HW3-resampling+model selection-Shuting Li

Shuting

2/10/2022

5.8

(a)

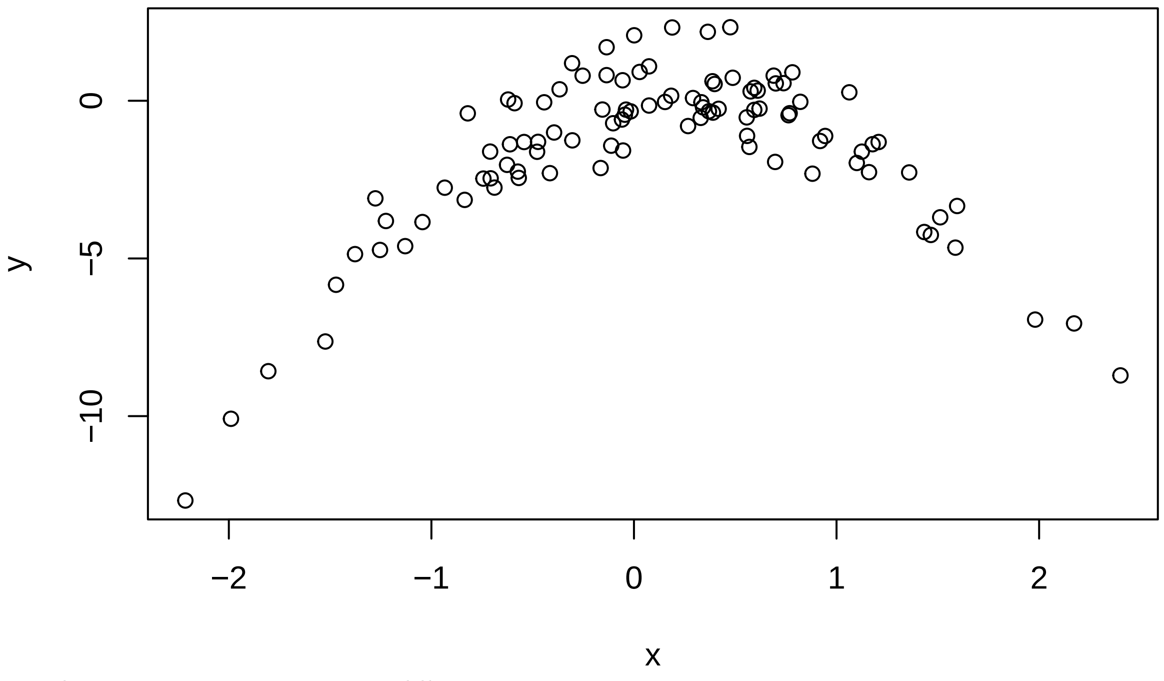
```r
set.seed(1)
x <- rnorm(100)
y <- x - 2 * x^2 + rnorm(100)
```

Answer: n is 100, p is 2. Model is

$$y = x - 2x^2 + \epsilon$$

(b)

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



Answer:
It is quadratic plot, points concentrate on middle.

(c)

```r
library(boot)
Data <- data.frame(x, y)

set.seed(1)
glm.fit.i = glm(y ~ x)
cv.glm(Data, glm.fit.i)$delta[1]
```

Answer:

```
## [1] 7.288162
```

```r
glm.fit.ii = glm(y ~ poly(x,2))
cv.glm(Data, glm.fit.ii)$delta[1]
```

```
## [1] 0.9374236
```

```r
glm.fit.iii = glm(y ~ poly(x,3))
cv.glm(Data, glm.fit.iii)$delta[1]
```

```
## [1] 0.9566218
```

```r
glm.fit.iv = glm(y ~ poly(x,4))
cv.glm(Data, glm.fit.iv)$delta[1]
```

```
## [1] 0.9539049
```

(d)

```r

set.seed(2)
glm.fit.i = glm(y ~ x)
cv.glm(Data, glm.fit.i)$delta[1]
```

Answer:

```
## [1] 7.288162
```

```r
glm.fit.ii = glm(y ~ poly(x,2))
cv.glm(Data, glm.fit.ii)$delta[1]
```

```
## [1] 0.9374236
```

```r
glm.fit.iii = glm(y ~ poly(x,3))
cv.glm(Data, glm.fit.iii)$delta[1]
```

```
## [1] 0.9566218
```

```r
glm.fit.iv = glm(y ~ poly(x,4))
cv.glm(Data, glm.fit.iv)$delta[1]
```

```
## [1] 0.9539049
```

Exactly same with the output in (c). Because repeat LOOCV will not change the way to split test data.

(e)

Answer: Second model has the smallest error, it exactly is what we expected, because it has x and x^2 as predictors, same with the true model of data.

