Resampling and Regularization Homework

Pruthvi Bharadwaj

February 14, 2022

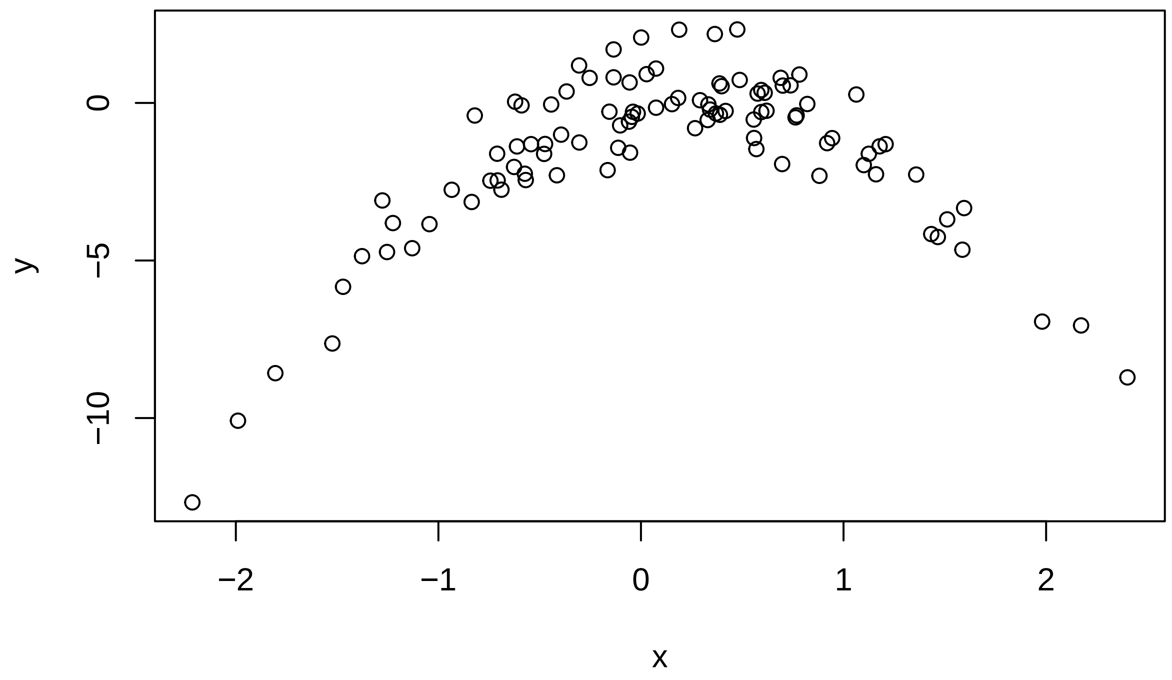
5.8. We will now perform cross-validation on a simulated data set. (a) Generate a simulated data set as follows:

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

In this data set, what is n and what is p? Write out the model used to generate the data in equation form. (b) Create a scatterplot of X against Y . Comment on what you find.

The model used to generate the data is $y = (x - 2)x^2 + \epsilon$ In this data set, n = 100 and p = 2.

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



From the scatterplot, we see a clear non-linear relationship between x and y. The plot looks more like an inverted parabola implying a quadratic relationship between x and y. And, the range of x seems to be approximately from -2 to 2.

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

i. Y $= \beta_0 + \beta_1 X + \epsilon$
ii. Y $= \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$
iii. Y $= \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$
iv. Y $= \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$. Note you may find it helpful to use the data.frame() function to create a single data set containing both X and Y .

```
# i)
set.seed(1)
df <- data.frame(x,y)
fit1 <- glm(y ~ x, data = df)
print(paste0("LOOCV error for i): ", cv.glm(df, fit1)$delta[1]))
```

```
## [1] "LOOCV error for i): 7.28816160667281"
```

```
# ii)
fit2 <- glm(y ~ poly(x,2), data = df)
print(paste0("LOOCV error for ii): ", cv.glm(df, fit2)$delta[1]))
```

```
## [1] "LOOCV error for ii): 0.937423637615552"
```

```
# iii)
fit3 <- glm(y ~ poly(x,3), data = df)
print(paste0("LOOCV error for iii): ", cv.glm(df, fit3)$delta[1]))
```

```
## [1] "LOOCV error for iii): 0.95662183010894"
```

```
# iv)
fit4 <- glm(y ~ poly(x,4), data = df)
print(paste0("LOOCV error for iv): ", cv.glm(df, fit4)$delta[1]))
```

```
## [1] "LOOCV error for iv): 0.953904892744804"
```

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

```
set.seed(99)
df <- data.frame(x,y)
# i)
fit1 <- glm(y ~ x, data = df)
print(paste0("LOOCV error for i): ", cv.glm(df, fit1)$delta[1]))
```

```
## [1] "LOOCV error for i): 7.28816160667281"
```

```
# ii)
fit2 <- glm(y ~ poly(x,2), data = df)
print(paste0("LOOCV error for ii): ", cv.glm(df, fit2)$delta[1]))
```

```
## [1] "LOOCV error for ii): 0.937423637615552"
```

```
# iii)
fit3 <- glm(y ~ poly(x,3), data = df)
print(paste0("LOOCV error for iii): ", cv.glm(df, fit3)$delta[1]))
```

```
## [1] "LOOCV error for iii): 0.95662183010894"
```

```
# iv)
fit4 <- glm(y ~ poly(x,4), data = df)
print(paste0("LOOCV error for iv): ", cv.glm(df, fit4)$delta[1]))
```

```
## [1] "LOOCV error for iv): 0.953904892744804"
```

The results in (c) and (d) are the same because LOOCV trains the model on all the observations except one. Therefore, each time, the model is trained with the same set of observations for each cross validation set.

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

Model ii) has the least LOOCV error. This is expected as we saw in (b) that x and y share a quadratic relationship.

(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?

