ISLR - Ch6 - Linear Model Selection and Regularization

Darshan Patel

2/7/2019

The following set of problems are from the applied exercises section in ISLR Chapter 4: Classification.

```
rm(list = ls())
library(MASS)
library(ISLR)
library(tidyverse)
```

```
## Warning: package 'tibble' was built under R version 3.4.4
```

```
## Warning: package 'tidyr' was built under R version 3.4.4
```

```
## Warning: package 'purrr' was built under R version 3.4.4
```

```
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
library(gridExtra)
library(scales)
```

```
## Warning: package 'scales' was built under R version 3.4.4
```

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

```
## Warning: package 'glmnet' was built under R version 3.4.4
```

```
## Warning: package 'Matrix' was built under R version 3.4.4
```

```
library(pls)
```

```
## Warning: package 'pls' was built under R version 3.4.4
```

Question 8: In this exercise, generate simulated data and 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(2019)
X = rnorm(100)
eps = rnorm(100)
```

(b) Generate a response vector Y of length $n = 100$ according to the model

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \varepsilon$$

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

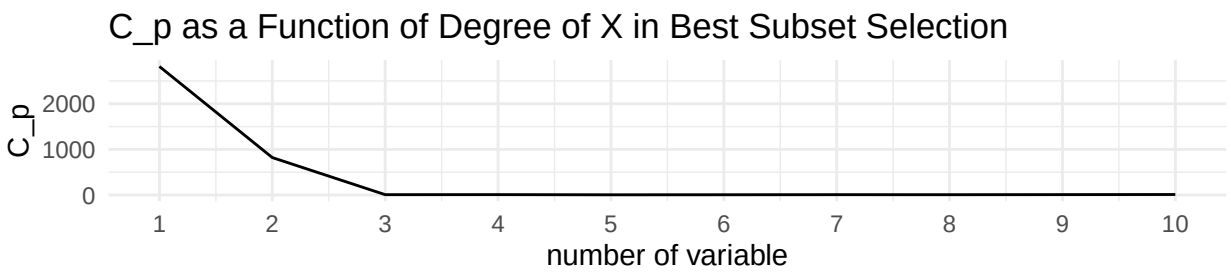
```
Y = 2 + (5*X) - (4*X^2) + (2*X^3) + eps
```

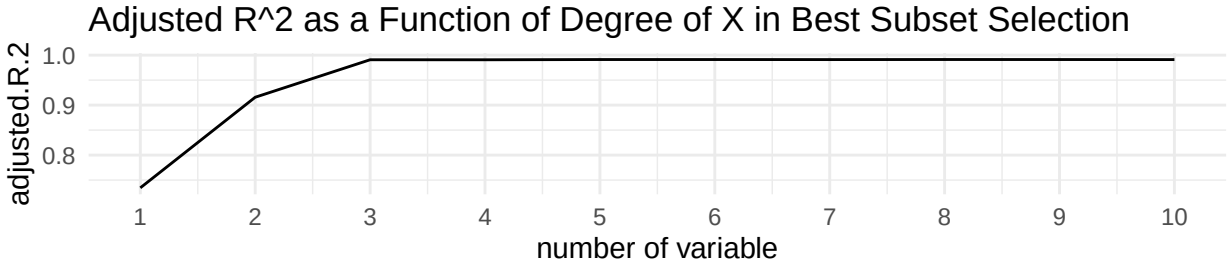
(c) Use the `regsubsets()` function to perform best subset selection in order to choose the best model containing the predictors $X, X^2, \ldots, X^{10}$. What is the best model obtained according to C_p, BIC and adjusted R^2? Show some plots to provide evidence for your answer and report the coefficients of the best model obtained.

```r
df = data.frame(y = Y, x = X)
bss1 = regsubsets(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
                    I(x^7) + I(x^8) + I(x^9) + I(x^10), df, nvmax = 10)
bss_model_stats = data.frame("num_vars" = seq(1,10),
                             "C_p" = summary(bss1)$cp,
                            "BIC" = summary(bss1)$bic,
                            "adjusted R^2" = summary(bss1)$adjr2)

g1 = ggplot(bss_model_stats, aes(x = num_vars, y = C_p)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("C_p as a Function of Degree of X in Best Subset Selection") +
  theme_minimal()
g2 = ggplot(bss_model_stats, aes(x = num_vars, y = BIC)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("BIC as a Function of Degree of X in Best Subset Selection") +
  theme_minimal()
g3 = ggplot(bss_model_stats, aes(x = num_vars, y = adjusted.R.2)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("Adjusted R^2 as a Function of Degree of X in Best Subset Selection") +
  theme_minimal()

grid.arrange(g1,g2,g3,ncol=1)
```





The best model has the following number of variables

```
min(which.min(summary(bss1)$cp),
    which.min(summary(bss1)$bic),
    which.max(summary(bss1)$adjr2))
```

