HW8

2025-03-30

Setup 1: Simulation

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

Questions

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

```
n = 100
X = rnorm(n)
epsilon <- rnorm(n)

head(X)
```

```
## [1]  0.3621808 -1.0790503 -1.2485667 -1.0059602 -1.4164992 -0.5598122
```

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 $\beta_0, \beta_1, \beta_2$, and β_3 are constants of your choice.

```
beta0 = 1
beta1 = 2
beta2 = 3
beta3 = 4

# Generate response Y based on the model
Y = beta0 + beta1 * X + beta2 * X^2 + beta3 * X^3 + epsilon

head(Y)
```

```
## [1]  1.6276135 -3.2357757 -4.2592406 -0.3630507 -7.1341362 -0.3871872
```

c. Use the regsubsets() function to perform best subset selection in order to choose the best model containing the predictors $X, X^2, \cdots, X^{10}$. What is the best model obtained according to Cp, BIC, and adjusted R^2? 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.

```r
library(leaps)

data = data.frame(
  Y = Y,
  X1 = X,
  X2 = X^2,
  X3 = X^3,
  X4 = X^4,
  X5 = X^5,
  X6 = X^6,
  X7 = X^7,
  X8 = X^8,
  X9 = X^9,
  X10 = X^10
)

# Perform best subset selection
library(leaps)
regfit.full = regsubsets(Y ~ ., data = data, nvmax = 10)
reg.summary = summary(regfit.full)

# Plot Cp, BIC, and Adjusted R^2
par(mfrow = c(1, 3))

# Adjusted R^2
plot(reg.summary$adjr2, type = "b", xlab = "Number of Predictors", ylab = "Adjusted R^2")
which.max(reg.summary$adjr2)
```

```
## [1] 7
```

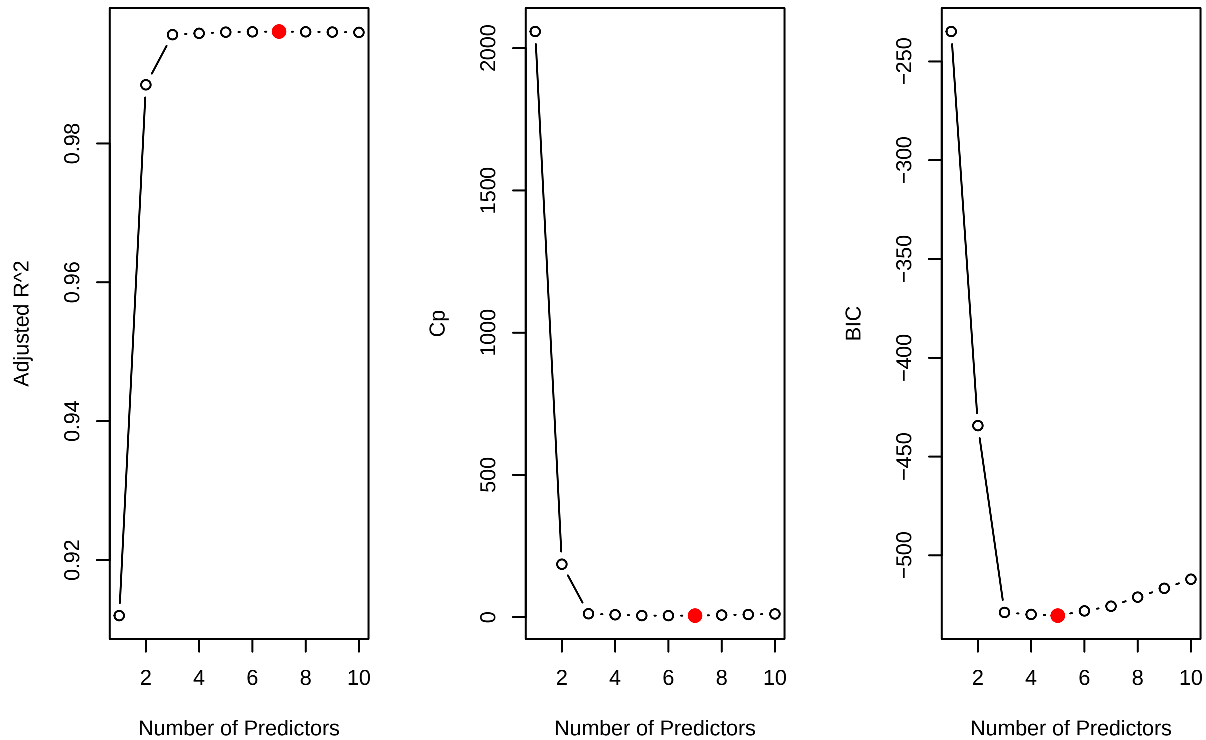
```r
points(which.max(reg.summary$adjr2), max(reg.summary$adjr2), col = "red", cex = 2, pch = 20)

# Cp
plot(reg.summary$cp, type = "b", xlab = "Number of Predictors", ylab = "Cp")
which.min(reg.summary$cp)
```

```
## [1] 7
```

```r
points(which.min(reg.summary$cp), min(reg.summary$cp), col = "red", cex = 2, pch = 20)

# BIC
plot(reg.summary$bic, type = "b", xlab = "Number of Predictors", ylab = "BIC")
which.min(reg.summary$bic)
```

```
## [1] 5
```

```r
points(which.min(reg.summary$bic), min(reg.summary$bic), col = "red", cex = 2, pch = 20)
```



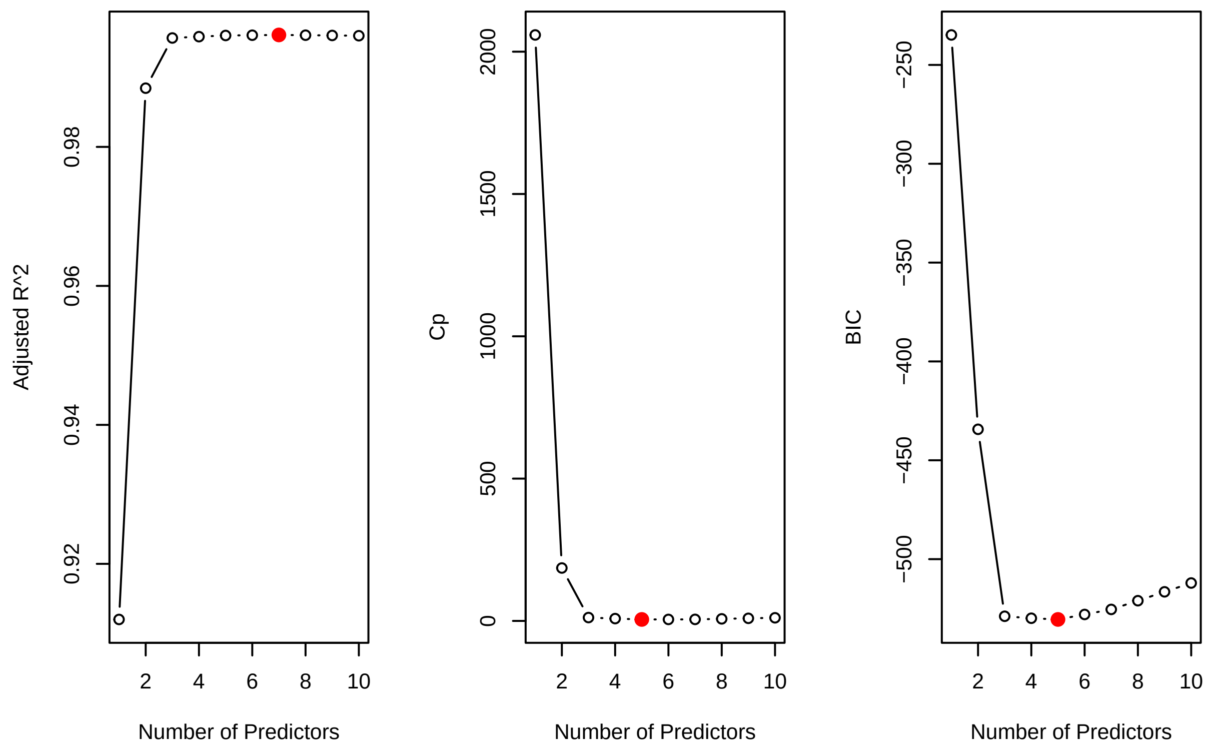

```
# Best model based on BIC
best.bic = which.min(reg.summary$bic)
coef(regfit.full, best.bic)
```

```
##   (Intercept)            X1            X2            X3            X7
##  0.6739149136  1.8690898336  3.2684052419  4.1697877792 -0.0077979304
##           X10
## -0.0003862224
```