(f)

```
summary(glm.fit.iv)
```

Answer:

```
##
## Call:
## glm(formula = y ~ poly(x, 4))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0550  -0.6212  -0.1567   0.5952   2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(x, 4)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, 4)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, 4)3   0.26411    0.95905   0.275    0.784
## poly(x, 4)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

The p-value of x^3 and x^4 are not significant, so we believe x^3 and x^4 not have much influence on y. It agrees with the CV results.

6.2

(a)

Answer: For lasso, iii is right, because it adds penalty term which decreases the flexibility.

iii. Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

(b)

Answer: For ridge, iii is right, because it adds penalty term which decreases the flexibility.

(c)

Answer: For no-linear model, ii is right, because it allows power term of the model, like quadratic term, increases the flexibility.

ii. More flexible and hence will give improved prediction accu- racy when its increase in variance is less than its decrease in bias.

6.9

(a)

```
library(ISLR2)
```

Answer:

```
## Warning: package 'ISLR2' was built under R version 4.1.2
```

```
sum(is.na(College))
```

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

```
#dim(College)
set.seed(1)
train <- sample(1:777, 380,replace = FALSE)
College.train <- College[train,]
College.test <- College[-train,]
```

(b)

```
lm.fit <- lm(Apps~.,data=College.train)
lm.pred <- predict(lm.fit, newdata = College.test)
(lm.testerror <- mean((College.test$Apps-lm.pred)^2))
```

Answer:

```
## [1] 1129629
```

(c)

```
library(glmnet)
```

Answer:

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

```
## Loaded glmnet 4.1-3
```

```
x.train.mat = model.matrix(Apps~., data=College.train)[,-1]
x.test.mat = model.matrix(Apps~., data=College.test)[,-1]
y.train = College.train$Apps
y.test = College.test$Apps

grid = 10 ^ seq(4, -4, length=100)
set.seed(2)
ridge.cv = cv.glmnet(x.train.mat, y.train, alpha=0, lambda = grid) # if do not set grid, lambda is not
#plot(ridge.cv)
(lambda.best = ridge.cv$lambda.min)
```

```
## [1] 1e-04
```

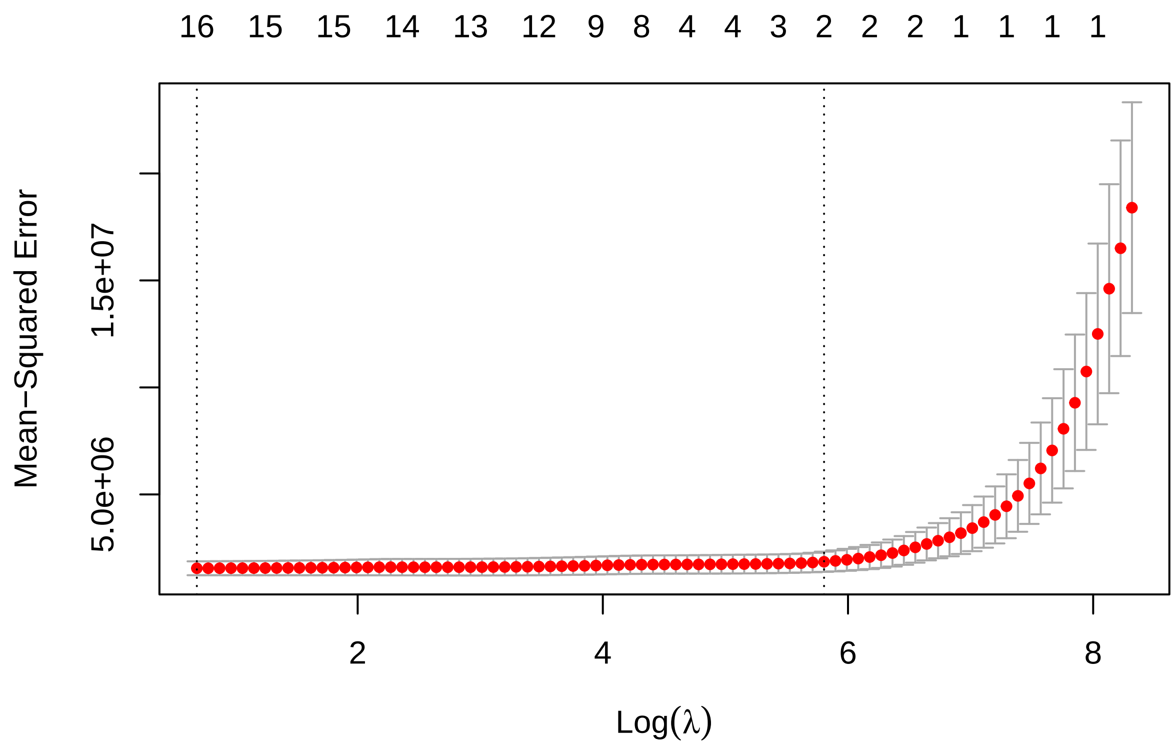
```
ridge.mod <- glmnet(x.train.mat, y.train, alpha=0)
ridge.pred <- predict(ridge.mod, newx=x.test.mat, s=lambda.best)
(ridge.testerror <- mean((y.test - ridge.pred)^2))
```

```
## [1] 967992.5
```

Best lambda is 1e-04, test error is 967992.5.

(d)

```
set.seed(2)
lasso.cv = cv.glmnet(x.train.mat, y.train, alpha=1)
plot(lasso.cv)
```



Answer:

```
(lambda.best = lasso.cv$lambda.min)
```

```
## [1] 1.989514
```

```
#########predict######
lasso.mod <- glmnet(x.train.mat, y.train, alpha=1)
lasso.pred <- predict(lasso.mod, newx=x.test.mat, s=lambda.best)
(lasso.testerror <- mean((y.test - lasso.pred)^2))
```

```
## [1] 1108793
```

```
#########coefficient######
x <- model.matrix(Apps~.,College)[,-1]
y <- College$Apps
out <- glmnet(x,y,alpha = 1)
lasso.coef <- predict(out,s=lambda.best,type = "coefficients")
```

The test error is 1108793, number of non-zero coefficient is 18.

(e)

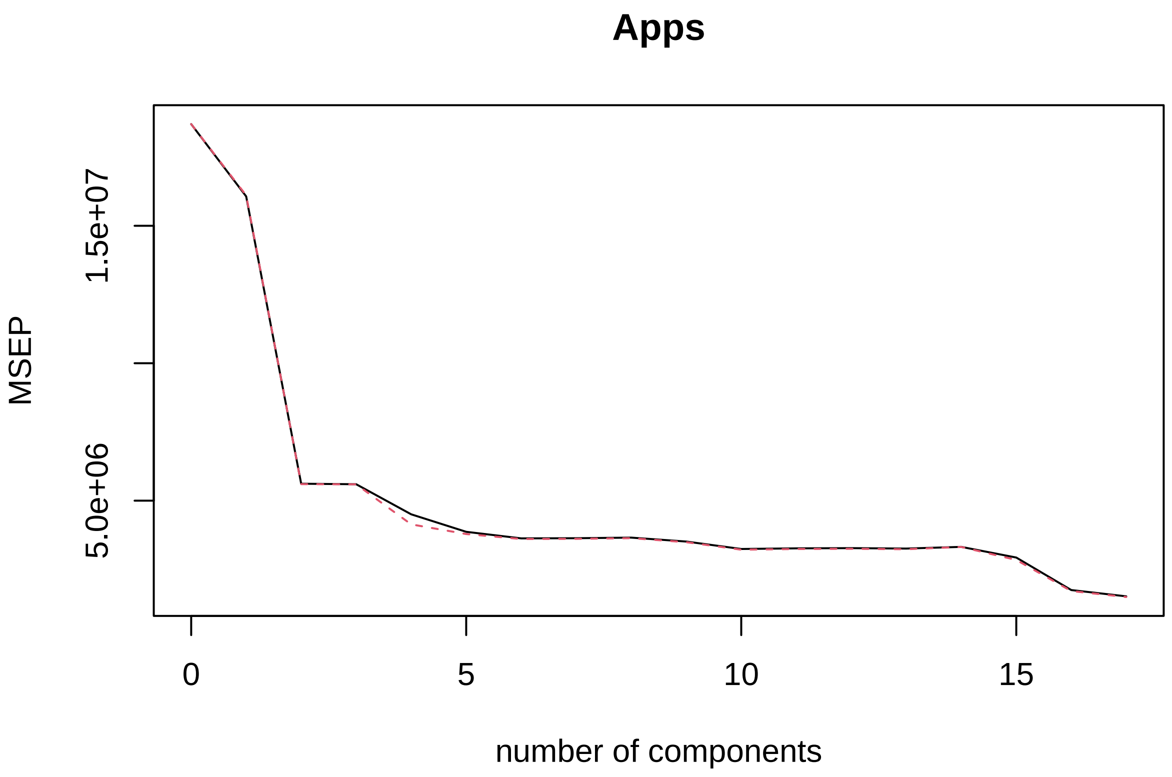
```
library(pls)
```

Answer:

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

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

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



```
pcr.pred = predict(pcr.fit, College.test, ncomp=10)
(pcr.testerror <- mean((y.test - pcr.pred)^2))
```

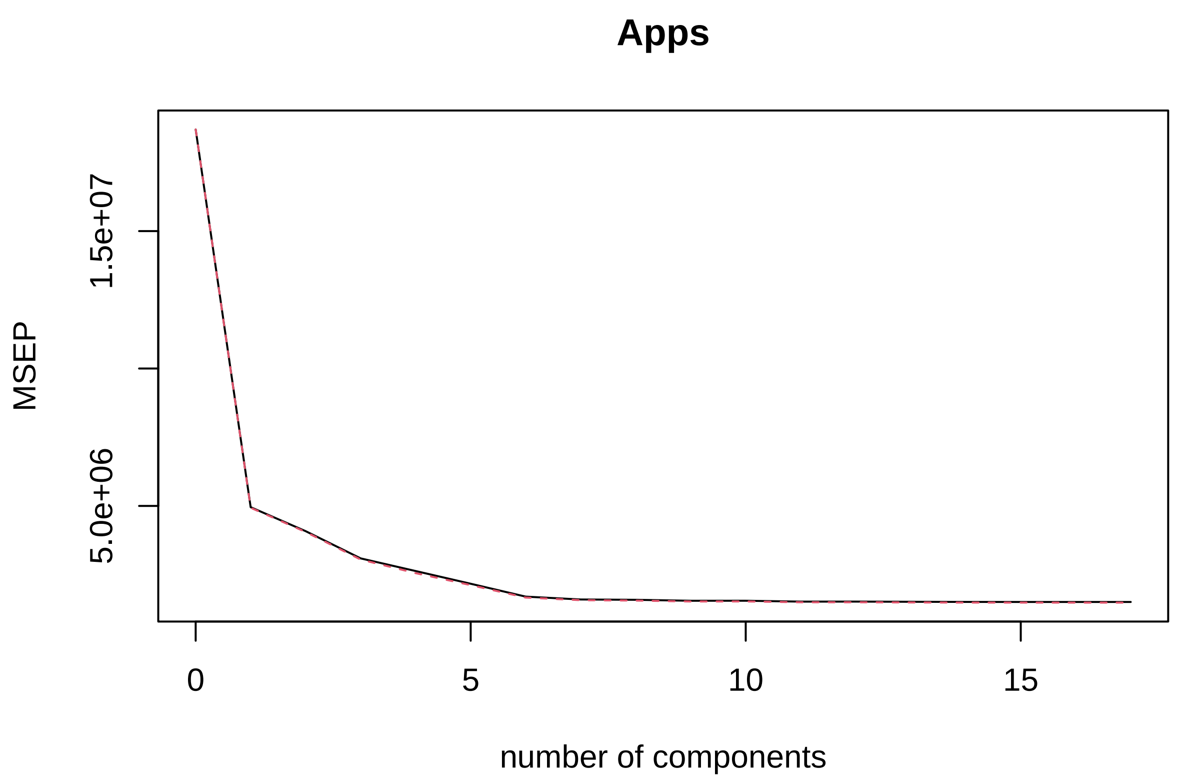
```
## [1] 1711146
```

The test error is 1711146.

(f)