```
## [1] 3
```

Therefore the coefficients of the model are

```
coef(bss1, 3)
```

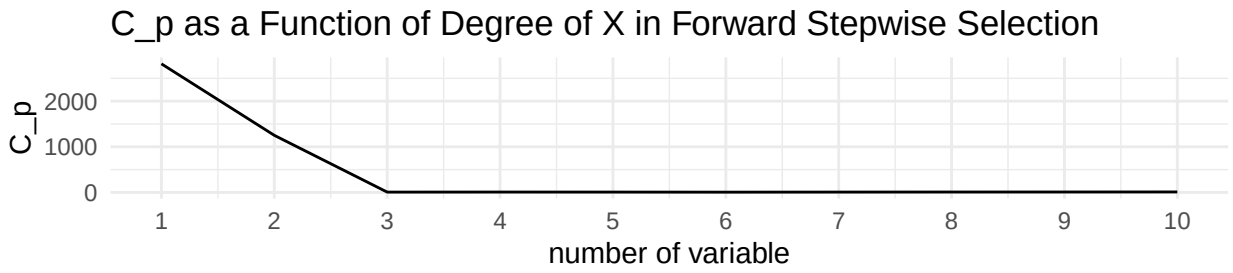
```
## (Intercept)           x      I(x^2)      I(x^3)
##    1.933207    5.160082   -4.110325    1.978292
```

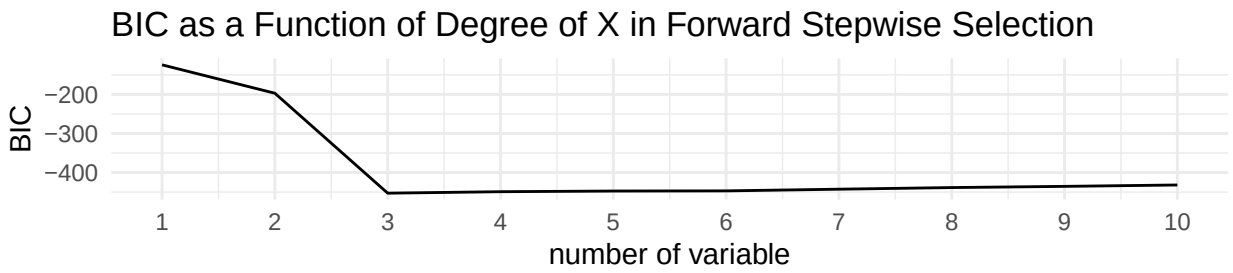
$$Y = 1.933207 + 5.160082X - 4.110325X^2 + 1.978292X^3$$

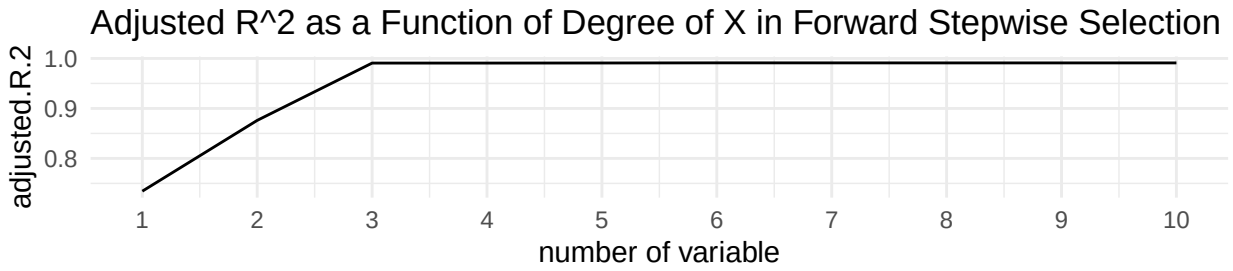
(d) Repeat (c), using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in (c)?

For forward stepwise selection,

```
fss1 = regsubsets(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
                    I(x^7) + I(x^8) + I(x^9) + I(x^10), df,
                  nvmax = 10, method = "forward")
fss_model_stats = data.frame("num_vars" = seq(1,10),
                             "C_p" = summary(fss1)$cp,
                            "BIC" = summary(fss1)$bic,
                            "adjusted R^2" = summary(fss1)$adjr2)
g4 = ggplot(fss_model_stats, aes(x = num_vars, y = C_p)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("C_p as a Function of Degree of X in Forward Stepwise Selection") +
  theme_minimal()
g5 = ggplot(fss_model_stats, aes(x = num_vars, y = BIC)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("BIC as a Function of Degree of X in Forward Stepwise Selection") +
  theme_minimal()
g6 = ggplot(fss_model_stats, aes(x = num_vars, y = adjusted.R.2)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("Adjusted R^2 as a Function of Degree of X in Forward Stepwise Selection") +
  theme_minimal()
grid.arrange(g4,g5,g6,ncol=1)
```







The best model has the following number of variables

```
min(which.min(summary(fss1)$cp),
    which.min(summary(fss1)$bic),
    which.max(summary(fss1)$adjr2))
```

```
## [1] 3
```

Therefore the coefficients of the model are

```
coef(fss1, 3)
```

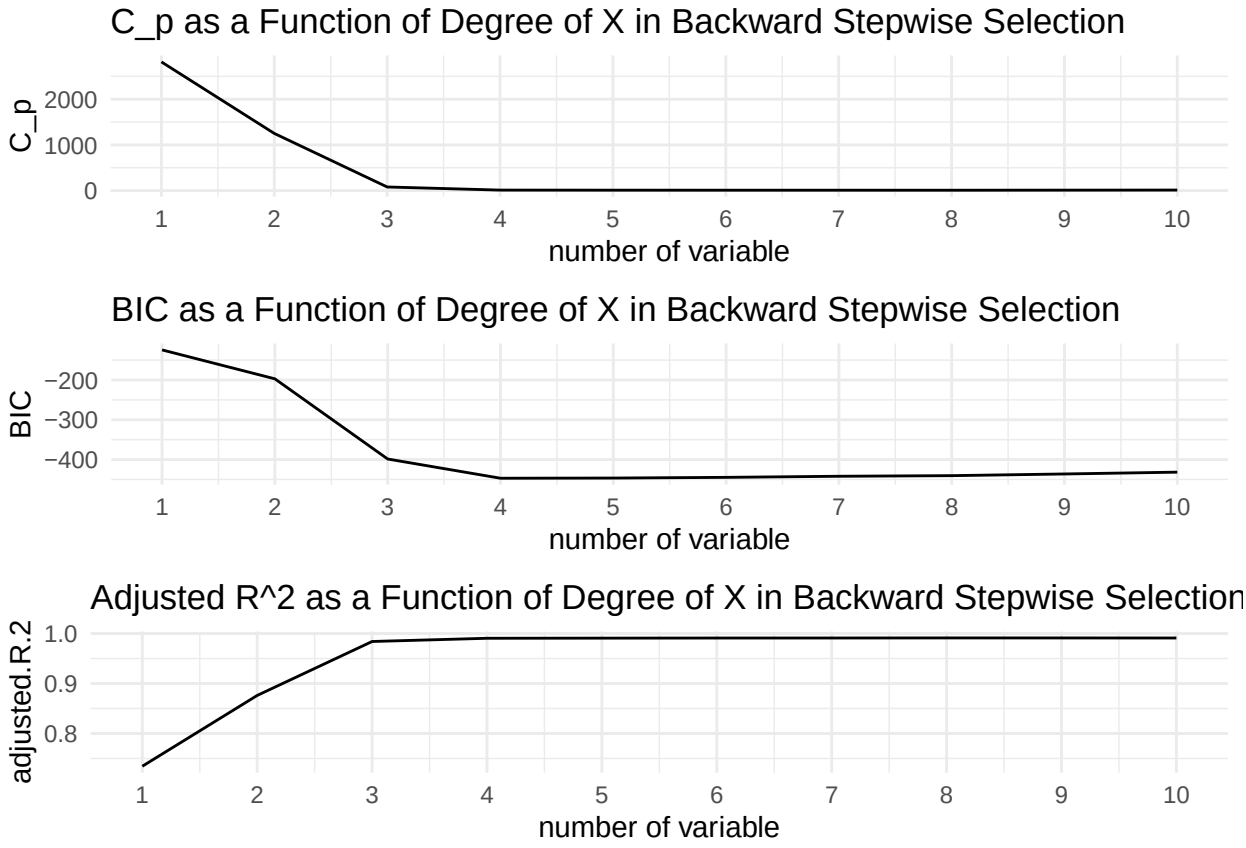
```
## (Intercept)           x      I(x^2)      I(x^3)
##    1.933207    5.160082   -4.110325    1.978292
```

$$Y = 1.933207 + 5.160082X - 4.110325X^2 + 1.978292X^3$$

For backward stepwise selection,