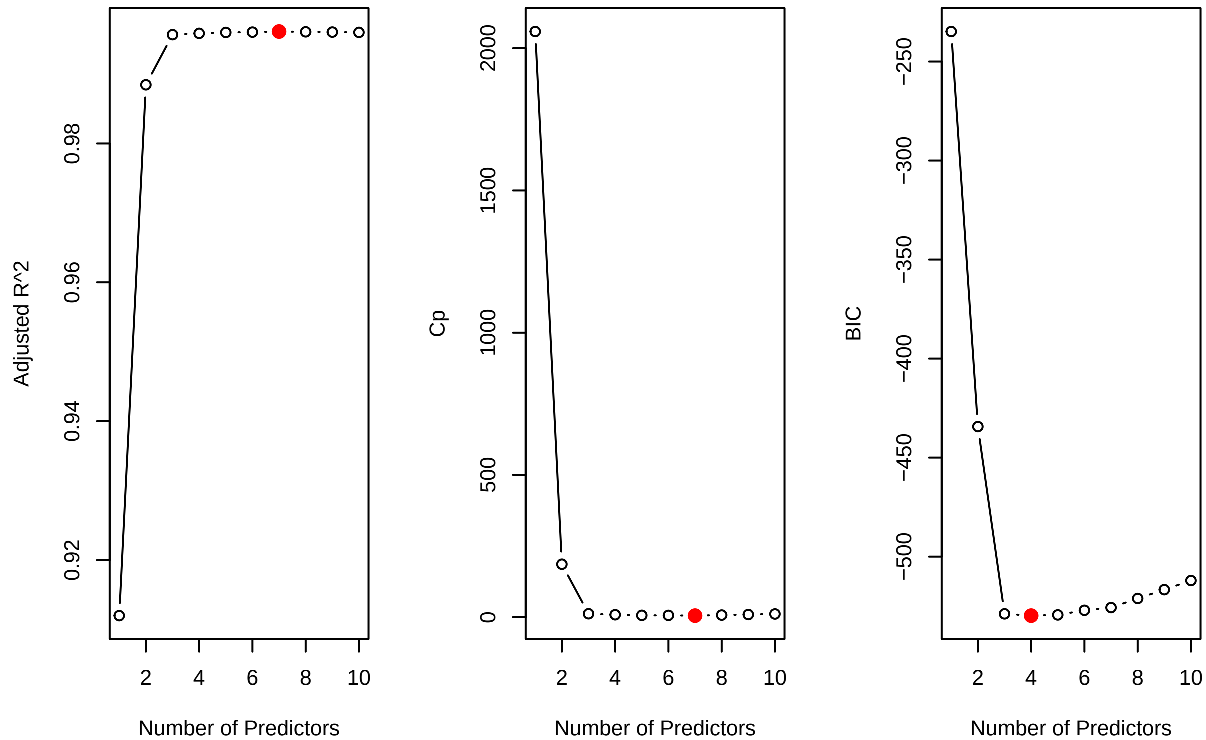
d. Repeat (c), using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in (c)?

```
# Forward stepwise selection
regfit.fwd = regsubsets(Y ~ ., data = data, nvmax = 10, method = "forward")
summary.fwd = summary(regfit.fwd)

# Plot metrics for forward selection
par(mfrow = c(1, 3))

plot(summary.fwd$adjr2, type = "b", xlab = "Number of Predictors", ylab = "Adjusted R^2")
points(which.max(summary.fwd$adjr2), max(summary.fwd$adjr2), col = "red", cex = 2, pch = 20)

plot(summary.fwd$cp, type = "b", xlab = "Number of Predictors", ylab = "Cp")
points(which.min(summary.fwd$cp), min(summary.fwd$cp), col = "red", cex = 2, pch = 20)
```

```r
plot(summary.fwd$bic, type = "b", xlab = "Number of Predictors", ylab = "BIC")
points(which.min(summary.fwd$bic), min(summary.fwd$bic), col = "red", cex = 2, pch = 20)
```




```r
# Backward stepwise selection
regfit.bwd = regsubsets(Y ~ ., data = data, nvmax = 10, method = "backward")
summary.bwd = summary(regfit.bwd)

# Plot metrics for backward selection
par(mfrow = c(1, 3))

plot(summary.bwd$adjr2, type = "b", xlab = "Number of Predictors", ylab = "Adjusted R^2")
points(which.max(summary.bwd$adjr2), max(summary.bwd$adjr2), col = "red", cex = 2, pch = 20)

plot(summary.bwd$cp, type = "b", xlab = "Number of Predictors", ylab = "Cp")
points(which.min(summary.bwd$cp), min(summary.bwd$cp), col = "red", cex = 2, pch = 20)

plot(summary.bwd$bic, type = "b", xlab = "Number of Predictors", ylab = "BIC")
points(which.min(summary.bwd$bic), min(summary.bwd$bic), col = "red", cex = 2, pch = 20)
```




```r
# Coefficients from best model under BIC for each method
coef(regfit.full, which.min(summary(regfit.full)$bic))   # Best subset
```

```
##   (Intercept)            X1            X2            X3            X7 
##  0.6739149136  1.8690898336  3.2684052419  4.1697877792 -0.0077979304 
##           X10 
## -0.0003862224
```

```r
coef(regfit.fwd, which.min(summary.fwd$bic))                # Forward
```

```
##   (Intercept)            X1            X2            X3            X7 
##  0.6739149136  1.8690898336  3.2684052419  4.1697877792 -0.0077979304 
##           X10 
## -0.0003862224
```

```r
coef(regfit.bwd, which.min(summary.bwd$bic))                # Backward
```

```
##  (Intercept)           X1           X2           X3           X8 
##  0.705929244  2.208129385  3.243083550  3.906524121 -0.001686833
```