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

```
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -9.5161  -0.6800   0.6812   1.5491   3.8183
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -1.550      0.260  -5.961 3.95e-08 ***
## poly(x, i)     6.189      2.600   2.380   0.0192 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for gaussian family taken to be 6.760719)
##
##     Null deviance: 700.85  on 99  degrees of freedom
## Residual deviance: 662.55  on 98  degrees of freedom
## AIC: 478.88
##
## Number of Fisher Scoring iterations: 2
##
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9650  -0.6254  -0.1288   0.5803   2.2700
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.5500     0.0958  -16.18  < 2e-16 ***
## poly(x, i)1   6.1888     0.9580    6.46 4.18e-09 ***
## poly(x, i)2 -23.9483     0.9580  -25.00  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9178258)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  89.029  on 97  degrees of freedom
## AIC: 280.17
##
## Number of Fisher Scoring iterations: 2
##
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9765  -0.6302  -0.1227   0.5545   2.2843
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09626 -16.102  < 2e-16 ***
## poly(x, i)1   6.18883    0.96263   6.429 4.97e-09 ***
## poly(x, i)2 -23.94830    0.96263 -24.878  < 2e-16 ***
## poly(x, i)3   0.26411    0.96263   0.274    0.784
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9266599)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  88.959  on 96  degrees of freedom
## AIC: 282.09
```

```
##
## Number of Fisher Scoring iterations: 2
##
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## 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, i)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, i)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, i)3   0.26411    0.95905   0.275    0.784
## poly(x, i)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 results from the least squares model for the four models above are consistent with the LOOCV results. In model i) and ii), all the coefficients are statistically significant. But, in models iii) and iv), the coefficients of $X^3 and X^4$ are not statistically significant, implying that y is second degree polynomial dependent on x.

6.2. For parts (a) through (c), indicate which of i. through iv. is correct. Justify your answer. (a) The lasso, relative to least squares, is: i. More flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance. ii. More flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias. iii. Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance. iv. Less flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias.

The lasso is: iii) less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance. This is because the lasso restricts the number of predictors by shrinking some of them to be exactly zero. This reduced flexibility results in increase in bias but a decrease in variance. Therefore, when the increase in bias is less than the decrease in variance, lasso will have better prediction accuarcy relative to least squares.

(b) Repeat (a) for ridge regression relative to least squares.

Similar to the lasso, ridge regression also is: iii) less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

(c) Repeat (a) for non-linear methods relative to least squares.

Since non-linear methods are more flexible relative to least squares, ii) is the correct answer here. They are more flexible and hence will give better prediction accuracy when the increase in variance is less than the decrease in bias.

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.

```
data(College)
set.seed(1)
data1 <- sample(1:dim(College)[1], dim(College)[1] / 2)
data2 <- -data1
train <- College[data1, ]
test <- College[data2, ]
```

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

```
fit1 <- lm(Apps ~ ., data = train)
pred.lm <- predict(fit1, test)
mean((pred.lm - test$Apps)^2)
```

```
## [1] 1135758
```

Using least squares on the training set, the test error obtained is 1135758.

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

```
train.mat <- model.matrix(Apps ~., data = train[ ,-1])
test.mat <- model.matrix(Apps ~., data = test[ ,-1])
set.seed(1)
cv.out <- cv.glmnet(train.mat, train$Apps, alpha = 0)
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 405.8404
```

```
pred.ridge <- predict(cv.out, s = bestlam, newx = test.mat)
mean((pred.ridge - test$Apps) ^2)
```

```
## [1] 1007688
```

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

```
set.seed(1)
cv.out <- cv.glmnet(train.mat, train$Apps, alpha = 1)
bestlam.lasso <- cv.out$lambda.min
bestlam.lasso
```

```
## [1] 2.165848
```

```r
pred.lasso <- predict(cv.out, s = bestlam.lasso, newx = test.mat)
mean((pred.lasso - test$Apps) ^2)
```

```
## [1] 1140473
```

```r
lasso.coef<- predict(cv.out, s = bestlam.lasso, type = "coefficients")[1:18, ]

length(lasso.coef[lasso.coef != 0])
```

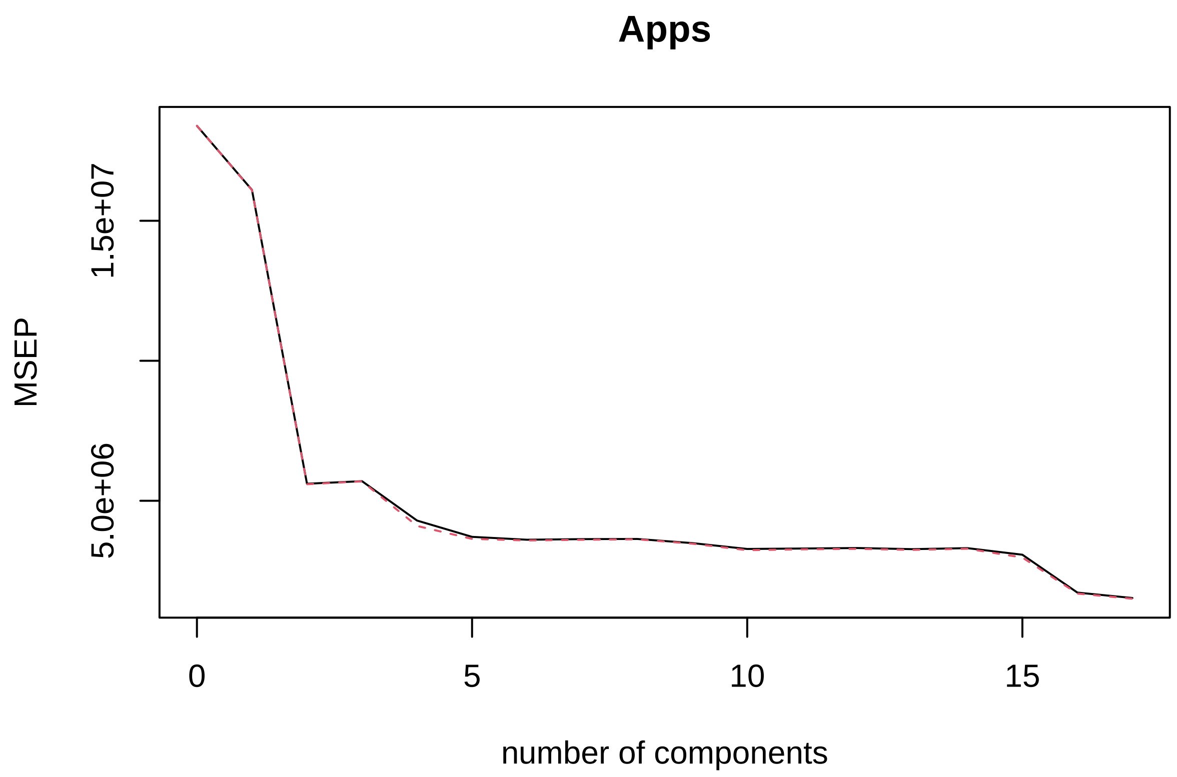
```
## [1] 16
```

(e)Fit a PCR 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.