```
bss2 = regsubsets(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
                    I(x^7) + I(x^8) + I(x^9) + I(x^10), df,
                  nvmax = 10, method = "backward")
bss2_model_stats = data.frame("num_vars" = seq(1,10),
                              "C_p" = summary(bss2)$cp,
                              "BIC" = summary(bss2)$bic,
                              "adjusted R^2" = summary(bss2)$adjr2)
g7 = ggplot(bss2_model_stats, aes(x = num_vars, y = C_p)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
```

```
  ggtitle("C_p as a Function of Degree of X in Backward Stepwise Selection") +
  theme_minimal()
g8 = ggplot(bss2_model_stats, aes(x = num_vars, y = BIC)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("BIC as a Function of Degree of X in Backward Stepwise Selection") +
  theme_minimal()
g9 = ggplot(bss2_model_stats, aes(x = num_vars, y = adjusted.R.2)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("Adjusted R^2 as a Function of Degree of X in Backward Stepwise Selection") +
  theme_minimal()
grid.arrange(g7,g8,g9,ncol=1)
```



The best model has the following number of variables

```
min(which.min(summary(bss2)$cp),
    which.min(summary(bss2)$bic),
    which.max(summary(bss2)$adjr2))
```

```
## [1] 4
```

Therefore the coefficients of the model are

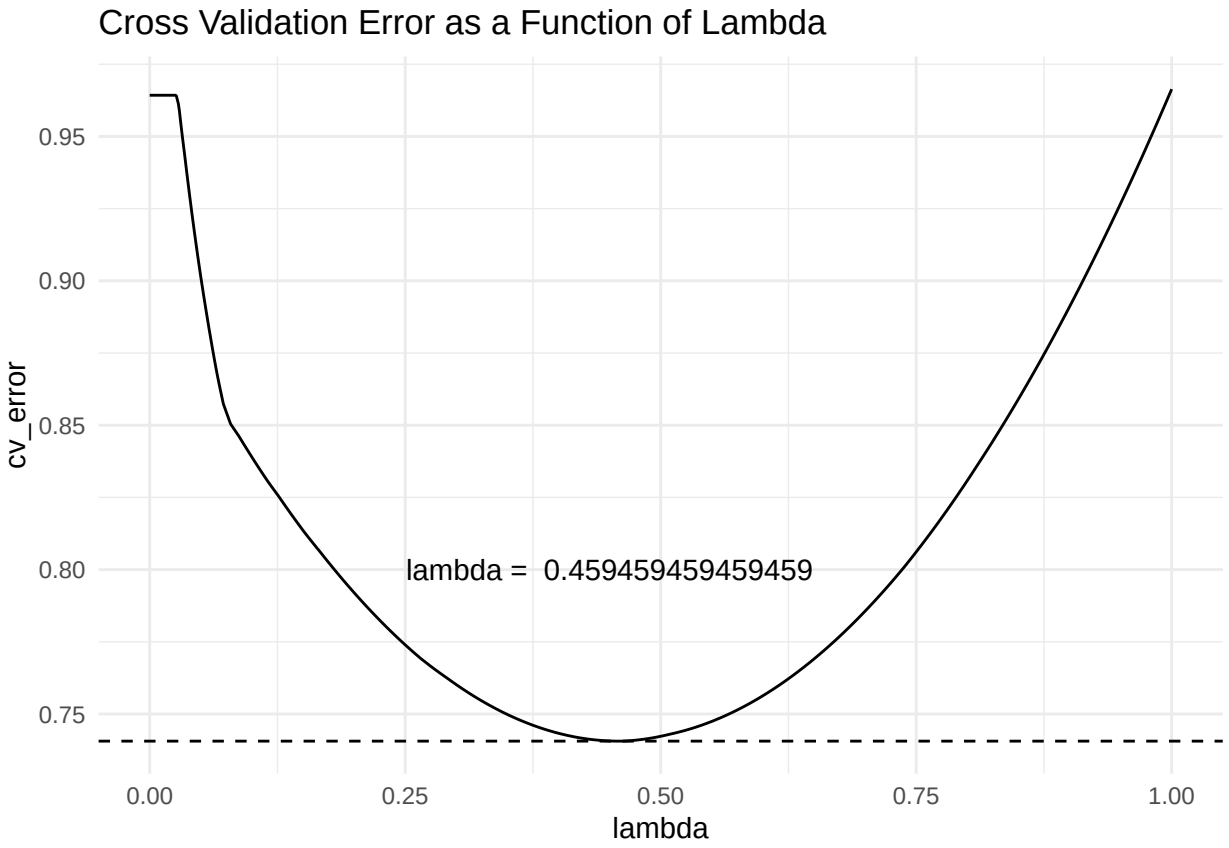
```
coef(bss2, 4)
```

```
## (Intercept)           x      I(x^2)      I(x^5)      I(x^7)
##  1.98255770  6.40039035 -4.11428941  0.67673696 -0.06107445
```

$$Y = 1.98255770 + 6.40039035X - 4.11428941X^2 + 0.67673696X^5 - 0.06107445X^7$$

(e) Now fit a lasso model to the stimulated data, again using $X, X^2, \ldots, X^{10}$ 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.

```
set.seed(2018)
indices = sample(1:nrow(df), size = 0.8*nrow(df))
Xs = data.frame(x = 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)
lasso_model = cv.glmnet(as.matrix(Xs[indices,]), Y[indices], alpha = 1)
Xrange = seq(0,1,length = 1000)
cv_error = c()
for(i in Xrange){
  cv_error = c(cv_error,
               mean((Y[-indices] - predict(lasso_model, s = i,
                                           newx = as.matrix(Xs[-indices,])))^2))
}
cv_error_df = data.frame("lambda" = Xrange, "cv_error" = cv_error)
ggplot(cv_error_df, aes(x = lambda, y = cv_error)) + geom_path() +
  ggtitle("Cross Validation Error as a Function of Lambda") +
  geom_hline(yintercept = cv_error[which.min(cv_error)], linetype = "dashed") +
  annotate("text", x = 0.45, y = 0.8,
           label = paste("lambda = ",
                         cv_error_df[which.min(cv_error), 1])) +
  theme_minimal()
```



The optimal λ value is 0.4594595. The resulting coefficient estimates are