```

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



Answer:

```r
pls.pred = predict(pls.fit, College.test, ncomp=15)
(pls.testerror <- mean((y.test - pls.pred)^2))
```

```
## [1] 1129734
```

The test error is 1129734.

(g)

```r

tss <- mean((y.test-mean(y.test))^2)
(lm.r2 <- 1-lm.testerror/tss)
```

Answer:

```
## [1] 0.9005125
```

```r
(ridge.r2 <- 1-ridge.testerror/tss)
```

```
## [1] 0.914748
```

```r
(lasso.r2 <- 1-lasso.testerror/tss)
```

```
## [1] 0.9023475
```

```r
(pcr.r2 <- 1-pcr.testerror/tss)
```

```
## [1] 0.8492977
```

```r
(pls.r2 <- 1-pls.testerror/tss)
```

```
## [1] 0.9005033
```

There is no much difference between these approaches.

6.10

(a)

```
set.seed(1)
p = 20
n = 1000
x = matrix(rnorm(n * p), n, p)
beta = sample(1:100,p)
beta[c(5,7,10)] = 0
eps = rnorm(p)
y = x %*% beta + eps
```

Answer:

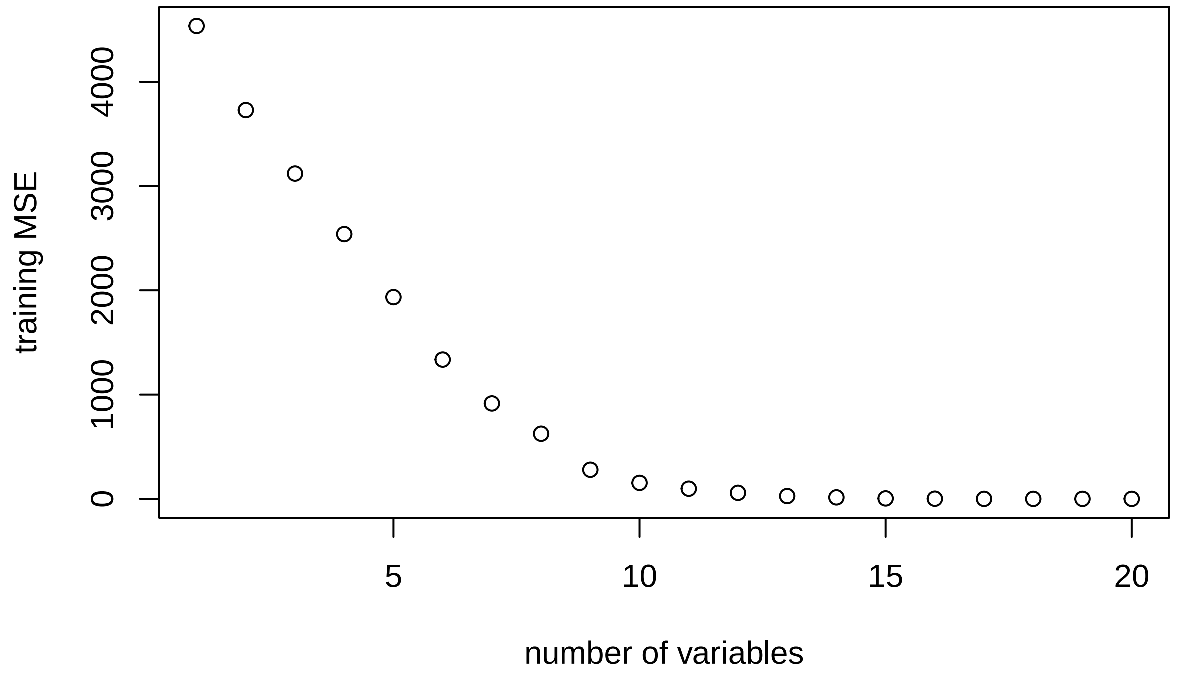
(b)

```
set.seed(1)
train <- sample(1:1000,100,replace = FALSE)
data <- data.frame(x,y)
train.data <- data[train,]
test.data <- data[-train,]
```