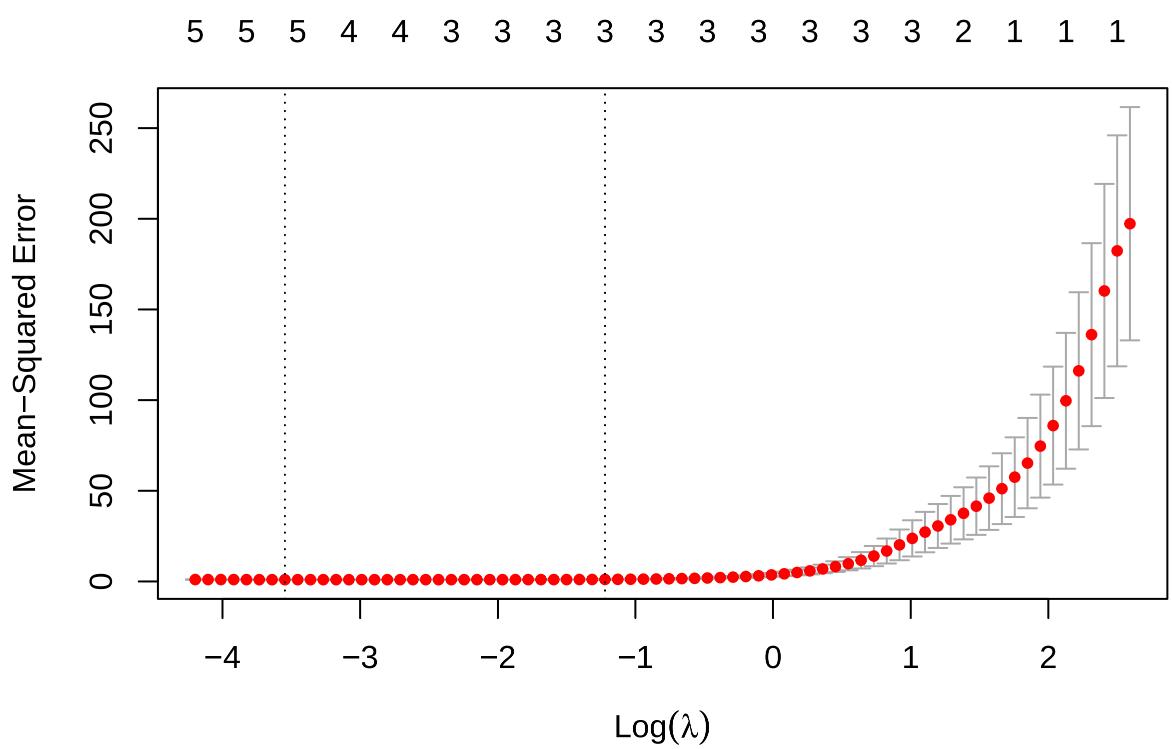
e. Now fit a lasso model to the simulated data, again using $X, X^2, \cdots, X^{10}$ as predictors. Use cross-validation to select the optimal value of λ. Create plots of the cross-validation error as a function of λ. Report the resulting coefficient estimates, and discuss the results obtained.

```r
library(glmnet)
```

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

```
## Loaded glmnet 4.1-8
```

```r
# Create design matrix (excluding intercept)
x = as.matrix(data[, -1])  # All columns except Y
y = data$Y

set.seed(1)  # for reproducibility
cv.lasso = cv.glmnet(x, y, alpha = 1)  # default uses MSE

# Plot cross-validation error
plot(cv.lasso)
```




```r
# Best lambda (minimizes cross-validation error)
best_lambda = cv.lasso$lambda.min
best_lambda
```

```
## [1] 0.02881359
```

```r
# Coefficients at best lambda
coef(cv.lasso, s = best_lambda)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept)  0.7625617313
## X1           2.1542228862
## X2           3.1486628843
## X3           3.9307068553
## X4           .
## X5           .
## X6           .
## X7           .
## X8           .
## X9          -0.0001549552
## X10         -0.0002053928
```

```r
lambda_1se = cv.lasso$lambda.1se
coef(cv.lasso, s = lambda_1se)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                   s1
## (Intercept) 1.029287
## X1          2.030008
## X2          2.801819
## X3          3.863810
## X4          .
## X5          .
## X6          .
## X7          .
## X8          .
## X9          .
## X10         .
```

f. Now generate a response vector Y according to the model

$$Y = \beta_0 + \beta_7 X^7 + \epsilon,$$

and perform best subset selection and the lasso. Discuss the results obtained.

```r
# Define new beta values
beta0 = 1
beta7 = 7

# Generate response Y with only X^7 as the true predictor
Y = beta0 + beta7 * X^7 + epsilon

# Update Y in the data frame
data$Y = Y
```

Setup 2: Ridge and lasso regression

In this exercise, we will predict per capita crime rate in the Boston data set from ISLR2 library.

Questions

a. Split the data set into a training set and a test set.

```r
# Load libraries
library(ISLR2)
library(glmnet)

head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio lstat medv
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3  4.98 24.0
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8  9.14 21.6
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8  4.03 34.7
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7  2.94 33.4
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7  5.33 36.2
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7  5.21 28.7
```

```r
train_indices = sample(1:nrow(Boston), nrow(Boston)/2)

train = Boston[train_indices, ]
test = Boston[-train_indices, ]
```

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

```r
lm.fit = lm(crim ~ ., data = train)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = crim ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.461  -2.743  -0.685   1.087  70.666
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  20.946202  12.802632   1.636  0.10313
## zn            0.053504   0.032863   1.628  0.10482
## indus         0.003554   0.143253   0.025  0.98023
## chas         -1.580534   1.912809  -0.826  0.40946
## nox         -16.106330   9.498916  -1.696  0.09126 .
## rm            1.339782   1.160223   1.155  0.24934
## age           0.006077   0.034051   0.178  0.85851
## dis          -1.246370   0.491867  -2.534  0.01192 *
## rad           0.680138   0.163817   4.152 4.59e-05 ***
## tax          -0.003157   0.009561  -0.330  0.74151
## ptratio      -0.547182   0.332303  -1.647  0.10094
## lstat         0.030062   0.148969   0.202  0.84024
## medv         -0.356412   0.108317  -3.290  0.00115 **
## ---
```

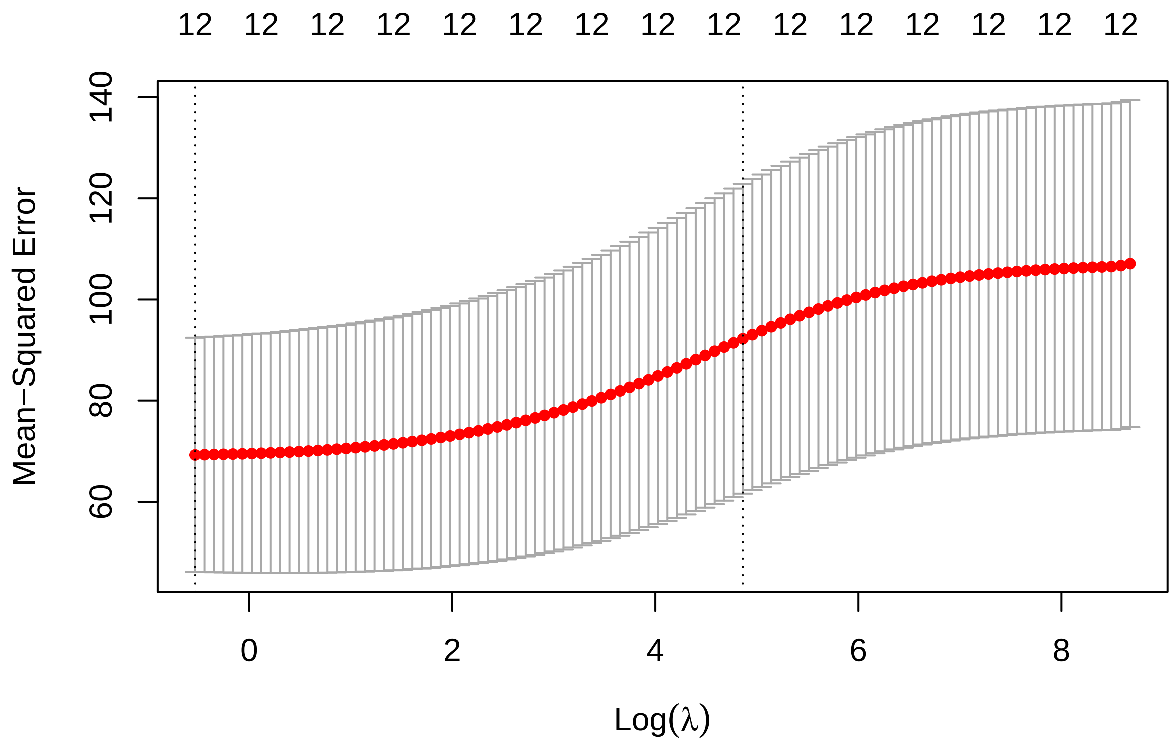
```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.245 on 240 degrees of freedom
## Multiple R-squared:  0.3943, Adjusted R-squared:  0.364
## F-statistic: 13.02 on 12 and 240 DF,  p-value: < 2.2e-16
```

```r
# Predict
lm.pred = predict(lm.fit, newdata = test)

# Test error
test_mse = mean((test$crim - lm.pred)^2)
test_mse
```

```
## [1] 19.26762
```