```r
pcr.fit <- pcr(Apps ~ ., data = train, scale = TRUE, validation = "CV")
summary(pcr.fit)
```

```
## Data:    X dimension: 388 17
##  Y dimension: 388 1
## Fit method: svdpc
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            4288     4013     2368     2388     2072     1926     1900
## adjCV         4288     4012     2364     2386     2025     1907     1893
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1905     1907     1868      1811      1815      1820      1809
## adjCV     1899     1903     1862      1799      1807      1812      1801
##        14 comps  15 comps  16 comps  17 comps
## CV         1819      1753      1312      1236
## adjCV      1813      1725      1298      1225
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       32.20    57.78    65.31    70.99    76.37    81.27     84.8    87.85
## Apps    13.44    70.93    71.07    79.87    81.15    82.25     82.3    82.33
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       90.62     92.91     94.98     96.74     97.79     98.72     99.42
## Apps    83.38     84.76     84.80     84.84     85.11     85.14     90.55
##       16 comps  17 comps
## X        99.88    100.00
## Apps     93.42     93.89
```

```r
validationplot(pcr.fit, val.type = "MSEP")
```



The CV score is provided for each possible number of components, ranging from M=0 onwards. One can also plot the cross-validation scores using the "validationplot" function. We see that the smallest cross-validation error occurs when about M =17 components are used.

```
pred.pcr <- predict(pcr.fit, test.mat, ncomp = 5)
mean((pred.pcr - test$Apps)^2)
```

```
## [1] 1963819
```

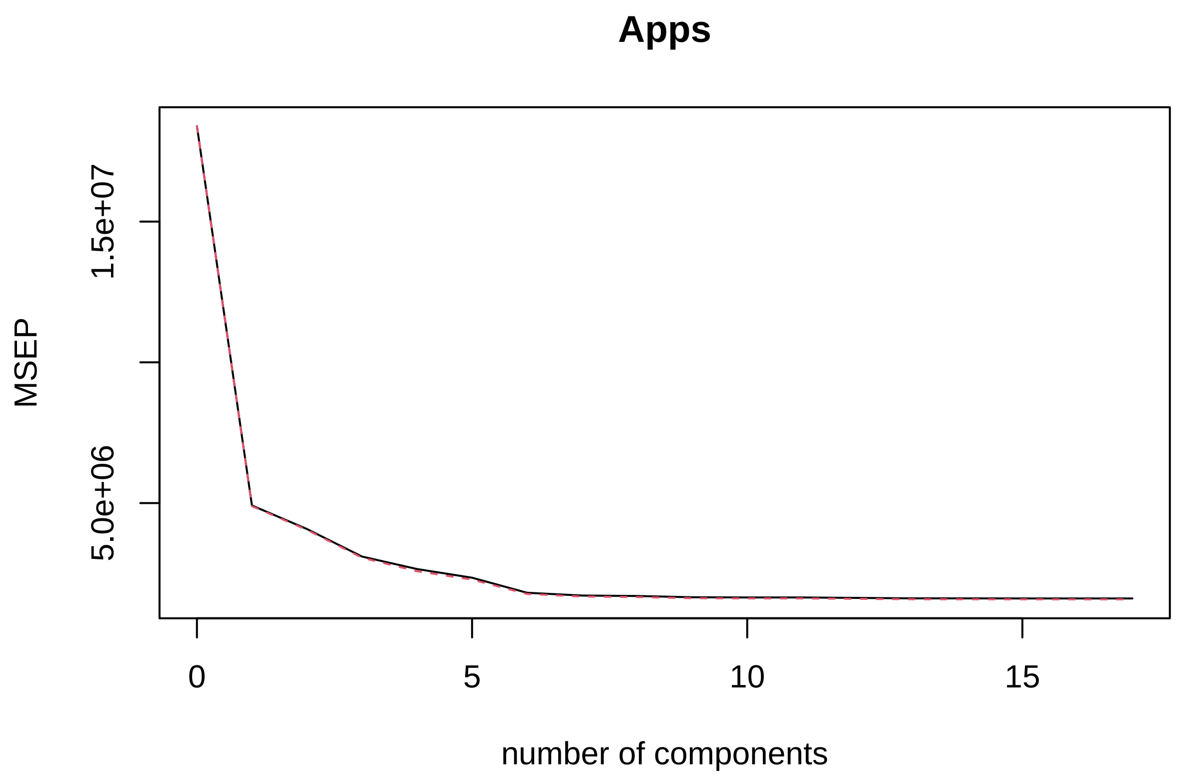
test set MSE is 1963819.

(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.