Answer:

(c)

```
library(leaps)
regfit.full = regsubsets(y ~ ., data = train.data, nvmax = p)
reg.summary <- summary(regfit.full)
plot(reg.summary$rss/1000,xlab="number of variables",ylab="training MSE")
```



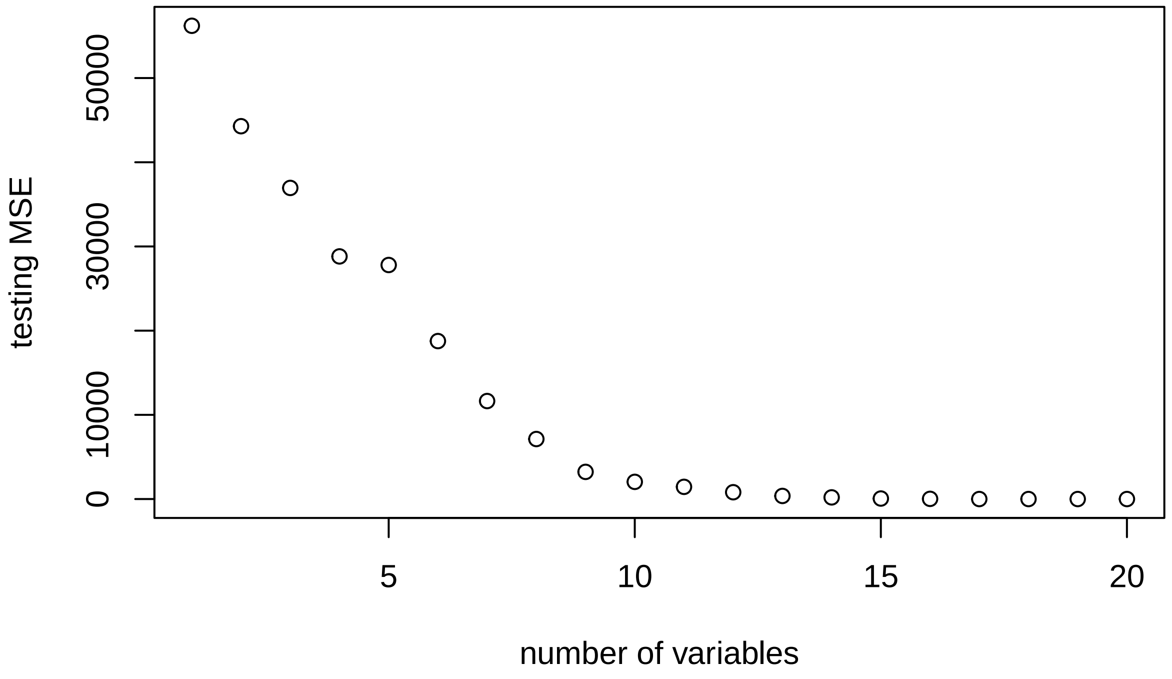
Answer:

```
minnum <- which.min(reg.summary$bic)
coef(regfit.full,18)
```

```
## (Intercept)          X1          X2          X3          X4          X6
##   0.2260659  81.0010471   9.9818069  66.0996285   3.9196738  73.8896038
##          X7          X8          X9         X11         X12         X13
##  -0.2945929  61.1168809  92.9715467  78.1053871  96.9834443  86.0819471
##         X14         X15         X16         X17         X18         X19
##  23.8635535   6.1149616  18.7800299  25.0859091  32.9185822  11.9813580
##         X20
##  84.8421572
```

(d)

```
val.error <- rep(NA,20)
test.mat <- model.matrix(y ~ ., data = test.data)
for (i in 1:20){
  coefi <- coef(regfit.full, id=i)
  pred=test.mat[,names(coefi)]%*%coefi
  val.error[i] <- mean((test.data$y-pred)^2)
}
minnum <- which.min(val.error)
mincoef <- coef(regfit.full,17)
plot(val.error,xlab="number of variables",ylab="testing MSE")
```



Answer:

(e)

```
which.min(val.error)
```

Answer:

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

Testing MSE will be minimum when model takes 17 coefficients. It agrees with the true form of data.

