```
     lm   ridge   lasso
1355557 1408910 1408910
```

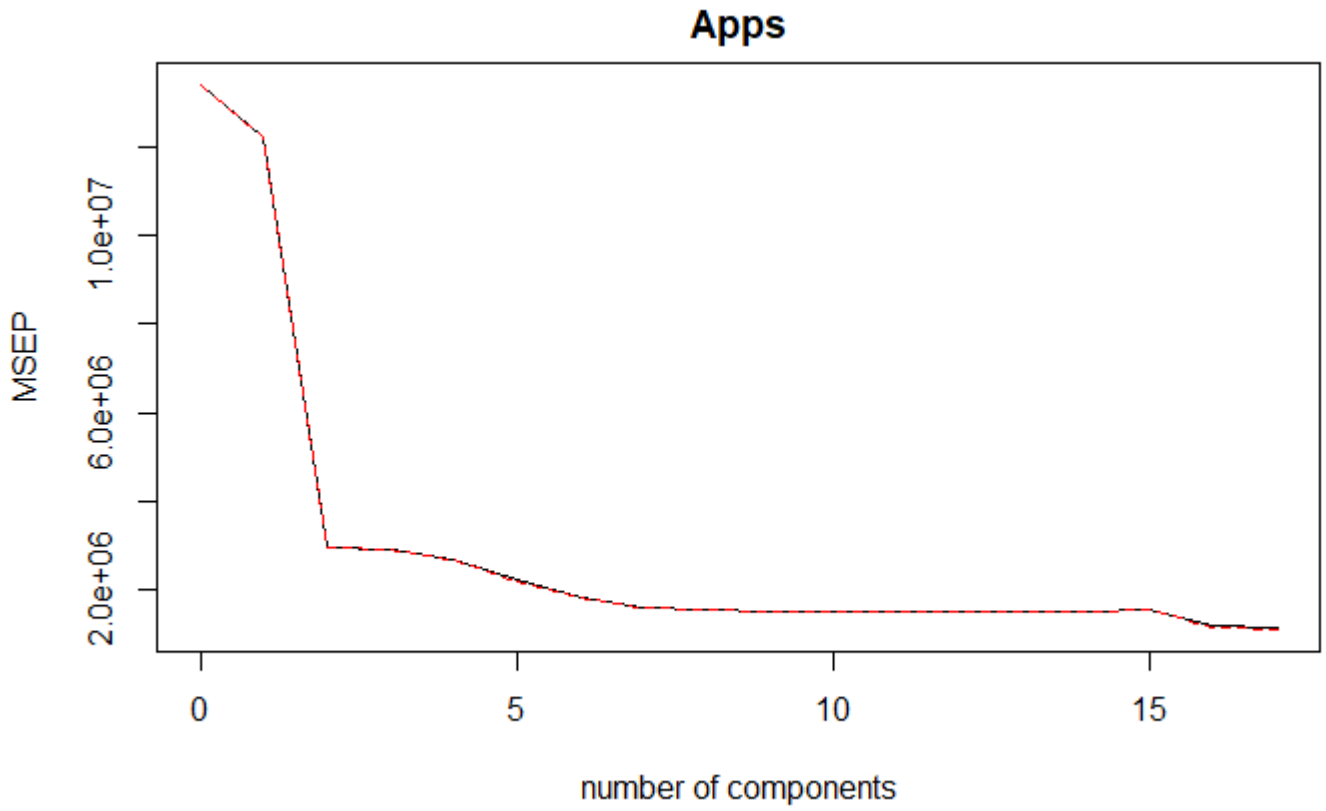
e. Fit a PCR model on the training set, with M chosen by crossvalidation. Report the test error obtained, along with the value of M selected by cross-validation.

Hide

```
library(pls)
```

Hide

```
set.seed(2)
pcr.fit=pcr(Apps~.,data=College,scale=TRUE,validation='CV',subset=train)
validationplot(pcr.fit,val.type="MSEP")
```