```
set.seed(1)
pls.fit <- plsr(Apps ~ ., data = train, scale = TRUE, validation = "CV")
summary(pls.fit)
```

```
## Data:    X dimension: 388 17
##  Y dimension: 388 1
## Fit method: kernelpls
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            4288     2217     2019     1761     1630     1533     1347
## adjCV         4288     2211     2012     1749     1605     1510     1331
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1309     1303     1286      1283      1283      1277      1271
## adjCV     1296     1289     1273      1270      1270      1264      1258
##        14 comps  15 comps  16 comps  17 comps
## CV         1270      1270      1270      1270
## adjCV      1258      1257      1257      1257
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       27.21    50.73    63.06    65.52    70.20    74.20    78.62    80.81
## Apps    75.39    81.24    86.97    91.14    92.62    93.43    93.56    93.68
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       83.29     87.17     89.15     91.37     92.58     94.42     96.98
## Apps    93.76     93.79     93.83     93.86     93.88     93.89     93.89
##       16 comps  17 comps
## X        98.78    100.00
## Apps     93.89     93.89
```

```
validationplot(pls.fit, val.type = "MSEP")
```



```
pred.pls <- predict(pls.fit, test.mat, ncomp = 10)
mean((pred.pls - test$Apps) ^2)
```

```
## [1] 1181808
```

The test MSE here in partial least squares is 1181808.

(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?

In order to compare the results obtained above, we need to compute the test R^2 for all models.

```
test.avg <- mean(test$Apps)
lm <- 1- mean((pred.lm - test$Apps)^2) / mean((test.avg - test$Apps)^2)
ridge <- 1- mean((pred.ridge - test$Apps)^2) / mean((test.avg - test$Apps)^2)
lasso <- 1- mean((pred.lasso - test$Apps)^2) / mean((test.avg - test$Apps)^2)
pcr <- 1- mean((pred.pcr - test$Apps)^2) / mean((test.avg - test$Apps)^2)
pls <- 1- mean((pred.pls - test$Apps)^2) / mean((test.avg - test$Apps)^2)
lm
```

```
## [1] 0.9015413
```

```
ridge
```

```
## [1] 0.9126437
```

```
lasso
```

```
## [1] 0.9011326
```

```
pcr
```

```
## [1] 0.8297569
```

```
pls
```

```
## [1] 0.8975493
```

So the test R^2 for the least squares is 0.9015413, the test R^2 for ridge regression is 0.9015558, the test R^2 for lasso regression is 0.9011326, the test R^2 for PCR is 0.8297569, and the test R^2 for PLS is 0.8975493. Except PCR, all models predict college applications with high accuracy.

6.10. We have seen that as the number of features used in a model increases, the training error will necessarily decrease, but the test error may not. We will now explore this in a simulated data set. (a) Generate a data set with p = 20 features, n = 1,000 observations, and an associated quantitative response vector generated according to the model Y = $X\beta + \epsilon$, where β has some elements that are exactly equal to zero.

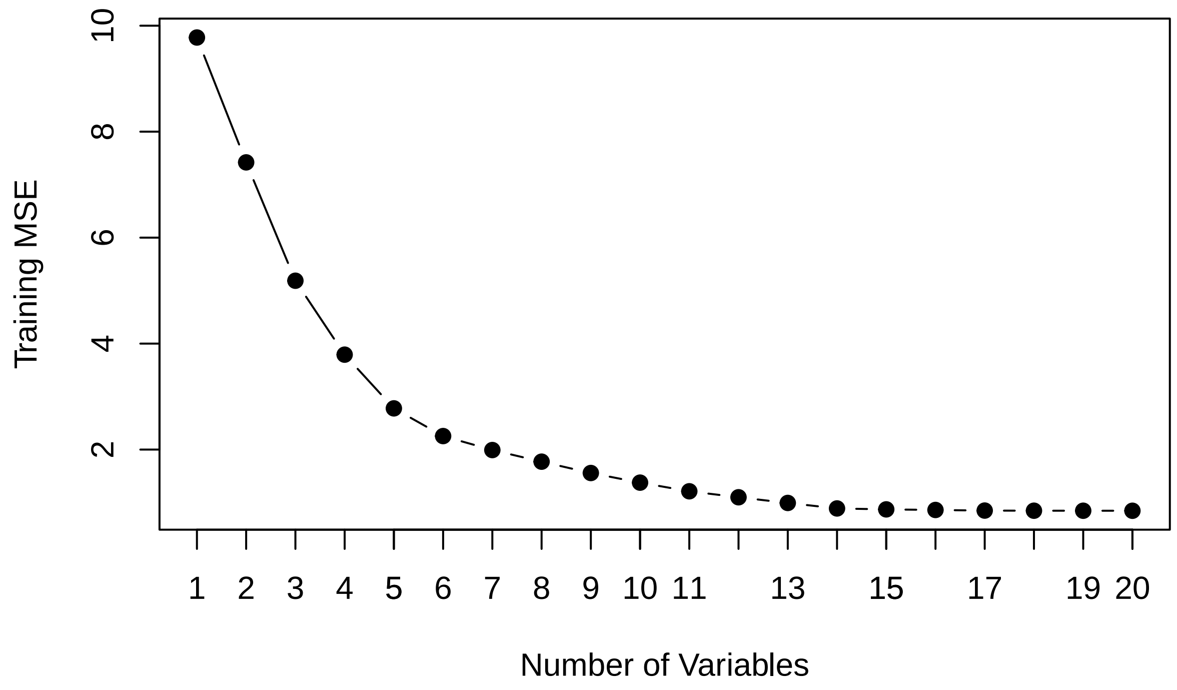
```
set.seed(9)
x <- matrix(rnorm(1000*20), 1000, 20)
b <- matrix(rnorm(20), 20, 1)
b[2] <- 0
b[5] <- 0
b[9] <- 0
b[14] <- 0
b[18] <- 0
err <- rnorm(1000)
y <- x%*%b + err
```

(b) Split your data set into a training set containing 100 observations and a test set containing 900 observations.