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

```r
# Design matrices
x_train = model.matrix(crim ~ ., data = train)[, -1]
x_test = model.matrix(crim ~ ., data = test)[, -1]
y_train = train$crim
y_test = test$crim

# Cross-validation to choose best lambda
cv.ridge = cv.glmnet(x_train, y_train, alpha = 0)  # alpha = 0 for ridge
plot(cv.ridge)
```




```
best_lambda_ridge = cv.ridge$lambda.min
best_lambda_ridge
```

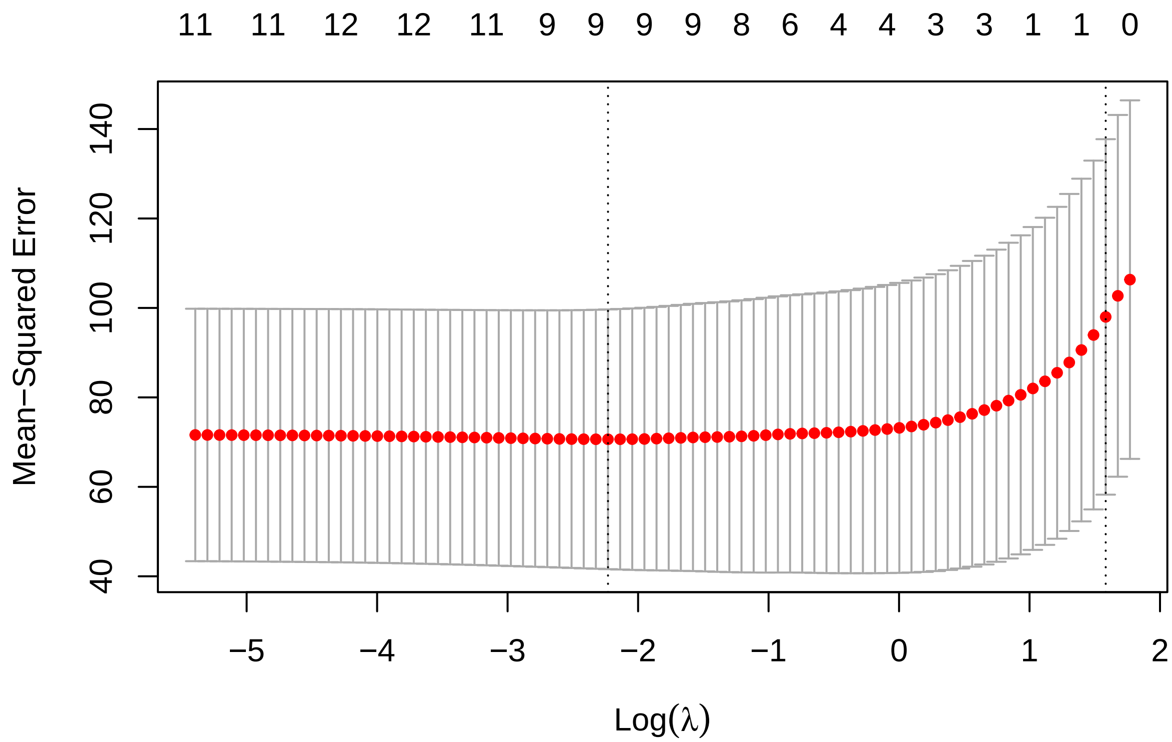
```
## [1] 0.5869899
```

```
# Predict crim using ridge
ridge.pred = predict(cv.ridge, s = best_lambda_ridge, newx = x_test)

# Test MSE
ridge_mse = mean((y_test - ridge.pred)^2)
ridge_mse
```

```
## [1] 18.01993
```

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.

```
# Fit lasso model with cross-validation
cv.lasso = cv.glmnet(x_train, y_train, alpha = 1)
plot(cv.lasso)
```




```r
best_lambda_lasso = cv.lasso$lambda.min
best_lambda_lasso
```

```
## [1] 0.1074625
```

```r
# Predict on test set and compute MSE
lasso.pred = predict(cv.lasso, s = best_lambda_lasso, newx = x_test)

lasso_mse = mean((y_test - lasso.pred)^2)
lasso_mse
```

```
## [1] 18.01794
```

```r
# Report number of non-zero coefficients
lasso.coef = predict(cv.lasso, s = best_lambda_lasso, type = "coefficients")

# Count non-zero coefficients (excluding intercept)
nonzero_count = sum(lasso.coef != 0) - 1
nonzero_count
```

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

e. Propose a model (or set of models) that seem to perform well on this data set, and justify your answer.

```
nonzero_count
```

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

Setup 3: PCR and PLS

In this exercise, we will predict the per capita crime rate in the Boston data set from the ISLR2 library.

Questions

a. Split the data set into a training set and a test set.

```
library(ISLR2)
n <- nrow(Boston)
train_index <- sample(1:n, size = 0.8 * n)

train_data <- Boston[train_index, ]
test_data  <- Boston[-train_index, ]
```

b. Fit a principal component regression (PCR) on the training set, with 10-fold cross-validation. Report the training and test errors obtained.

```
library(pls)
```

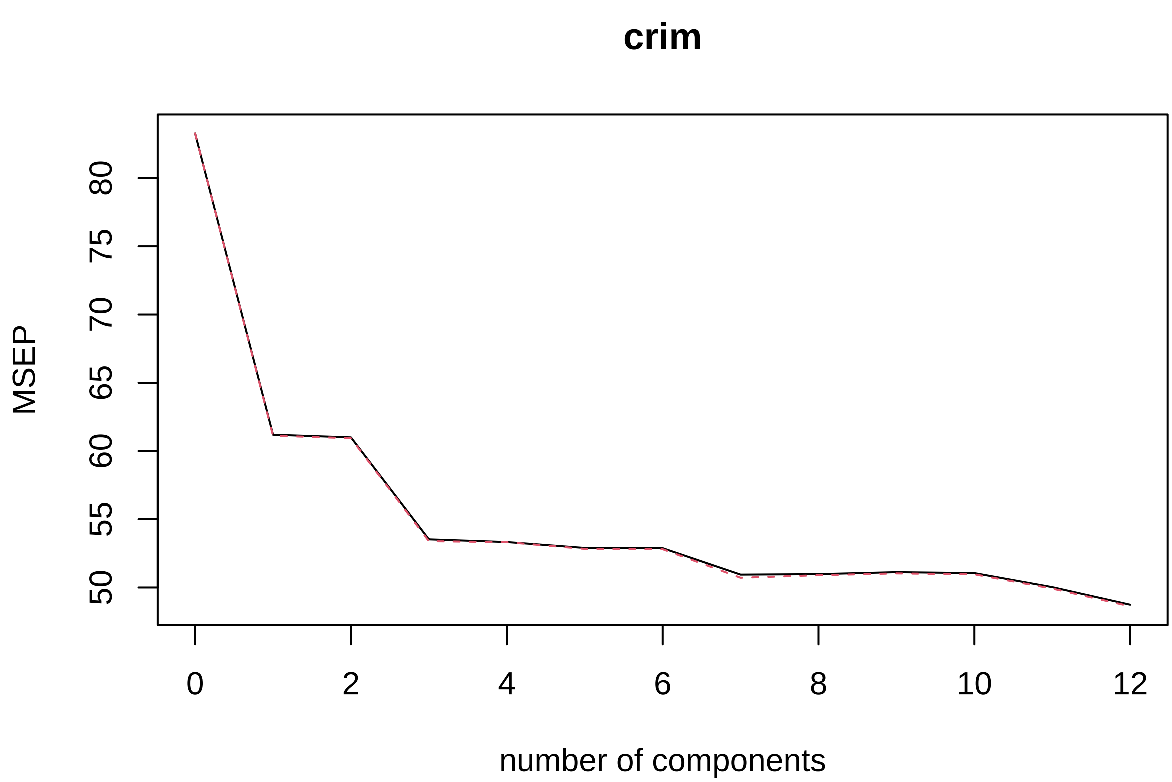
```
##
## Attaching package: 'pls'

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