```
predict(lasso_model, s = cv_error_df[which.min(cv_error),1],
        newx = as.matrix(Xs[-indices,]), type = "coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  1.642054
## x            4.971182
## x2          -3.770200
## x3           1.851105
## x4           .
## x5           .
## x6           .
## x7           .
## x8           .
## x9           .
## x10          .
```

$$Y = 1.642054 + 4.971182X - 3.770200X^2 + 1.851105X^3$$

(f) Now generate a response variable Y according to the model

$$Y = \beta_0 + \beta_7 X^7 + \varepsilon$$

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

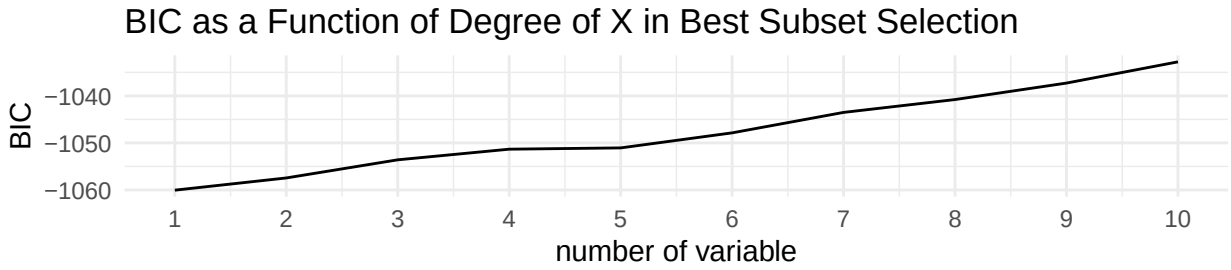
```
Y = 1 + (2 * X^7) + eps
```

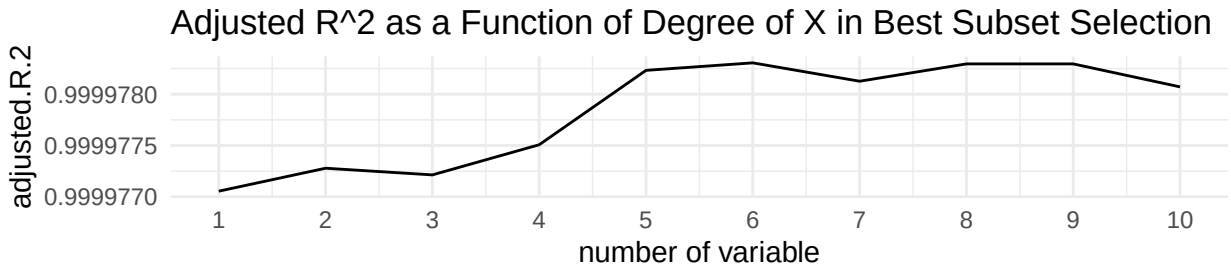
When performing best subset estimates

```
df = data.frame(y = Y, x = X)
bss3 = regsubsets(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
                    I(x^7) + I(x^8) + I(x^9) + I(x^10), df, nvmax = 10)
bss3_model_stats = data.frame("num_vars" = seq(1,10),
                              "C_p" = summary(bss3)$cp,
                              "BIC" = summary(bss3)$bic,
                              "adjusted R^2" = summary(bss3)$adjr2)

g10 = ggplot(bss3_model_stats, aes(x = num_vars, y = C_p)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("C_p as a Function of Degree of X in Best Subset Selection") +
  theme_minimal()
g11 = ggplot(bss3_model_stats, aes(x = num_vars, y = BIC)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("BIC as a Function of Degree of X in Best Subset Selection") +
  theme_minimal()
g12 = ggplot(bss3_model_stats, aes(x = num_vars, y = adjusted.R.2)) + geom_path() +
  scale_x_continuous(breaks = seq(1, 10, by = 1)) +
  labs(x = "number of variable") +
  ggtitle("Adjusted R^2 as a Function of Degree of X in Best Subset Selection") +
  theme_minimal()

grid.arrange(g10,g11,g12,ncol=1)
```





There does not appear to be a clear consensus on the best number of variables. Use the minimum value.

```
min(which.min(summary(bss3)$cp),
    which.min(summary(bss3)$bic),
    which.max(summary(bss3)$adjr2))
```

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

Therefore the coefficients of the model are

```
coef(bss3, 1)
```

```
## (Intercept)      I(x^7)
##   0.8322321   1.9997286
```

$$Y = 0.8322321 + 1.9997286X^7$$

Now, when using the lasso, the coefficients of the model are

```
lasso_model2 = cv.glmnet(as.matrix(Xs[indices,]), Y[indices], alpha = 1)
lasso = glmnet(as.matrix(Xs[-indices,]), Y[-indices], alpha = 1)
predict(lasso, s = lasso_model2$lambda.min, type = "coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) 2.40922964
## x           .
## x2          .
## x3          .
```

```
## x4          .
## x5          .
## x6          0.09487506
## x7          1.88777445
## x8          .
## x9          .
## x10         .
```

$$Y = 2.40922964 + 0.09487506X^6 + 1.88777445X^7$$

Question 9: In this exercise, predict the number of applications received using the other variables in the `College` dataset.

(a) Split the dataset into a training set and a test set.

```
set.seed(9)
df = College
indices = sample(1:nrow(df), size = 0.7 * nrow(df))
train = df[indices,]
test = df[-indices,]
```

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

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

```
## [1] 1858239
```

The test error is 1858239.

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

```
train_matrix = model.matrix(Apps~., df[indices,])
test_matrix = model.matrix(Apps~., df[-indices,])

model2 = glmnet(train_matrix, df[indices, "Apps"],
                alpha = 0,
                lambda = 10^seq(10, -2, length = 1000),
                thresh = 1e-12)
model2_cv = cv.glmnet(train_matrix, df[indices, "Apps"],
                      alpha = 0,
                      lambda = 10^seq(10, -2, length = 1000),
                      thresh = 1e-12)
mean((predict(model2, s = model2_cv$lambda.min,
              newx = test_matrix) - df[-indices, "Apps"])^2)
```

```
## [1] 2033792
```

The test error obtained using ridge regression is 2008793, higher than by linear modeling.

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