```
Data <- data.frame(x,y)
train <- Data[1:100,]
test <- Data[101:1000,]
```

(c) Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size.

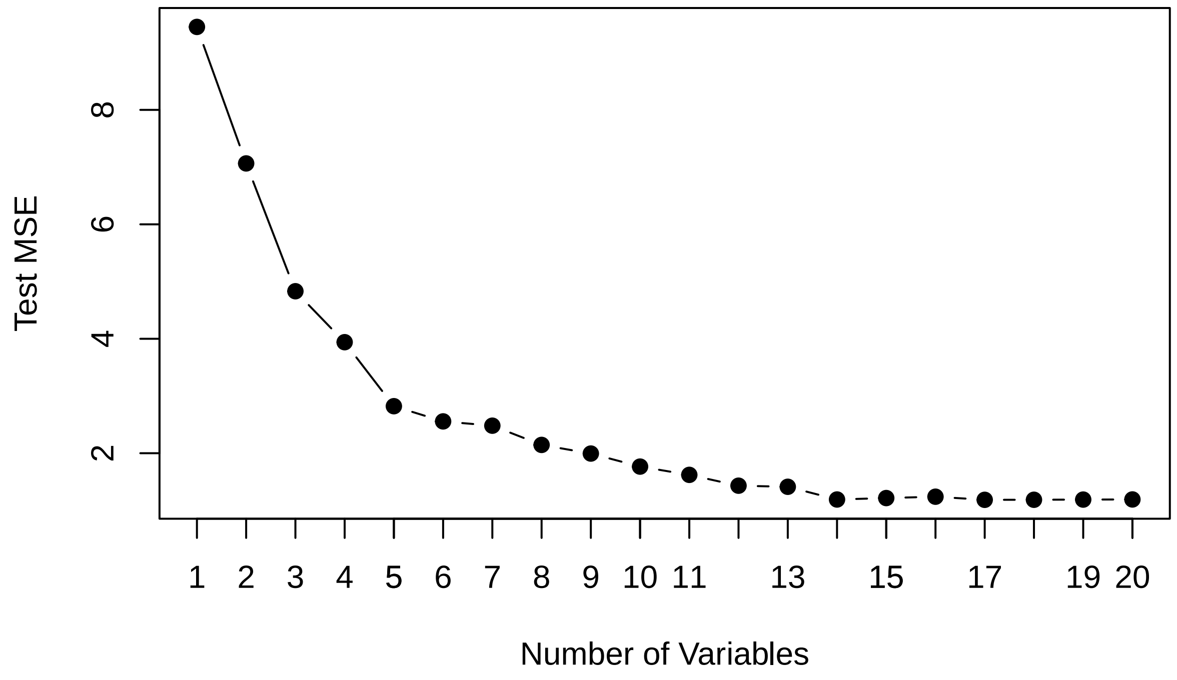
```
n <- 100
subset1 <- regsubsets(y ~., train, nvmax = 20)
plot((1/n)*summary(subset1)$rss, xlab = "Number of Variables", ylab = "Training MSE", type = "b", pch =
axis(1, at = seq(1, 20, 1))
```



(d) Plot the test set MSE associated with the best model of each size.

```
test.mat <- model.matrix(y ~., test, nvmax = 20)
errs <- rep(NA, 20)
for (i in 1:20) {
  coeff <- coef(subset1, id = i)
  pred <- test.mat[, names(coeff)] %*%coeff
  errs[i] <- mean((pred - test[,21])^2)
}
```

```
plot(errs, xlab = "Number of Variables", ylab = "Test MSE", type = "b", pch = 19)
axis(1, at = seq(1, 20, 1))
```



(e) For which model size does the test set MSE take on its minimum value? Comment on your results. If it takes on its minimum value for a model containing only an intercept or a model containing all of the features, then play around with the way that you are generating the data in (a) until you come up with a scenario in which the test set MSE is minimized for an intermediate model size.

```
which.min(errs)
```

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

The model with 17 variables has the smallest MSE.

(f) How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient values.

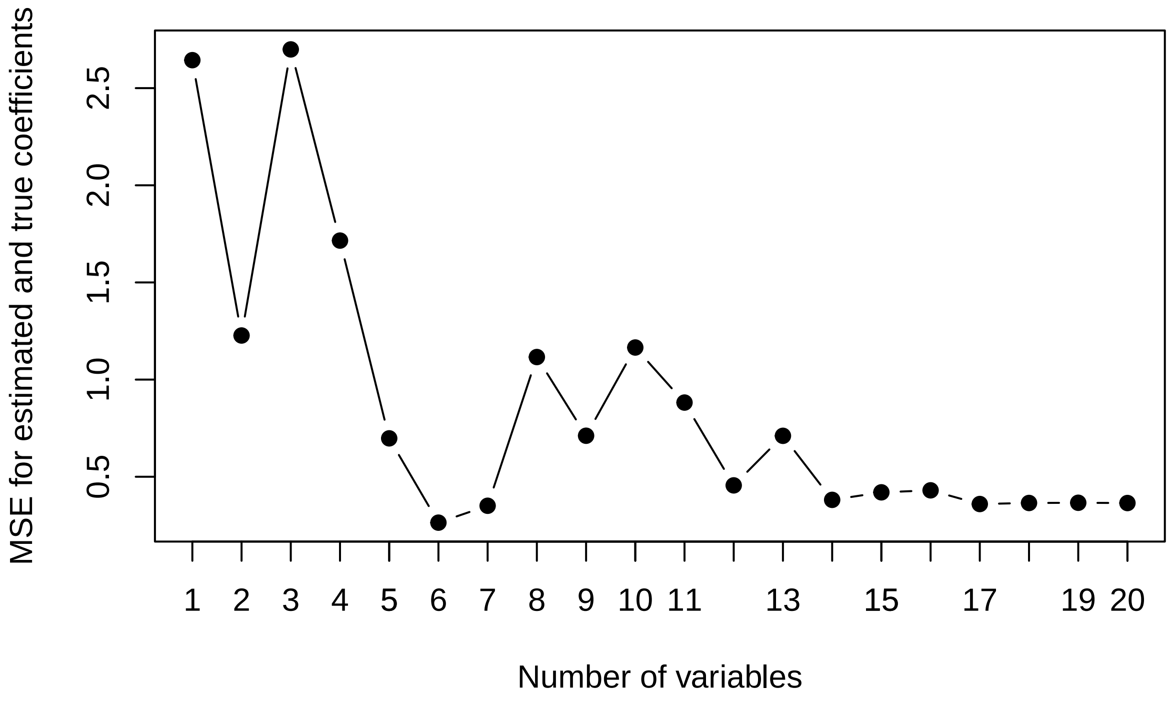
```
coef(subset1, which.min(errs))
```