```
# Fit PCR model on training set
pcr_model <- pcr(crim ~ .,
                 data = train_data,
                 scale = TRUE,
                 validation = "CV",
                 segments = 10)

summary(pcr_model)
```

```
## Data:    X dimension: 404 12
##  Y dimension: 404 1
## Fit method: svdpc
## Number of components considered: 12
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
```

```
## CV           9.125    7.823    7.810    7.316    7.303    7.273    7.272
## adjCV        9.125    7.818    7.806    7.307    7.302    7.268    7.267
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## CV       7.137    7.140    7.150     7.145     7.073     6.981
## adjCV    7.122    7.135    7.144     7.139     7.065     6.973
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       49.89    63.09    72.41    79.83    86.54    90.05    92.68    94.94
## crim    27.70    27.97    36.68    37.04    37.74    37.89    40.20    40.39
##       9 comps  10 comps  11 comps  12 comps
## X       96.79     98.23     99.44    100.00
## crim    40.50     40.65     42.04     43.71
```

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




```r
# Training error (MSE):
train_pred <- predict(pcr_model, ncomp = 5, newdata = train_data)

train_mse <- mean((train_data$crim - train_pred)^2)
cat("Training MSE:", train_mse, "\n")
```

```
## Training MSE: 51.59105
```

```r
# Test error (MSE):
test_pred <- predict(pcr_model, ncomp = 5, newdata = test_data)

test_mse <- mean((test_data$crim - test_pred)^2)
cat("Test MSE:", test_mse, "\n")
```

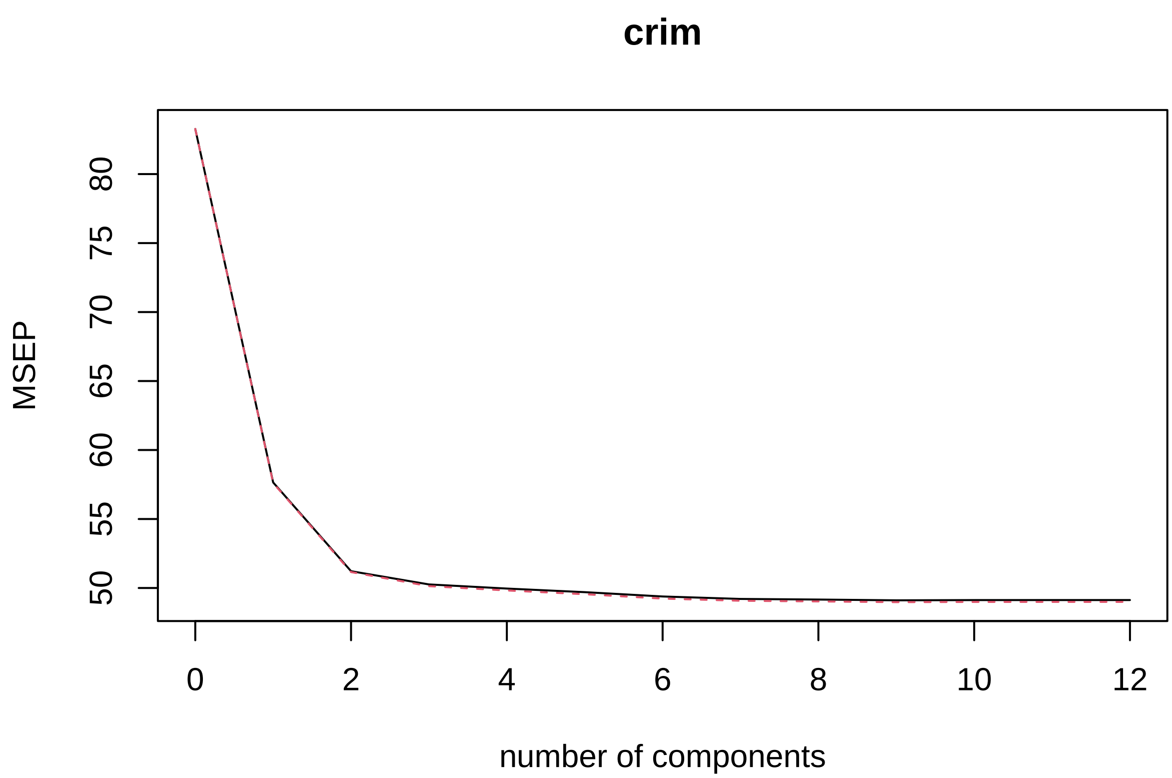
```
## Test MSE: 20.1244
```

c. Fit a partial least squares (PLS) regression on the training set, with 10-fold cross-validation. Report the training and test errors obtained.

```r
pls_model <- plsr(crim ~ .,
                  data = train_data,
                  scale = TRUE,
                  validation = "CV",
                  segments = 10)  # 10-fold CV

summary(pls_model)
```

```
## Data:    X dimension: 404 12
##  Y dimension: 404 1
## Fit method: kernelpls
## Number of components considered: 12
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           9.125    7.593    7.157    7.089    7.068    7.050    7.027
## adjCV        9.125    7.590    7.152    7.081    7.058    7.039    7.017
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## CV       7.015    7.012    7.008     7.009     7.009     7.009
## adjCV    7.005    7.002    6.998     6.999     6.999     6.999
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       49.38     59.0    65.04    74.18    80.00    83.02    87.47    91.52
## crim    31.55     39.9    41.98    42.77    43.19    43.58    43.65    43.70
##       9 comps  10 comps  11 comps  12 comps
## X       94.40     96.41     98.26    100.00
## crim    43.71     43.71     43.71     43.71
```

```r
# Plot cross-validated MSEP
validationplot(pls_model, val.type = "MSEP")
```




```r
# Training MSE
train_pred_pls <- predict(pls_model, ncomp = 5, newdata = train_data)

train_mse_pls <- mean((train_data$crim - train_pred_pls)^2)
cat("PLS Training MSE:", train_mse_pls, "\n")
```

```
## PLS Training MSE: 47.06915
```

```r
# Test MSE
test_pred_pls <- predict(pls_model, ncomp = 5, newdata = test_data)

test_mse_pls <- mean((test_data$crim - test_pred_pls)^2)
cat("PLS Test MSE:", test_mse_pls, "\n")
```

```
## PLS Test MSE: 17.88246
```

d. Propose a model (or set of models) from PCR and PLS that seem to perform well on this data set, and justify your answer.

PLS with 5 components is recommended because of lower training and test MSE.

Setup 4: Simulation

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.

Questions

a. Generate a data set with $p = 20$ features, $n = 100$ 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. For example, you may take $\beta_0 = 5, \beta_1 = 3, \beta_2 = -2, \beta_3 = -1.5, \beta_4 = 4, \beta_5 = 2.5$ and $\beta_6 = \beta_7 = \cdots = \beta_{20} = 0$.