Hide

```
pcr.fit$ncomp
```

```
[1] 17
```

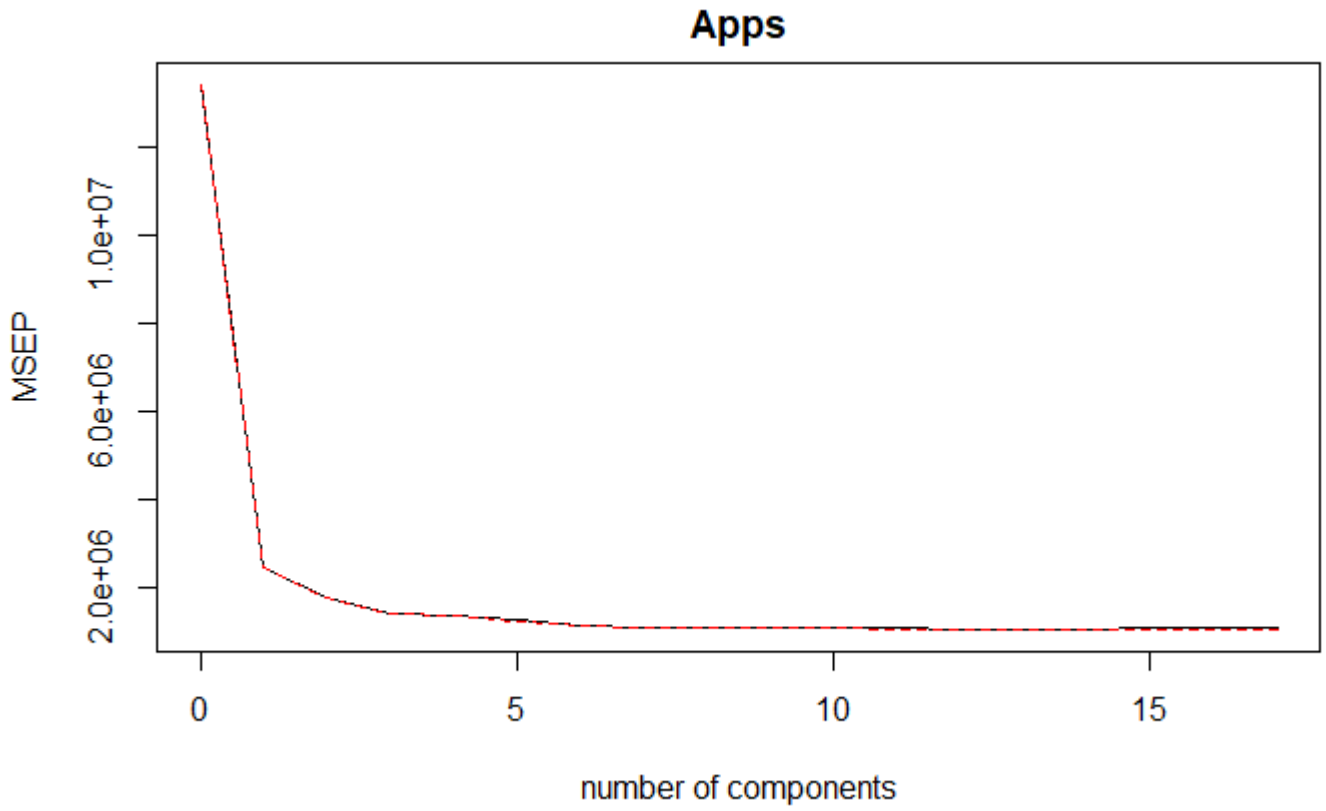
Hide

```
pcr.pred=predict(pcr.fit,newdata = College[test,-c(2)],ncomp = 16)
x=c(x,pcr=mean( (pcr.pred-College[test,2])^2))
```

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

Hide

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



Hide

```
pls.fit$ncomp
```

```
[1] 17
```

Hide

```
pls.pred=predict(pls.fit,newdata = College[test,-c(2)],ncomp = 10)
x=c(x,pls=mean( (pls.pred-College[test,2])^2))
```

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?

Hide

```
sort(x)
```

```
     lm     pls   ridge   lasso     pcr 
1355557 1365337 1408910 1408910 1531268 
```

We can see here that best performing model is linear model with least error and worst is PCR model

Hide

```
summary(lm.fit)
```

```
Call:
lm(formula = Apps ~ ., data = College[train, ])

Residuals:
    Min      1Q  Median      3Q     Max
-2842.8  -487.6   -90.9   347.3  5224.9

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.967e+02  5.513e+02  -0.901 0.368171
PrivateYes  -5.574e+02  1.896e+02  -2.941 0.003482 **
Accept       1.342e+00  6.902e-02  19.451  < 2e-16 ***
Enroll      -6.678e-01  2.465e-01  -2.709 0.007070 **
Top10perc    5.059e+01  7.358e+00   6.876 2.64e-11 ***
Top25perc   -1.422e+01  6.140e+00  -2.316 0.021121 *
F.Undergrad  1.097e-01  3.888e-02   2.821 0.005039 **
P.Undergrad  7.432e-02  3.893e-02   1.909 0.057031 .
Outstate    -7.834e-02  2.741e-02  -2.858 0.004501 **
Room.Board   3.032e-01  6.717e-02   4.514 8.56e-06 ***
Books        2.648e-02  3.577e-01   0.074 0.941029
Personal     8.379e-03  9.253e-02   0.091 0.927894
PhD         -7.400e+00  6.474e+00  -1.143 0.253776
Terminal    -2.347e+00  7.030e+00  -0.334 0.738667
S.F.Ratio   -1.447e+01  1.823e+01  -0.794 0.427878
perc.alumni -1.319e+01  5.674e+00  -2.325 0.020606 *
Expend       6.334e-02  1.725e-02   3.671 0.000277 ***
Grad.Rate    1.124e+01  4.067e+00   2.764 0.005989 **
---
Signif. codes:  0 □***□ 0.001 □**□ 0.01 □*□ 0.05 □.□ 0.1 □ □ 1

Residual standard error: 997.5 on 370 degrees of freedom
Multiple R-squared:  0.9288,    Adjusted R-squared:  0.9256
F-statistic: 284.1 on 17 and 370 DF,  p-value: < 2.2e-16
```

Hide

```
avg_apps=mean(College[,"Apps"])
1 - mean((College[test, "Apps"] - lm.pred)^2) /mean((College[test, "Apps"] - avg_apps)^2)
```

```
[1] 0.9180139
```

91% variance is explained by linear model.