```
## (Intercept)          X1          X2          X3          X4          X6
##   0.2055328  -0.1227184  -0.1237108  -0.4311793   0.3720967  -0.5984137
##          X7          X8         X10         X11         X12         X13
##  -0.4565580   1.4688580  -1.3445077   0.5176785  -1.6423076   0.4358270
##         X15         X16         X17         X18         X19         X20
##  -0.4302169  -0.3327596  -1.0936251   0.1259921  -1.0709751   0.5470251
```

The best model caught only three out of the five zero coefficients.

(g) Create a plot displaying $\sqrt{\Sigma_{j=1}^{p}}(\beta_j - \hat{\beta}_j^r)^2$ for a range of values of r, where $\hat{\beta}_j^r$ is the jth coefficient estimate for the best model containing r coefficients. Comment on what you observe. How does this compare to the test MSE plot from (d)?

```
errors <- rep(NA, 20)
x_colname <- colnames(x, do.NULL = FALSE, prefix = "X")
for (i in 1:20) {
  coeff <- coef(subset1, id = i)
  errors[i] <- sqrt(sum((b[x_colname %in% names(coeff)] - coeff[names(coeff) %in% x_colname])^2) + sum(
}
plot(errors, xlab = "Number of variables", ylab = "MSE for estimated and true coefficients", type = "b"
axis(1, at = seq(1, 20, 1))
```



From the above plot, it is seen that the model with 6 variables has the least error. This implies that the model that gives coefficient estimates close to the true parameter values need not necessarily be the model that has the least MSE, i.e., it is not necessarily the best model.

6.11. We will now try to predict per capita crime rate in the Boston data set. (a) Try out some of the regression methods explored in this chapter, such as best subset selection, the lasso, ridge regression, and PCR. Present and discuss results for the approaches that you consider.

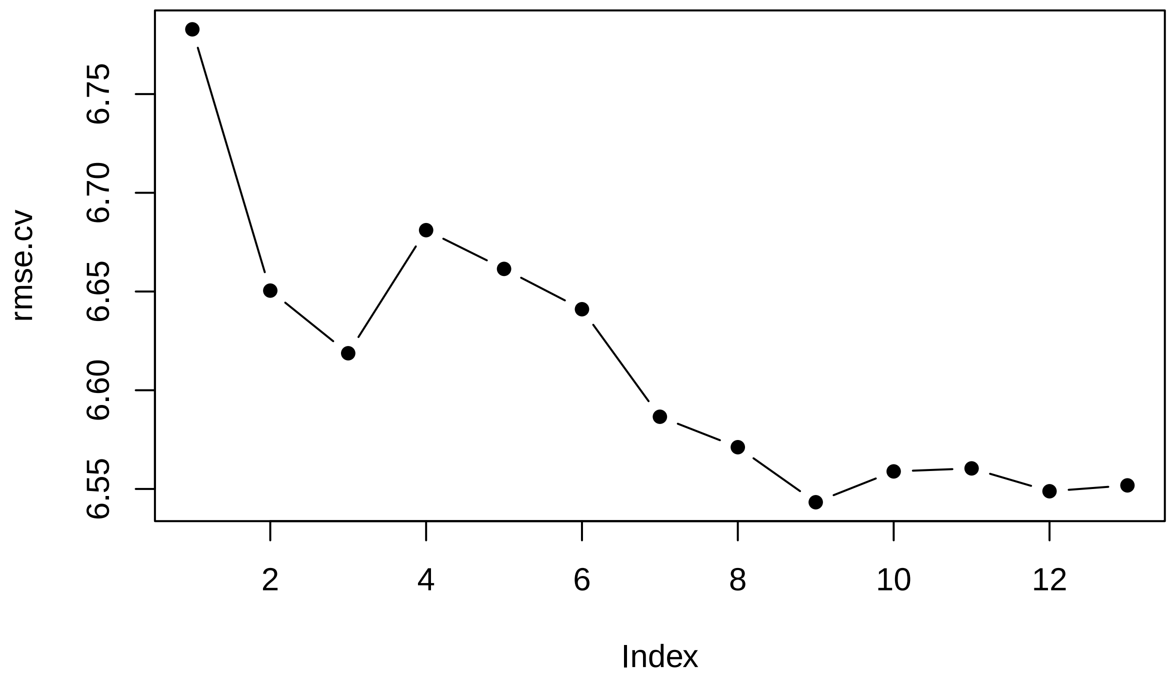
Best subset selection

```r
data(Boston)
set.seed(1)

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
}

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 <- sqrt(apply(cv.errors, 2, mean))
plot(rmse.cv, pch = 16, type = "b")
```



Here I try to choose among the models of different sizes using cross-validation. First, we create a vector that allocates each observation to one of k=10 folds, and I create a matrix in which I will store the results. Then I write a for loop that performs cross-validation. In the *i*th fold, the elements of folds that equal *i* are un the test set, and the remainder are in the training set. I make the predictions for each model size, compute the test errors on the appropriate subset, and store them in the appropriate slot in the matrix “cv.errors”. The code will automatically use the “predict.regsubsets()” function when I call “predict()” because the “best.fit” object has class “regsubsets”. After fitting 13 models, which equals to the number of variables minus 1, I will need to find the one model that minimizes the CV error on the test data.