(f)

```
coef(regfit.full,17)
```

Answer:

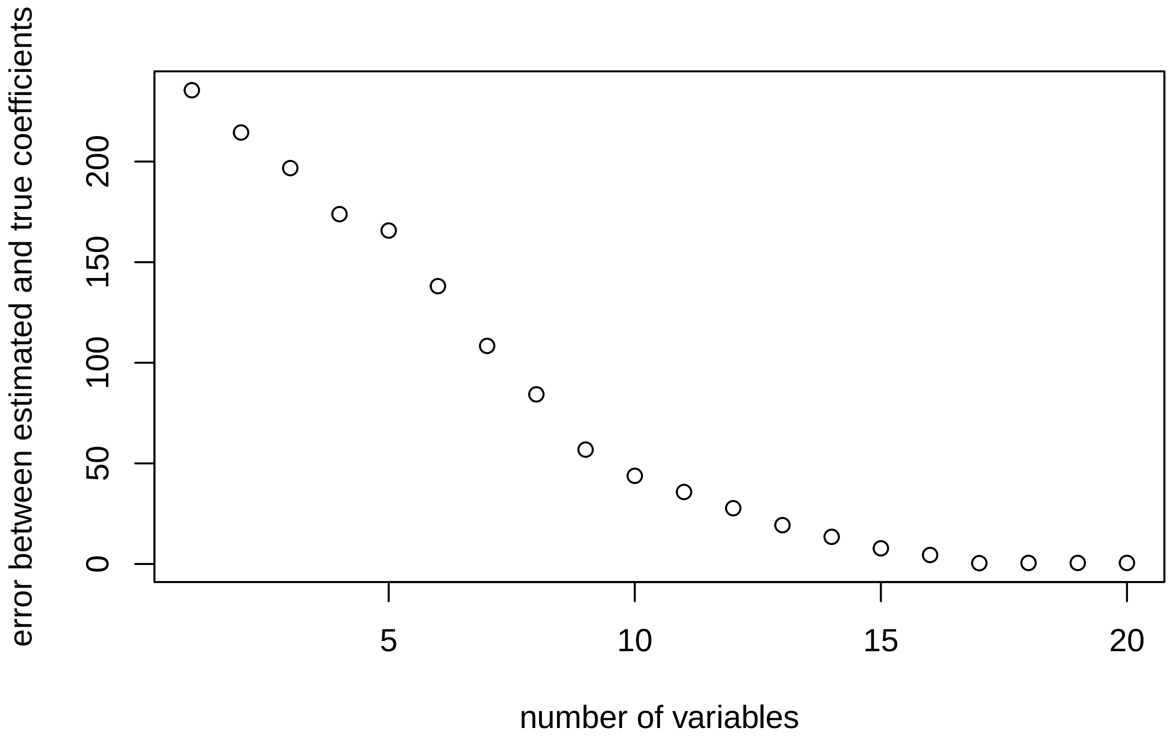
```
## (Intercept)          X1          X2          X3          X4          X6
##   0.2585713  81.0302464  10.0013874  66.1017857   3.9076418  73.8880298
##          X8          X9         X11         X12         X13         X14
##  61.0737205  92.9724785  78.0620397  96.9648818  86.0398570  23.8243248
##         X15         X16         X17         X18         X19         X20
##   6.1348546  18.8092550  25.0848027  32.9650467  11.9631277  84.8228862
```

The model with smallest test MSE has coefficients that similar to the true model.

(g)

```
coef.error <- rep(NA,20)
names(beta) <- paste0('X', 1:20)
for (r in 1:20){
  coefr <- coef(regfit.full, id=r)
  coef.error[r] <- sqrt(sum((beta[names(beta) %in% names(coefr)]-coefr[names(coefr) %in% names(beta)])^
}

plot(coef.error,xlab="number of variables",ylab="error between estimated and true coefficients")
```



Answer:
When number of variables increases, the estimated coefficients close into the true value, bias decreases.

In (d), when number of variables bigger than 17, the testing MSE increases, which means the amount of variance increases bigger than bias decreases.

6.11

(a)

```
library(MASS)
```

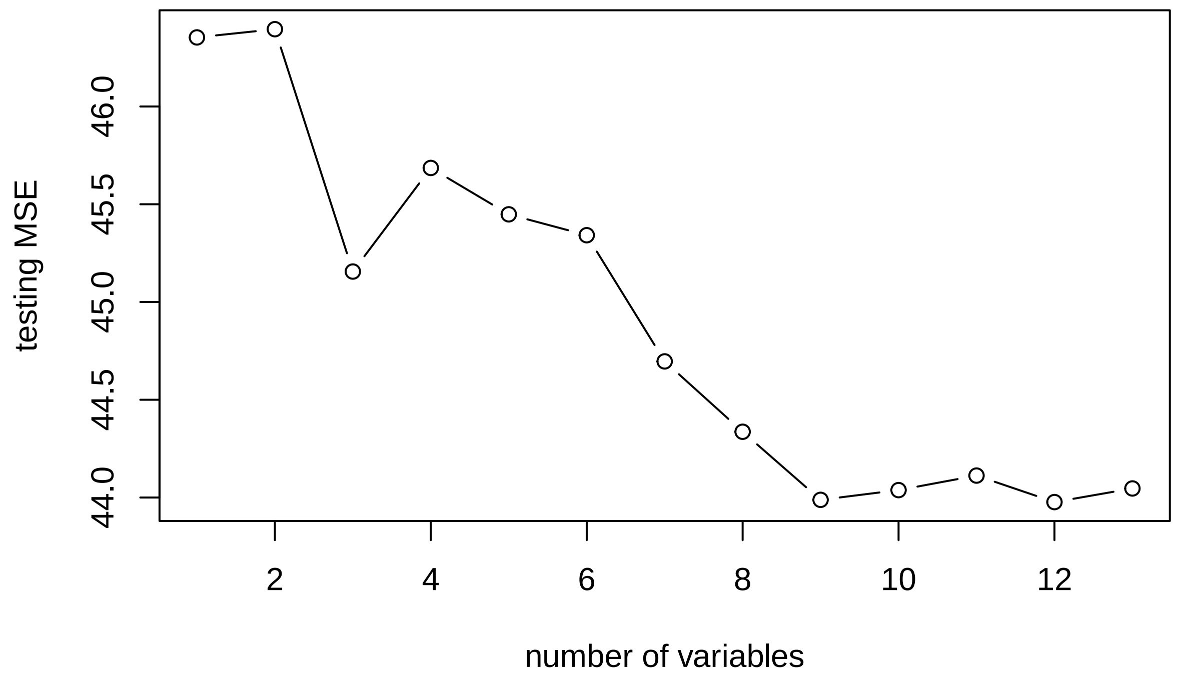
Answer:

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```
library(leaps)
library(glmnet)
library(pls)

predict.regsubsets = function(object, newdata, id, ...) {
    form = as.formula(object$call[[2]])
    mat = model.matrix(form, newdata)
    coefi = coef(object, id = id)
    mat[, names(coefi)] %*% coefi
}
#### test MSE with 10-fold in different subset selection #####
k = 10
p = ncol(Boston) - 1
folds = sample(rep(1:k, length = nrow(Boston)))
cv.errors = matrix(NA, k, p)
for (i in 1:k) {
    best.fit = regsubsets(crim ~ ., data = Boston[folds != i, ], nvmax = p)
    for (j in 1:p) {
        pred = predict(best.fit, Boston[folds == i, ], id = j)
        cv.errors[i, j] = mean((Boston$crim[folds == i] - pred)^2)
    }
}
rmse.cv = (apply(cv.errors, 2, mean))
plot(rmse.cv, type = "b",xlab="number of variables",ylab="testing MSE")
```



```
which.min(rmse.cv)
```

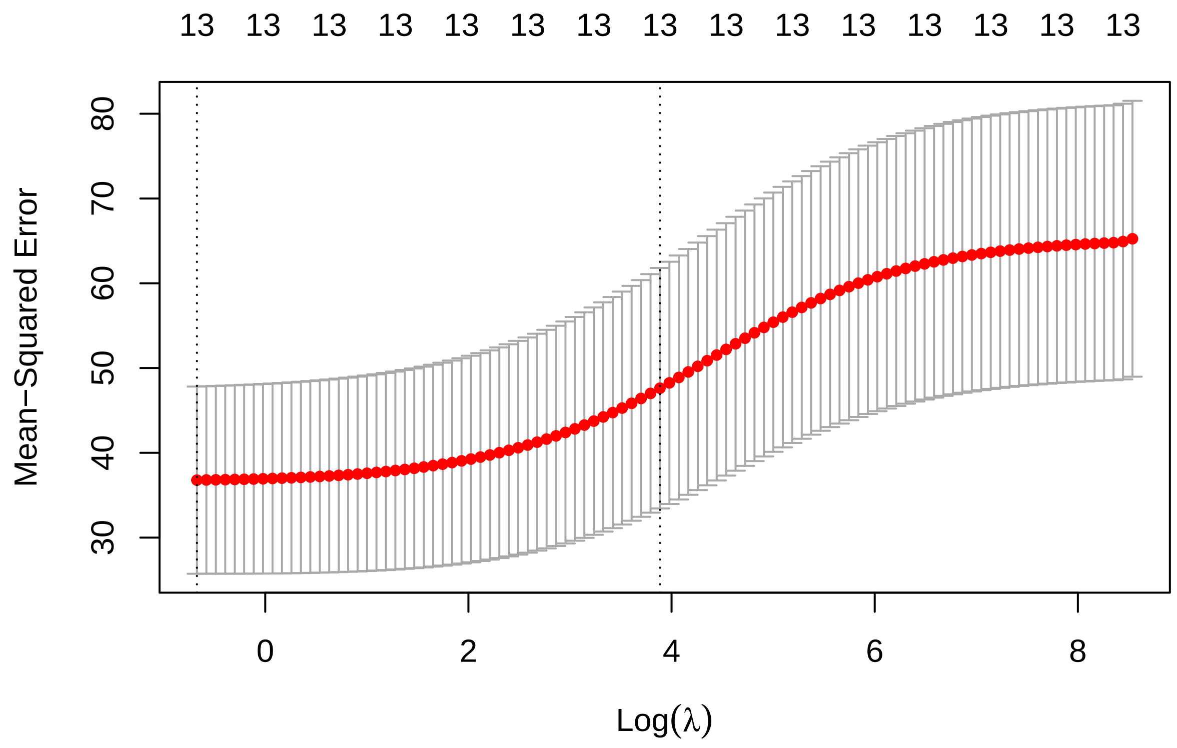
```
## [1] 12
```