```
model3 = glmnet(train_matrix, df[indices, "Apps"], alpha = 1,
                lambda = 10^seq(10, -2, length = 1000), thresh = 1e-12)
```

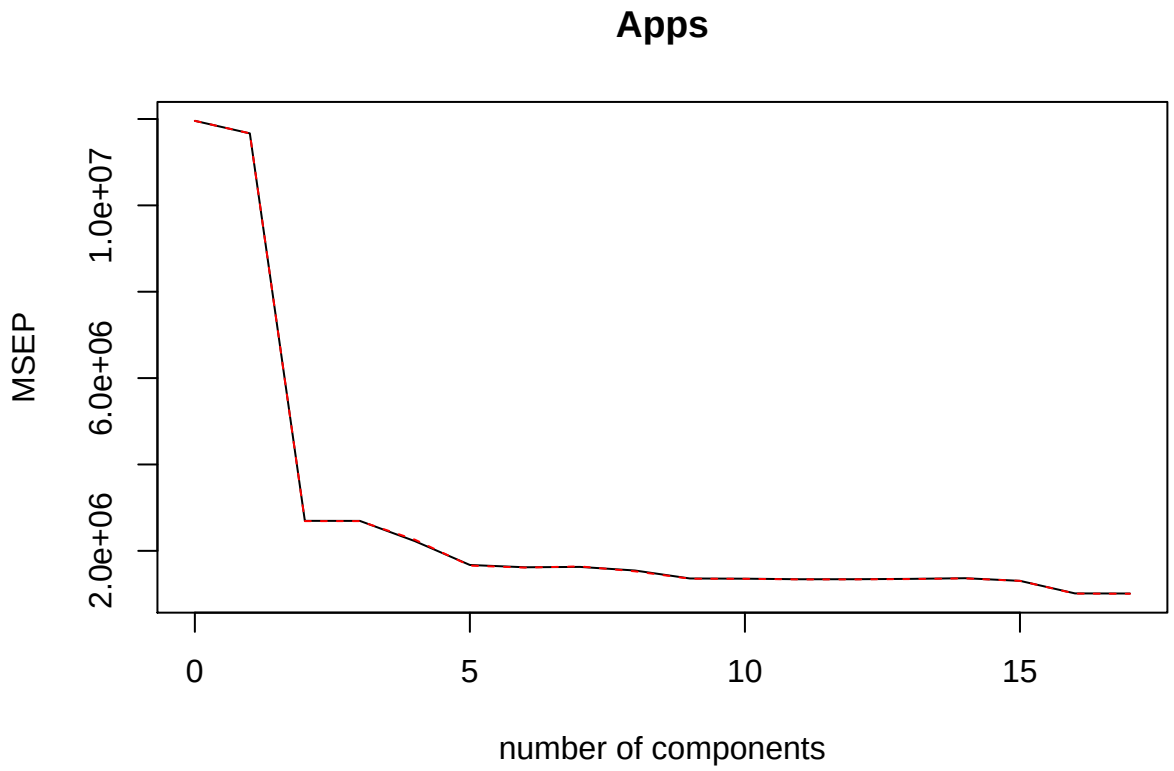
```
model3_cv = cv.glmnet(train_matrix, df[indices, "Apps"], alpha = 1,
                      lambda = 10^seq(10, -2, length = 1000), thresh = 1e-12)
mean((predict(model3, s = model3_cv$lambda.min,
              newx = test_matrix) - df[-indices, "Apps"])^2)
```

```
## [1] 1882129
```

The test error obtained using the lasso model is 1866080, lower than ridge regression but higher than regular linear regression.

(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.

```
model4 = pcr(Apps~., data = df[indices,], scale = TRUE, validation = "CV")
validationplot(model4, val.type = "MSEP")
```




```
mean((df[-indices, "Apps"] - predict(model4, newdata = df[-indices,], ncomp = 17))^2)
```

```
## [1] 1858239
```

The test error obtained using the PCR model is 1858239, same as the one for regular linear regression.

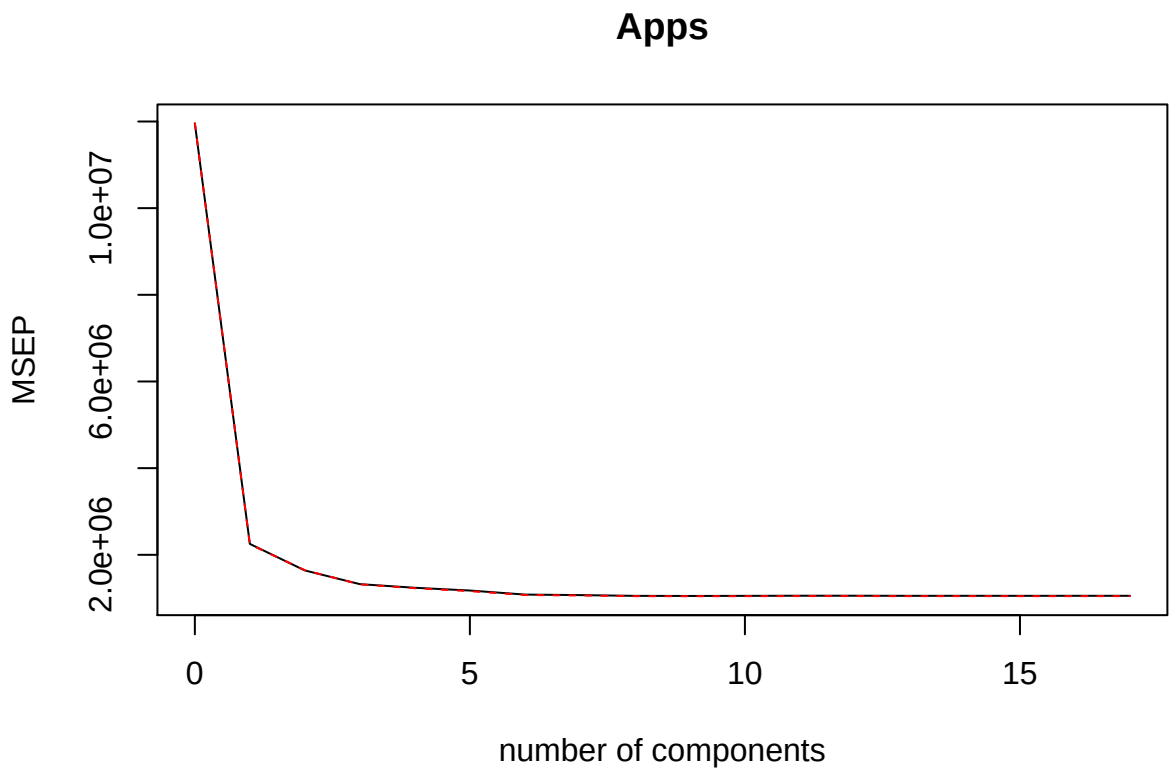
If `ncomp` is set to one less component, then the test set error is

```
mean((df[-indices, "Apps"] - predict(model4, newdata = df[-indices,], ncomp = 16))^2)
```

```
## [1] 2071459
```

(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.