```r
summary(best.fit)
```

```
## Subset selection object
## Call: regsubsets.formula(crim ~ ., data = Boston[folds != i, ], nvmax = p)
## 13 Variables  (and intercept)
##         Forced in Forced out
## zn          FALSE      FALSE
## indus       FALSE      FALSE
## chas        FALSE      FALSE
## nox         FALSE      FALSE
## rm          FALSE      FALSE
## age         FALSE      FALSE
## dis         FALSE      FALSE
## rad         FALSE      FALSE
## tax         FALSE      FALSE
## ptratio     FALSE      FALSE
## black       FALSE      FALSE
## lstat       FALSE      FALSE
## medv        FALSE      FALSE
## 1 subsets of each size up to 13
## Selection Algorithm: exhaustive
##           zn  indus chas nox rm  age dis rad tax ptratio black lstat medv
## 1  ( 1 )  " " " "   " "  " " " " " " " " "*" " " " "     " "   " "   " "
## 2  ( 1 )  " " " "   " "  " " " " " " " " "*" " " " "     " "   "*"   " "
## 3  ( 1 )  " " " "   " "  " " " " " " " " "*" " " " "     "*"   "*"   " "
## 4  ( 1 )  "*" " "   " "  " " " " " " " " "*" " " " "     "*"   "*"   " "
## 5  ( 1 )  "*" " "   " "  " " " " " " "*" "*" " " " "     " "   "*"   "*"
## 6  ( 1 )  "*" "*"   " "  " " " " " " "*" "*" " " " "     " "   "*"   "*"
## 7  ( 1 )  "*" "*"   " "  " " " " " " "*" "*" " " " "     "*"   "*"   "*"
## 8  ( 1 )  "*" " "   " "  "*" " " " " "*" "*" " " "*"     "*"   "*"   "*"
## 9  ( 1 )  "*" " "   " "  "*" "*" " " "*" "*" " " "*"     "*"   "*"   "*"
## 10  ( 1 ) "*" "*"   " "  "*" "*" " " "*" "*" " " "*"     "*"   "*"   "*"
## 11  ( 1 ) "*" "*"   "*"  "*" "*" " " "*" "*" " " "*"     "*"   "*"   "*"
## 12  ( 1 ) "*" "*"   "*"  "*" "*" " " "*" "*" "*" "*"     "*"   "*"   "*"
## 13  ( 1 ) "*" "*"   "*"  "*" "*" "*" "*" "*" "*" "*"     "*"   "*"   "*"
```

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

```
## [1] 9
```

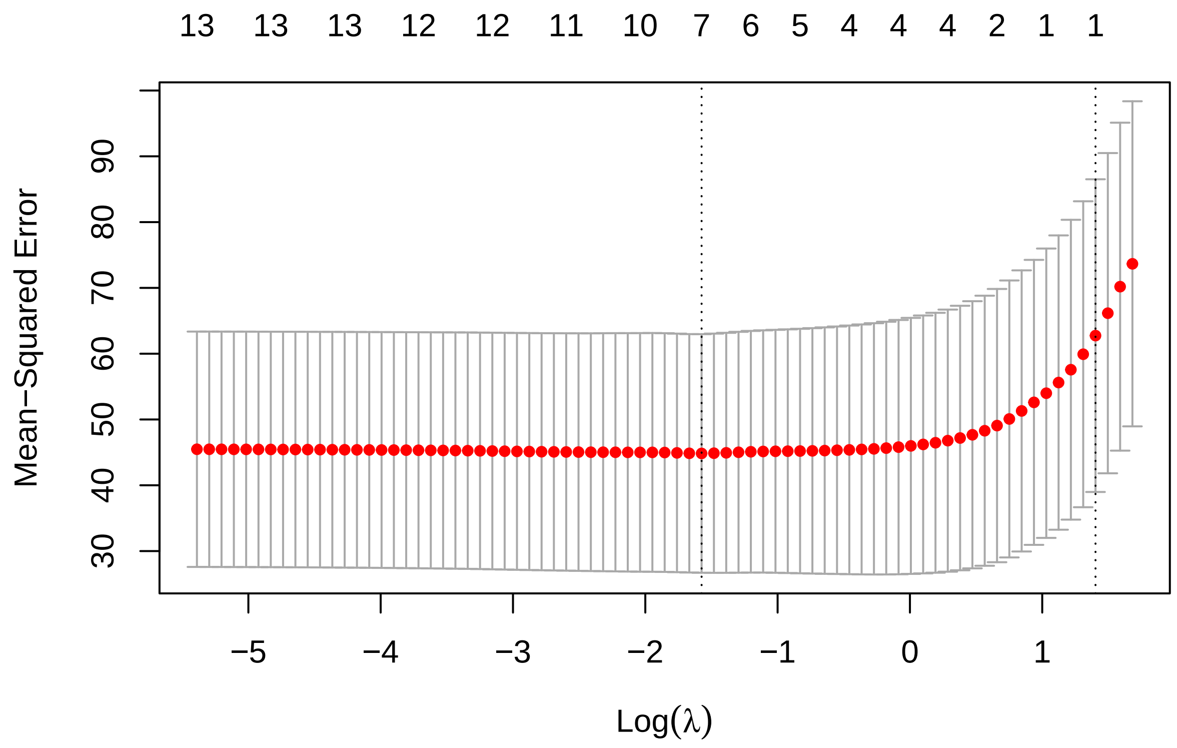
```r
boston.bsm.err <- (rmse.cv[which.min(rmse.cv)]) ^2
boston.bsm.err
```

```
## [1] 42.81453
```

I see that cross-validation selects a 9-variable model based on the test MSE. At 9-variables, the CV estimate for the test MSE is 43.32807. The variables that are included in this model are zn,indus,nox,dis,rad,ptratio,black,lstat, and medv.

The lasso

```
x <- model.matrix(crim ~ ., Boston)[, -1]
y <- Boston$crim
boston.lasso <- cv.glmnet(x, y, alpha = 1, type.measure = "mse")
plot(boston.lasso)
```



The graph above depicts the relationship between log λ and MSE. To help predict the training model on the test model, I will need to find the λ that reduces the error the most.