```
(rmse.error <- rmse.cv[9])
```

```
## [1] 43.98813
```

```
##ridge
x = model.matrix(crim ~ . - 1, data = Boston)
y = Boston$crim
set.seed(1)
train <- sample(1:506,400,replace = FALSE)

cv.ridge = cv.glmnet(x[train,], y[train], type.measure = "mse", alpha = 0)
plot(cv.ridge)
```

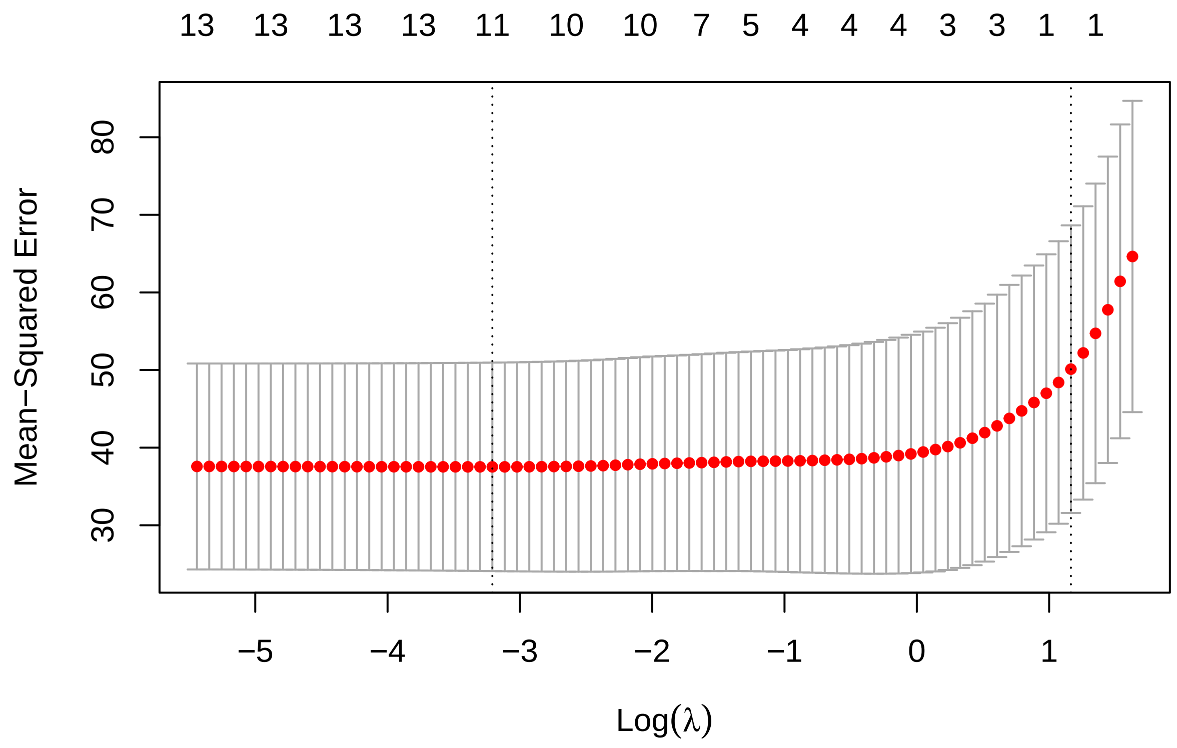


```
bestlam <- cv.ridge$lambda.min

ridge.mod = glmnet(x[train,], y[train], alpha = 0)
ridge.pred <- predict(ridge.mod, s=bestlam, newx = x[-train,])
(ridge.error <- mean((ridge.pred-y[-train])^2))
```

```
## [1] 68.44515
```

```
##lasso
cv.lasso = cv.glmnet(x[train,], y[train], type.measure = "mse", alpha = 1)
plot(cv.lasso)
```



```
bestlam <- cv.lasso$lambda.min

lasso.mod = glmnet(x[train,], y[train], alpha = 1)
lasso.pred <- predict(lasso.mod, s=bestlam, newx = x[-train,])
(lasso.error <- mean((lasso.pred-y[-train])^2))
```

```
## [1] 67.28498
```

```
##pcr
pcr.fit = pcr(crim ~ ., data = Boston, scale = TRUE, subset=train, validation = "CV")
#summary(pcr.fit) #13 comps
pcr.pred = predict(pcr.fit, x[-train,],ncomp=13)
(pcr.error = mean((pcr.pred-y[-train])^2))
```

```
## [1] 66.8352
```

Subset selection with 9 components has lowest testing MSE.

(b)

```

k = 10
p = ncol(Boston) - 1
folds = sample(rep(1:k, length = nrow(Boston)))
cv.errors = matrix(NA, k, p)
for (i in 1:k) {
    best.fit = regsubsets(crim ~ ., data = Boston[folds != i, ], nvmax = p)
    for (j in 1:p) {
        pred = predict(best.fit, Boston[folds == i, ], id = j)
        cv.errors[i, j] = mean((Boston$crim[folds == i] - pred)^2)
    }
}
```

```r
rmse.cv = (apply(cv.errors, 2, mean))
(rmse.error <- rmse.cv[9])
```

Answer:

```
## [1] 42.95908
```

When we use subset selection model with 9 components we will get lowest testing MSE.

(c)

Answer: The model I chose only involve 9 features, because it shows lowest testing MSE.