```
model5 = plsr(Apps~., data = df[indices,], scale = TRUE, validation = "CV")
validationplot(model5, val.type = "MSEP")
```




```
mean((df[-indices, "Apps"] - predict(model5, newdata = df[-indices,], ncomp = 8))^2)
```

```
## [1] 1950821
```

Using as few as 8 components, the test set error is 1950821.

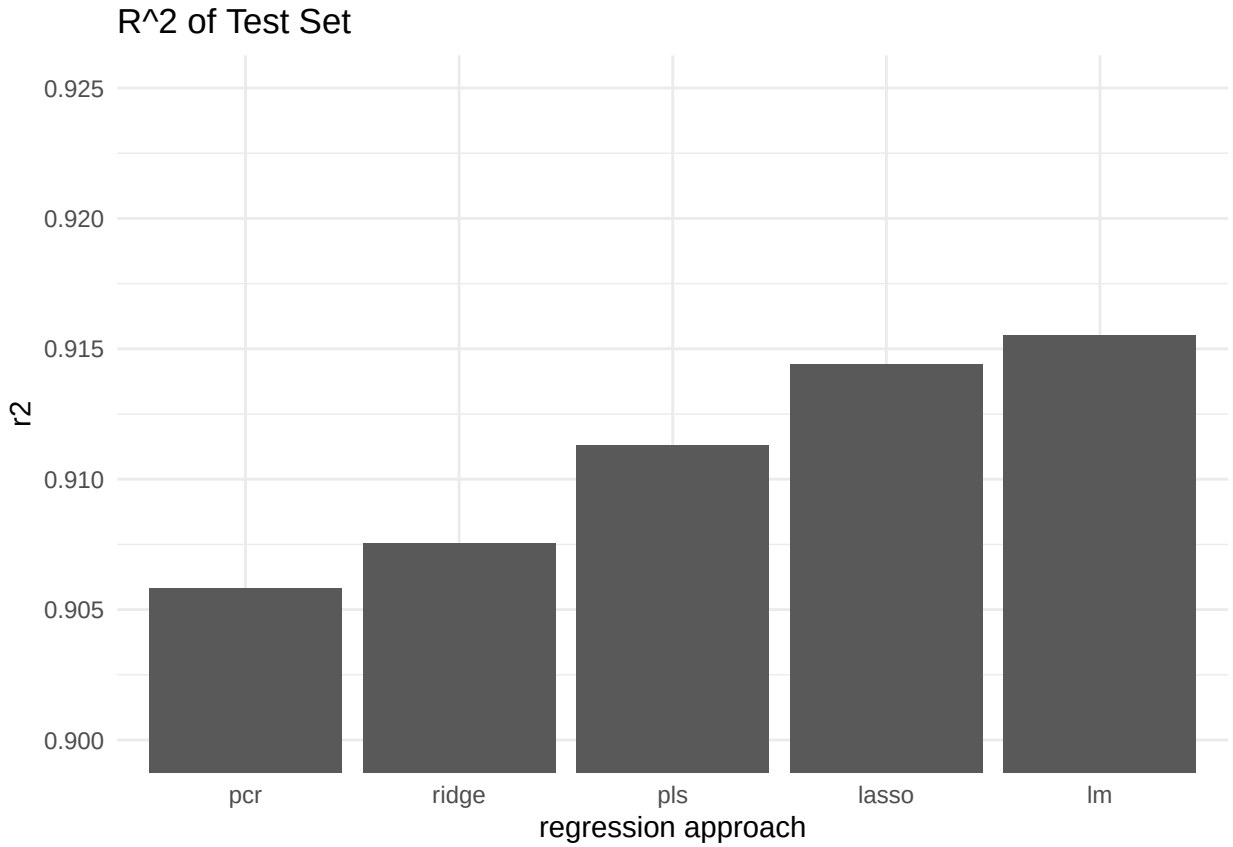
(g) Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

To determine the accuracy in the predictions, compute the R^2 value for each of the models on the test set.

```
test_mu_apps = mean(test$Apps)
lm_r2 = 1 - mean((pred - test$Apps)^2) / mean((test_mu_apps - test$Apps)^2)
ridge_r2 = 1 - mean((predict(model2, s = model2_cv$lambda.min,
                             newx = test_matrix) - test$Apps)^2) /
  mean((test_mu_apps - test$Apps)^2)
lasso_r2 = 1 - mean((predict(model3, s = model3_cv$lambda.min,
                             newx = test_matrix) - test$Apps)^2) /
  mean((test_mu_apps - test$Apps)^2)
pcr_r2 = 1 - mean((predict(model4, newdata = df[-indices,], ncomp = 16) - test$Apps)^2) /
  mean((test_mu_apps - test$Apps)^2)
pls_r2 = 1 - mean((predict(model5, newdata = df[-indices,], ncomp = 8) - test$Apps)^2) /
  mean((test_mu_apps - test$Apps)^2)

df_r2 = data.frame("approach" = c("lm", "ridge", "lasso", "pcr", "pls"),
                   "r2" = c(lm_r2, ridge_r2, lasso_r2, pcr_r2, pls_r2))