```r
n <- 100  # number of observations
p <- 20   # number of predictors

# Predictor matrix X (n x p) with standard normal entries
X <- matrix(rnorm(n * p), nrow = n, ncol = p)

# beta coefficients
beta <- c(5, 3, -2, -1.5, 4, 2.5, rep(0, p - 6))  # length 20

epsilon <- rnorm(n, mean = 0, sd = 1)

# Response vector Y
Y <- X %*% beta + epsilon

data <- data.frame(Y = as.vector(Y), X)
head(data)
```

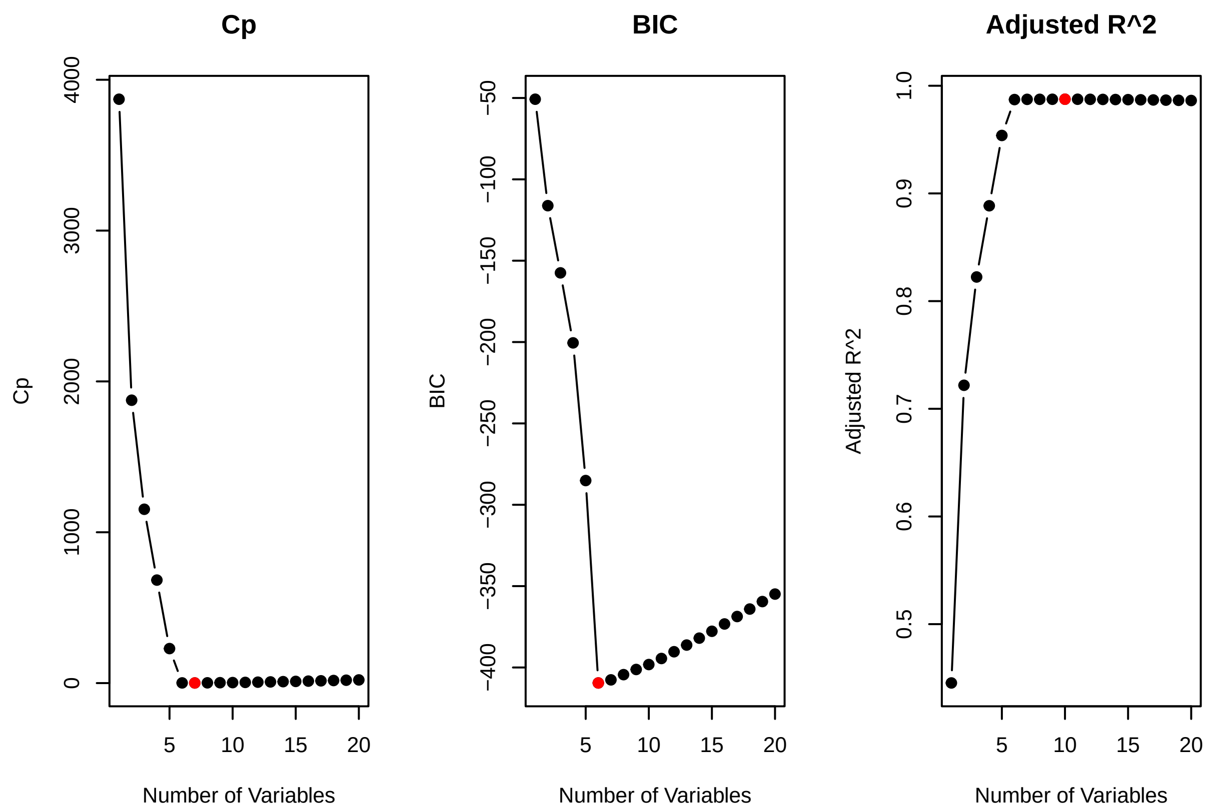
```
##             Y         X1         X2          X3          X4          X5
## 1 -10.165008 -1.0768583 -1.8382279 -0.09899057  0.82197560 -0.21430799
## 2   3.761119 -0.1153934 -1.2122184 -2.04188166  0.10081655  1.11854068
## 3 -10.893426 -1.3677358 -0.7722889  2.05374640  0.76286877  0.01637809
## 4  -3.887241  0.4775525 -0.3500790  0.61485858 -0.07909609 -1.14323258
## 5   5.570832  0.3060730  1.6812135  1.76955578  0.65796282  1.07018731
## 6 -11.612941 -0.5695257 -2.2906556  1.34456043 -0.52258263  0.32447596
##            X6          X7         X8         X9        X10         X11
## 1  0.8459935  2.07895132 -0.6822522 -0.8013787 -0.9523936  0.85713497
## 2 -0.6240430 -0.68903186  0.4354173 -0.5318423  0.8004883 -0.60531164
## 3  0.8847433  1.33553626  0.2507752  0.5350232  0.8029415  0.20302594
## 4  0.2261341 -0.59759416  1.7990032 -0.5669088 -0.7710643  0.63495908
## 5 -0.5610931 -0.06744711 -0.6331421 -2.3702270 -2.1592004  0.36434484
## 6 -0.5480882 -1.11857494  0.1216341  1.6586382 -0.1489608 -0.02924296
##           X12         X13        X14        X15        X16         X17
## 1  0.9279905  0.03835163  1.5198416  0.4672932 -0.3557857  0.29938660
## 2  0.3132932  0.14398998  0.2827584 -0.8664448 -0.7963201 -1.35222844
## 3  0.4060716 -0.10516484 -1.5514732  0.8850501 -0.8626919 -1.45750050
## 4 -1.1215840 -1.04478516 -0.4712982 -0.1557880 -0.7143957 -0.04298545
## 5 -0.5865006 -1.48427070  0.5848670 -0.9457282  0.9753716  0.57576470
## 6  0.3371968  0.16540946 -1.0192306 -0.2635098 -0.0790107  0.86061021
##            X18        X19          X20
## 1  1.36047317 -0.6428730 -0.296365796
## 2  0.97736555 -0.1925199 -0.001770592
## 3  1.18683760  0.9869745 -0.059001241
## 4  0.79404096 -1.5777959  0.916588949
## 5  0.06873992 -1.0077181 -0.090401539
## 6 -1.86319265  0.7588282  0.117370208
```

b. Generate a test set containing 1,000 observations.

```
n_test <- 1000
X_test <- matrix(rnorm(n_test * p), nrow = n_test, ncol = p)
epsilon_test <- rnorm(n_test, mean = 0, sd = 1)
Y_test <- X_test %*% beta + epsilon_test
test_data <- data.frame(Y = as.vector(Y_test), X_test)
```

c. Perform the best subset selection on the training set, and find the Cp statistic (BIC or adjusted R^2) associated with the best model of each size (number of regressor variables). Which model size do you suggest?

```
library(leaps)

# Best subset selection on training data
best_subset <- regsubsets(Y ~ ., data = data, nvmax = p)  # nvmax = 20

best_summary <- summary(best_subset)

par(mfrow = c(1, 3))

# Plot Cp
plot(best_summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "b", pch = 19)
points(which.min(best_summary$cp), min(best_summary$cp), col = "red", pch = 19)
title("Cp")

# Plot BIC
plot(best_summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "b", pch = 19)
points(which.min(best_summary$bic), min(best_summary$bic), col = "red", pch = 19)
title("BIC")

# Plot Adjusted R^2
plot(best_summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted R^2", type = "b", pch = 19)
points(which.max(best_summary$adjr2), max(best_summary$adjr2), col = "red", pch = 19)
title("Adjusted R^2")
```