```
coef(boston.lasso)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                   s1
## (Intercept) 2.176491
## zn          .
## indus       .
## chas        .
## nox         .
## rm           .
## age          .
## dis          .
## rad          0.150484
## tax          .
## ptratio      .
## black        .
## lstat        .
## medv         .
```

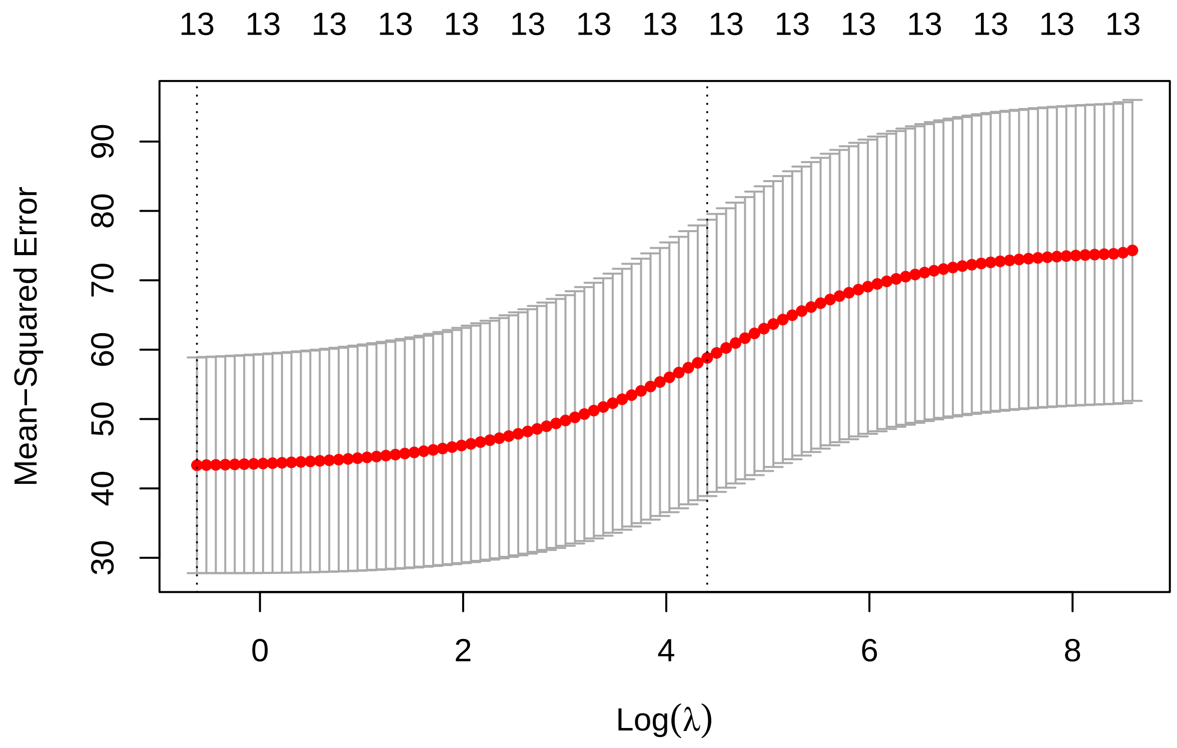
```r
boston.lasso.err <- (boston.lasso$cvm[boston.lasso$lambda == boston.lasso$lambda.1se])
boston.lasso.err
```

```
## [1] 62.74783
```

As we know that Lasso is a variable reduction method. From the results shown above, the Lasso model that reduces the MSE the model includes only one variable and has an MSE of 55.02399. The only variable included in this model is “rad”.

Ridge regression

```r
boston.ridge <- cv.glmnet(x, y, alpha = 0, type.measure = "mse")
plot(boston.ridge)
```



Ridge regression keeps all the variables but push their coefficient value close to zero if they do not have significance in the relationship with the response.

```
coef(boston.ridge)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  1.523899542
## zn          -0.002949852
## indus        0.029276741
## chas        -0.166526007
## nox          1.874769665
## rm          -0.142852604
## age          0.006207995
## dis         -0.094547258
## rad          0.045932737
## tax          0.002086668
## ptratio      0.071258052
## black       -0.002605281
## lstat        0.035745604
## medv        -0.023480540
```

```
boston.ridge.err <- boston.ridge$cvm[boston.ridge$lambda == boston.ridge$lambda.1se]
boston.ridge.err
```

```
## [1] 58.8156
```

The MSE for the ridge regression method is 61.37358 – much larger than those in other two methods. Ridge regression doesn't perform well. # PCR

```
boston.pcr <- pcr(crim ~ ., data = Boston, scale = TRUE, validation = "CV")
summary(boston.pcr)
```

```
## Data:    X dimension: 506 13
##  Y dimension: 506 1
## Fit method: svdpc
## Number of components considered: 13
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            8.61    7.175    7.180    6.724    6.731    6.727    6.727
## adjCV         8.61    7.174    7.179    6.721    6.725    6.724    6.724
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       6.722    6.614    6.618     6.607     6.598     6.553     6.488
## adjCV    6.718    6.609    6.613     6.602     6.592     6.546     6.481
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       47.70    60.36    69.67    76.45    82.99    88.00    91.14    93.45
## crim    30.69    30.87    39.27    39.61    39.61    39.86    40.14    42.47
##       9 comps  10 comps  11 comps  12 comps  13 comps
## X       95.40     97.04     98.46     99.52     100.0
## crim    42.55     42.78     43.04     44.13      45.4
```

Based on the CV error as well as the variances explained, the appropriate PCR model would only include 8 components. With 8 components, 93.45% of the variance is explained in the predictors by the model, and 42.47% of the variance is explained in the response variable by the model. Additionally, at 8 components, the MSE is at 6.614

###(b) Propose a model (or set of models) that seem to perform well on this data set, and justify your answer. Make sure that you are evaluating model performance using validation set error, cross-validation, or some other reasonable alternative, as opposed to using training error.

The model that has the lowest cross-validation error is the one chosen by the best subset selection method, which has a MSE of 43.32807.

###(c) Does your chosen model involve all of the features in the data set? Why or why not?

The model that was chosen by Best Subset Selection only includes 9 variables. The variables that are included in this model are zn, indus, nox, dis, rad, ptratio, black, lstat and medv. If the model were to include of the thrown-out features, more variation of the response would be present. For this particular problem, we are looking to have model prediction accuracy with low variance and low MSE.