ggplot(df_r2, aes(x = reorder(approach, r2), y = r2)) + geom_bar(stat = "identity") +
  coord_cartesian(ylim = c(0.9, 0.925)) + ggtitle("R^2 of Test Set") +
  labs(x = "regression approach") +
  theme_minimal()
```



According to this plot, the model predicted `Apps` the best was the regular linear regression model, followed closely by the lasso regression model. The principal components regression performed the worst.

Question 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. Now explore this in a simulated dataset.

(a) Generate a data set with $p = 20$ features, $n = 1,000$ observations and an associated quantitative response vector generated according to the model

$$Y = X\beta + \varepsilon$$

where β has some elements that are exactly equal to zero.

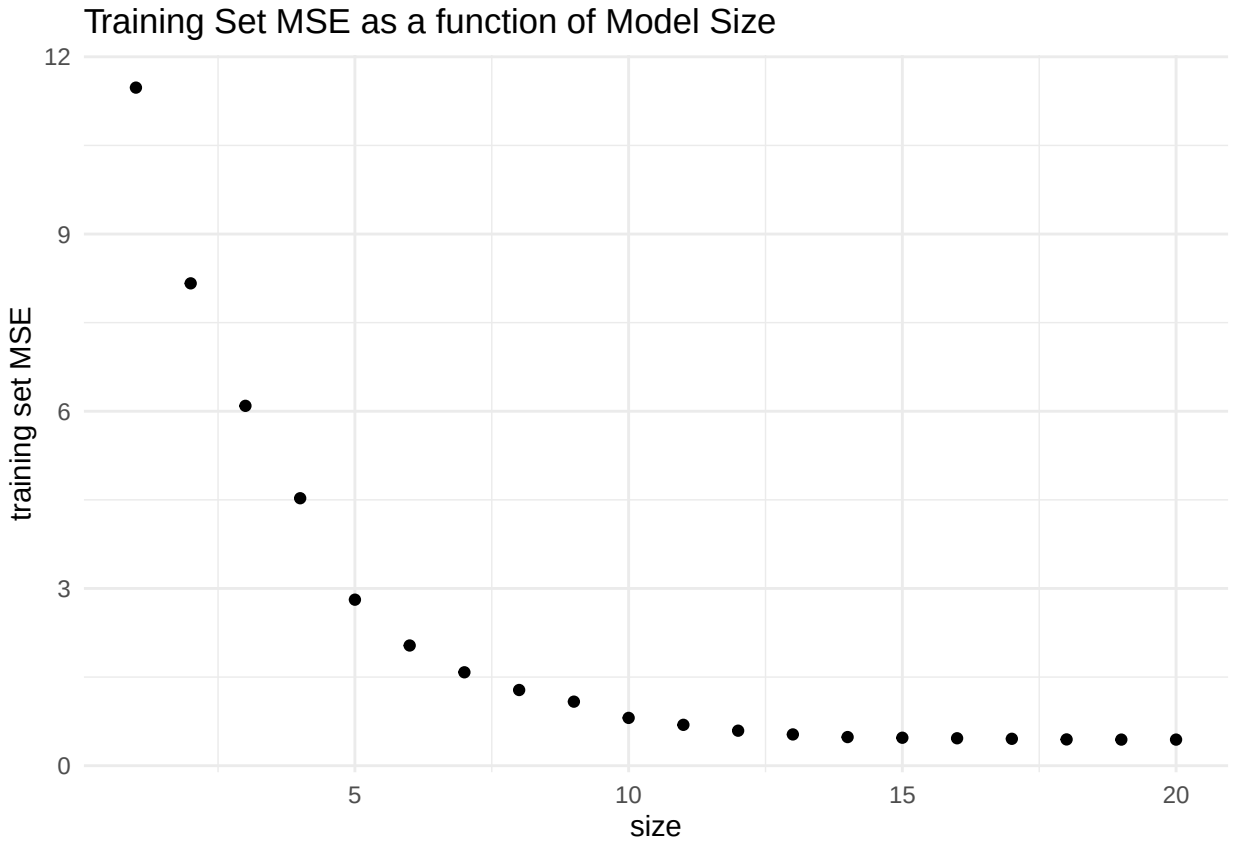
```
set.seed(50)
X = matrix(rnorm(1000 * 20), nrow = 1000, ncol = 20)
b = rnorm(20)
b[2] = 0
b[6] = 0
b[14] = 0
b[19] = 0
eps = rnorm(20)
Y = X %*% b + eps
```

(b) Split the dataset into a training set containing 100 observations and a test set containing 900 observations.

```r
indices = sample(1:1000, size = 100)
Xtrain = X[indices,]
Xtest = X[-indices,]
Ytrain = Y[indices]
Ytest = Y[-indices]
train = data.frame(x = Xtrain, y = Ytrain)
test = data.frame(x = Xtest, y = Ytest)
```

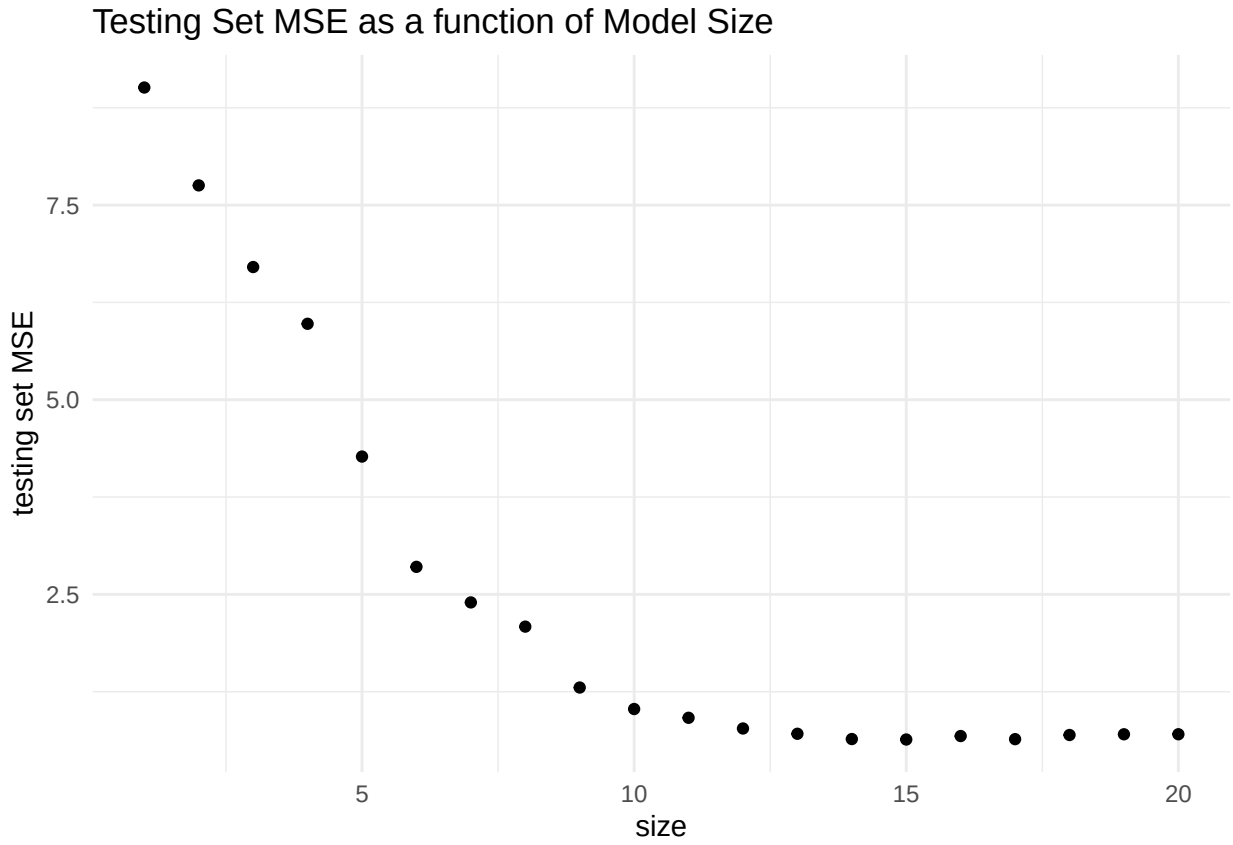
(c) Perform best subset selection on the training set and plot the training set MSE associated with the best model of each size.