```r
which.min(best_summary$cp)       # Best model size by Cp
```

```
## [1] 7
```

```r
which.min(best_summary$bic)      # Best model size by BIC
```

```
## [1] 6
```

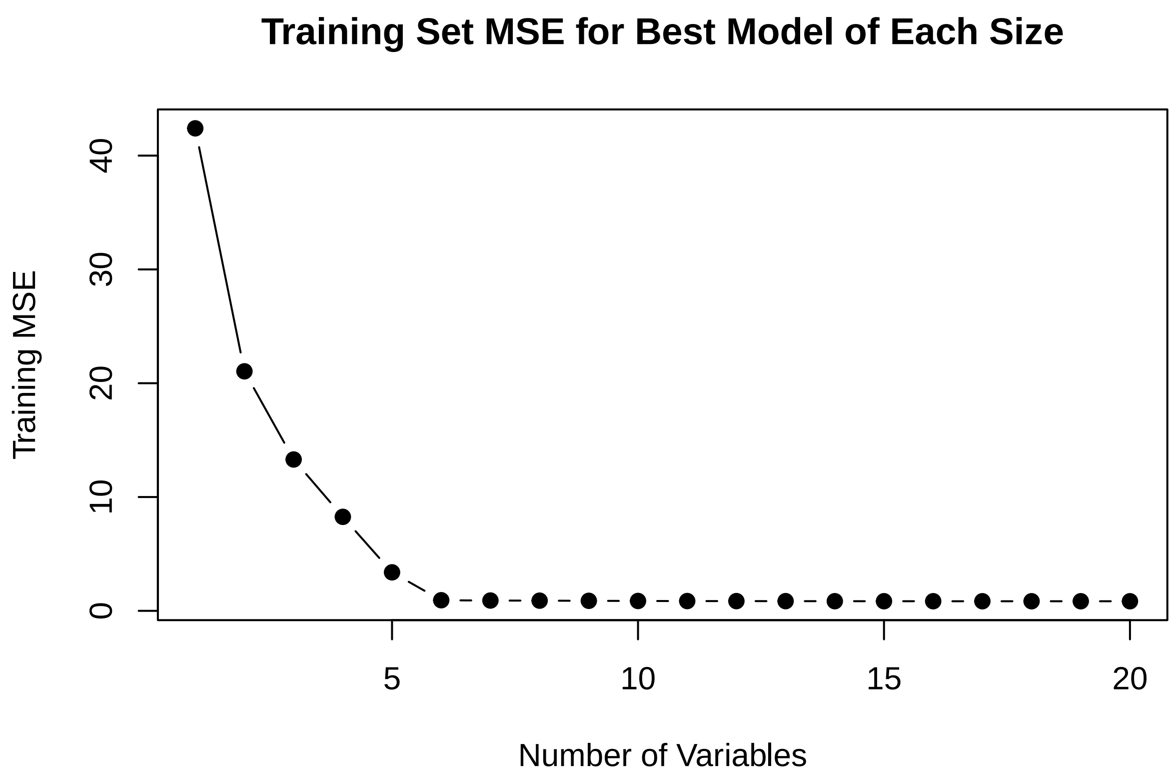
```r
which.max(best_summary$adjr2)    # Best model size by adjusted R^2
```

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

d. Plot the training set MSE associated with the best model of each size.

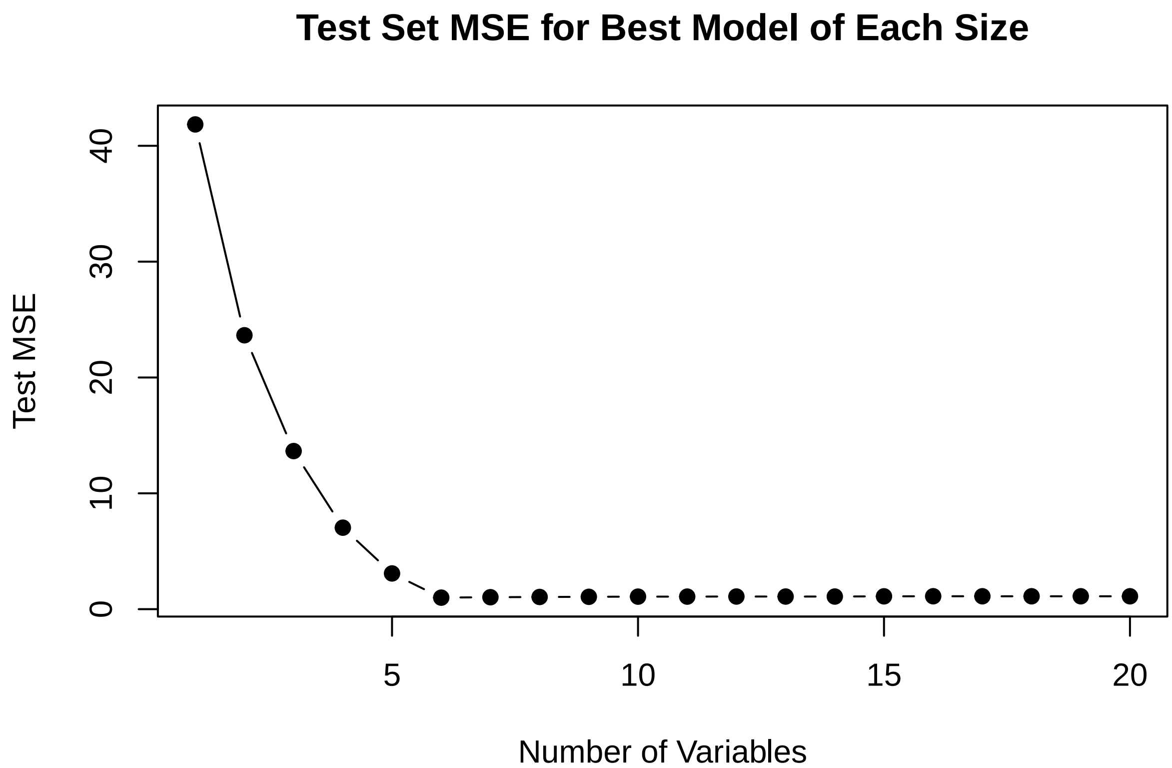
```r
# Number of observations in training set
n_train <- nrow(data)

# RSS from regsubsets summary
rss_values <- best_summary$rss

# Training MSE
train_mse <- rss_values / n_train
```

```r
# Plotting
plot(train_mse, type = "b", pch = 19, xlab = "Number of Variables", ylab = "Training MSE",
     main = "Training Set MSE for Best Model of Each Size")
```



e. Plot the test set MSE associated with the best model of each size.

```r
# Get model coefficients from regsubsets object
predict_regsubsets <- function(object, newdata, id) {
  model_coefs <- coef(object, id = id)
  vars <- names(model_coefs)
  pred_matrix <- model.matrix(as.formula(paste("~", paste(vars[-1], collapse = "+"))), newdata)
  return(pred_matrix %*% model_coefs)
}

# Initialize vector to store test MSEs
test_mse <- rep(NA, p)

for (i in 1:p) {
  pred <- predict_regsubsets(best_subset, newdata = test_data, id = i)
  test_mse[i] <- mean((test_data$Y - pred)^2)
}

# Plotting
plot(test_mse, type = "b", pch = 19, xlab = "Number of Variables", ylab = "Test MSE",
     main = "Test Set MSE for Best Model of Each Size")
```



f. For which model size does the test set MSE take on its minimum value? Is it similar to your suggestion based on the Cp statistic?

```
best_size_test_mse <- which.min(test_mse)
cat("Model size with minimum test set MSE:", best_size_test_mse, "\n")
```

```
## Model size with minimum test set MSE: 6
```

```
best_size_cp <- which.min(best_summary$cp)
cat("Model size with minimum Cp:", best_size_cp, "\n")
```

```
## Model size with minimum Cp: 7
```

The best model size based on test set performance (6 variables) is smaller than the model size suggested by Cp (11 variables). This demonstrates how training-based criteria (like Cp) can overestimate the ideal number of predictors, leading to models that may not generalize well. Therefore, test set MSE is a better guide for selecting model complexity in practice.

g. Now, perform steps (b) to (e) using LASSO, PCR, and PLS regression. Here, instead of using Cp (BIC or adjusted R^2), we will use a 10-fold cross-validation to select the optimum tuning parameter (λ for LASSO and M for PCR and PLS).

```r
library(glmnet)  # For LASSO
library(pls)     # For PCR and PLS
```

LASSO Regression (with Cross-Validation)

```r
X_mat <- as.matrix(data[, -1])
Y_vec <- data$Y
X_test_mat <- as.matrix(test_data[, -1])
Y_test_vec <- test_data$Y

# LASSO with 10-fold CV
cv_lasso <- cv.glmnet(X_mat, Y_vec, alpha = 1, nfolds = 10)

# Best lambda
best_lambda <- cv_lasso$lambda.min
cat("Best lambda for LASSO:", best_lambda, "\n")
```

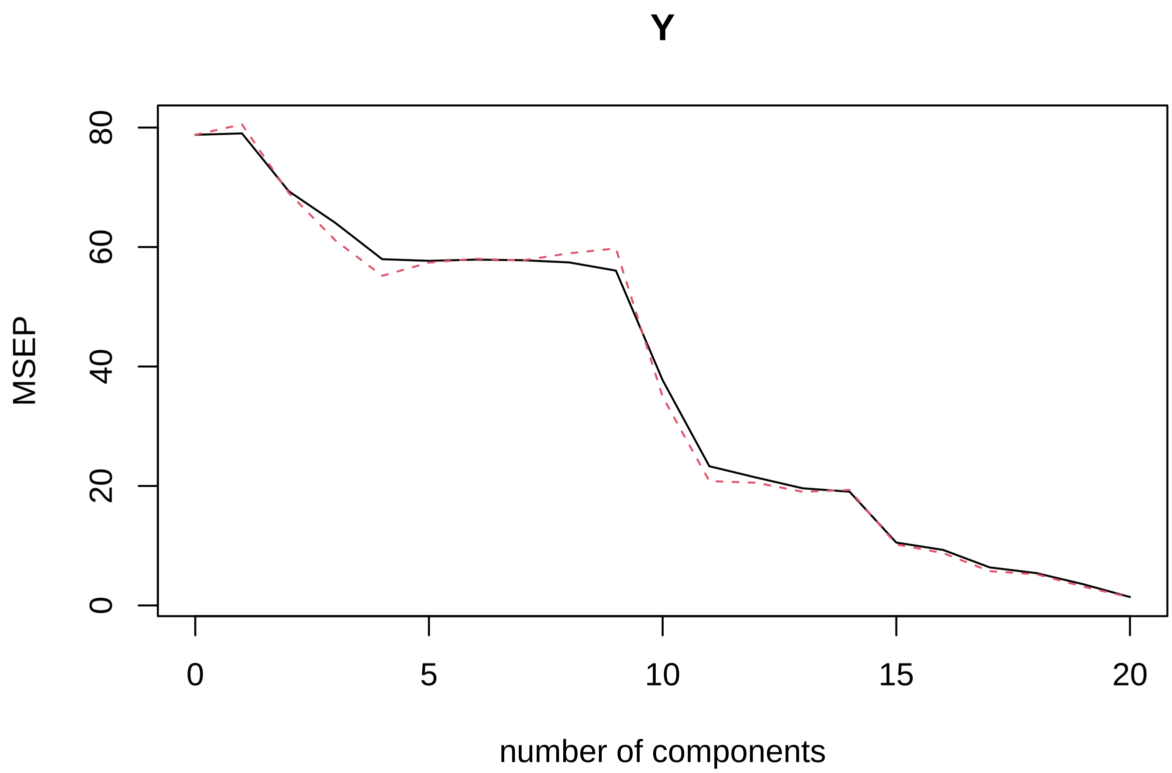
```
## Best lambda for LASSO: 0.06182319
```

```r
# Predict on test data
lasso_pred <- predict(cv_lasso, s = best_lambda, newx = X_test_mat)

# Test MSE
lasso_mse <- mean((Y_test_vec - lasso_pred)^2)
cat("Test MSE for LASSO:", lasso_mse, "\n")
```

```
## Test MSE for LASSO: 1.00106
```

Principal Components Regression (PCR)

```r
# PCR with 10-fold CV
pcr_model <- pcr(Y ~ ., data = data, scale = TRUE, validation = "CV", segments = 10)

validationplot(pcr_model, val.type = "MSEP")
```




```r
# Optimal number of components
opt_comp_pcr <- which.min(pcr_model$validation$PRESS)
cat("Optimal number of components for PCR:", opt_comp_pcr, "\n")
```

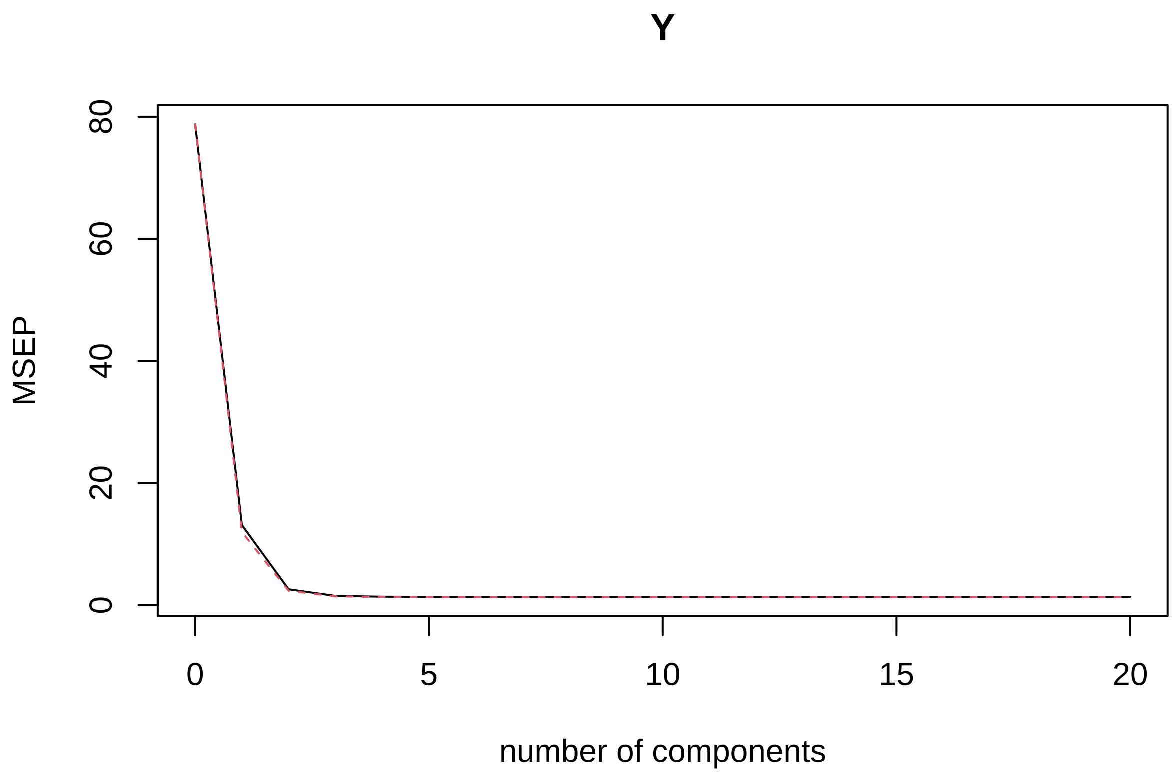
```
## Optimal number of components for PCR: 20
```

```r
# Predict on test data
pcr_pred <- predict(pcr_model, newdata = test_data, ncomp = opt_comp_pcr)
pcr_mse <- mean((Y_test_vec - pcr_pred)^2)
cat("Test MSE for PCR:", pcr_mse, "\n")
```

```
## Test MSE for PCR: 1.11128
```

Partial Least Squares Regression (PLS)

```r
# PLS with 10-fold CV
pls_model <- plsr(Y ~ ., data = data, scale = TRUE, validation = "CV", segments = 10)

# Plot MSE vs number of components
validationplot(pls_model, val.type = "MSEP")
```




```r
# Optimal number of components
opt_comp_pls <- which.min(pls_model$validation$PRESS)
cat("Optimal number of components for PLS:", opt_comp_pls, "\n")
```

```
## Optimal number of components for PLS: 7
```

```r
# Predict on test data
pls_pred <- predict(pls_model, newdata = test_data, ncomp = opt_comp_pls)
pls_mse <- mean((Y_test_vec - pls_pred)^2)
cat("Test MSE for PLS:", pls_mse, "\n")
```

```
## Test MSE for PLS: 1.111494
```