```r
model = regsubsets(y~., train, nvmax = 20)
train_matrix = model.matrix(y~., train, nvmax = 20)
errors = c()
for(i in 1:20){
  temp = coef(model, id = i)
  predictions = train_matrix[, names(temp)] %*% temp
  errors = c(errors, mean((predictions - Ytrain)^2))
}
error_df = data.frame(size = seq(1,20,by=1), "training MSE" = errors)
ggplot(error_df, aes(x = size, y = training.MSE)) + geom_point() +
  ggtitle("Training Set MSE as a function of Model Size") +
  labs(y = "training set MSE") +
  theme_minimal()
```



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

```
test_matrix = model.matrix(y~., test, nvmax = 20)
errors2 = c()
for(i in 1:20){
  temp = coef(model, id = i)
  predictions = test_matrix[, names(temp)] %*% temp
  errors2 = c(errors2, mean((predictions - Ytest)^2))
}
error2_df = data.frame(size = seq(1,20,by=1), "testing MSE" = errors2)
ggplot(error2_df, aes(x = size, y = testing.MSE)) + geom_point() +
  ggtitle("Testing Set MSE as a function of Model Size") +
  labs(y = "testing set MSE") +
  theme_minimal()
```



(e) For which model size does the test set MSE take on its minimum value? Comment on the 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(errors2)
```

```
## [1] 15
```

The test set MSE takes on its minimum value at the variable size of 15.

(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(model, which.min(errors2))
```

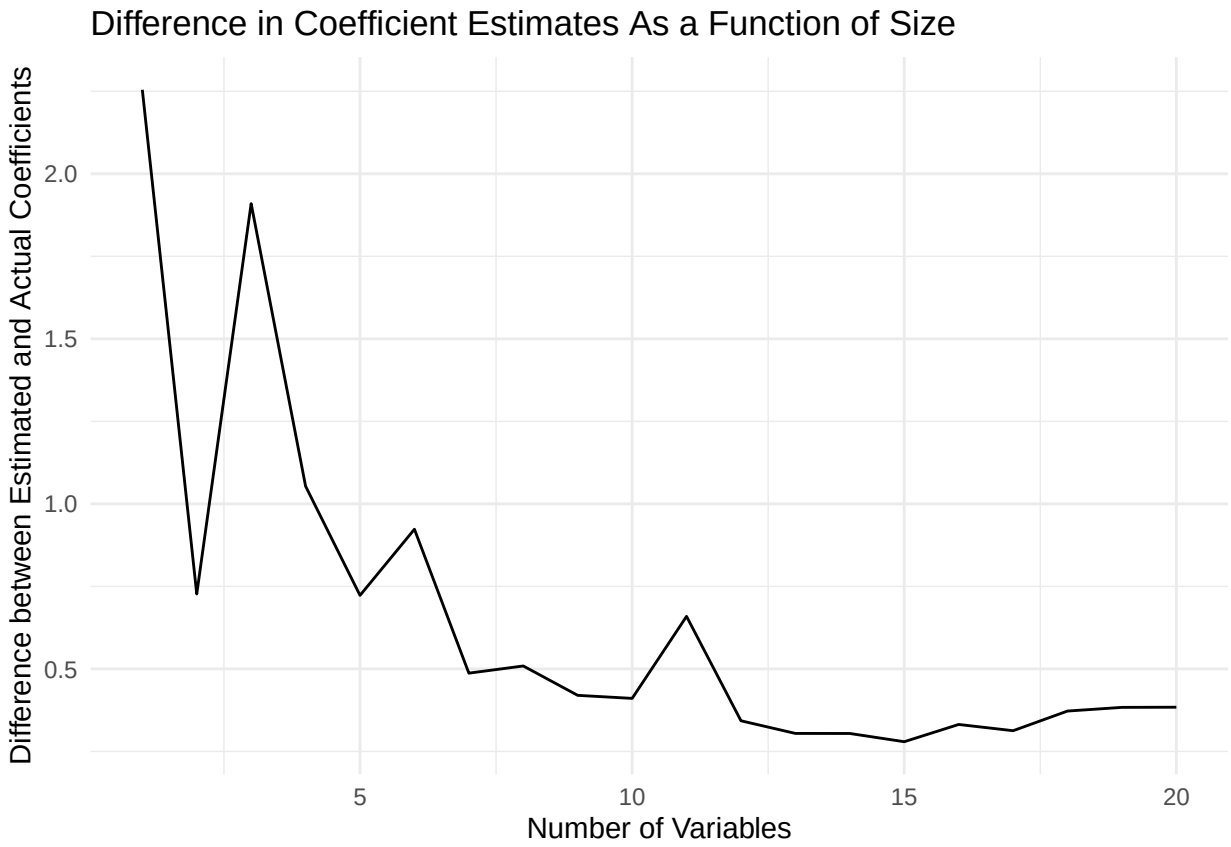
```
## (Intercept)         x.1         x.3         x.4         x.5         x.7
```

```
##  -0.2316827   0.3817065   0.2220810  -1.2138129  -0.2717723  -0.6888127
##         x.8         x.9        x.10        x.11        x.12        x.15
##   0.5856918  -0.2815707  -0.1110691  -1.1563543   1.2221562  -2.2863389
##        x.16        x.17        x.18        x.20
##   0.6647308   0.8830856   0.6678523   1.5963715
```

In the generation of the data, the constants for β_2, β_6, β_14 and β_19 were set to zero. The suggested model has captured this information.

(g) Create a plot displaying $\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_j^r)^2}$ 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 set MSE plot from (d)?

```
coef_diff = c()
xcols = colnames(X, do.NULL =  FALSE, prefix = "x.")
for(i in 1:20){
  temp = coef(model, id = i)
  coef_diff = c(coef_diff,
                sqrt(sum((b[xcols %in% names(temp)] -
                            temp[names(temp) %in% xcols])^2) +
                       sum(b[!(xcols %in% names(temp))])^2))
}
coeff_df = data.frame("num_coef" = seq(1,20,1), "coeff_diff" = coef_diff)
ggplot(coeff_df, aes(x = num_coef, y = coeff_diff)) + geom_path() +
  ggtitle("Difference in Coefficient Estimates As a Function of Size") +
  labs(x = "Number of Variables",
       y = "Difference between Estimated and Actual Coefficients") +
  theme_minimal()
```



As the number of variables increases, the total difference between estimated and actual coefficients decreases. There is a slight increase though when the number of variables is 11. Just like in the plot of the test set MSE, the minimum difference between estimated and actual coefficients occur when the number of variables is 15.

Question 11: Now try to predict per capita crime rate in the `Boston` dataset.

(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.

```
set.seed(11)
df = Boston
indices = sample(1:nrow(df), size = 0.7 * nrow(df))
train = df[indices,]
test = df[indices,]
mses = c()
```

First, validation set approach using regular linear regression.

```
model1 = lm(crim~., train)
mse_1 = c(mses, mean((predict(model1, test) - test$crim)^2))
mse_1
```

```
## [1] 47.22682
```

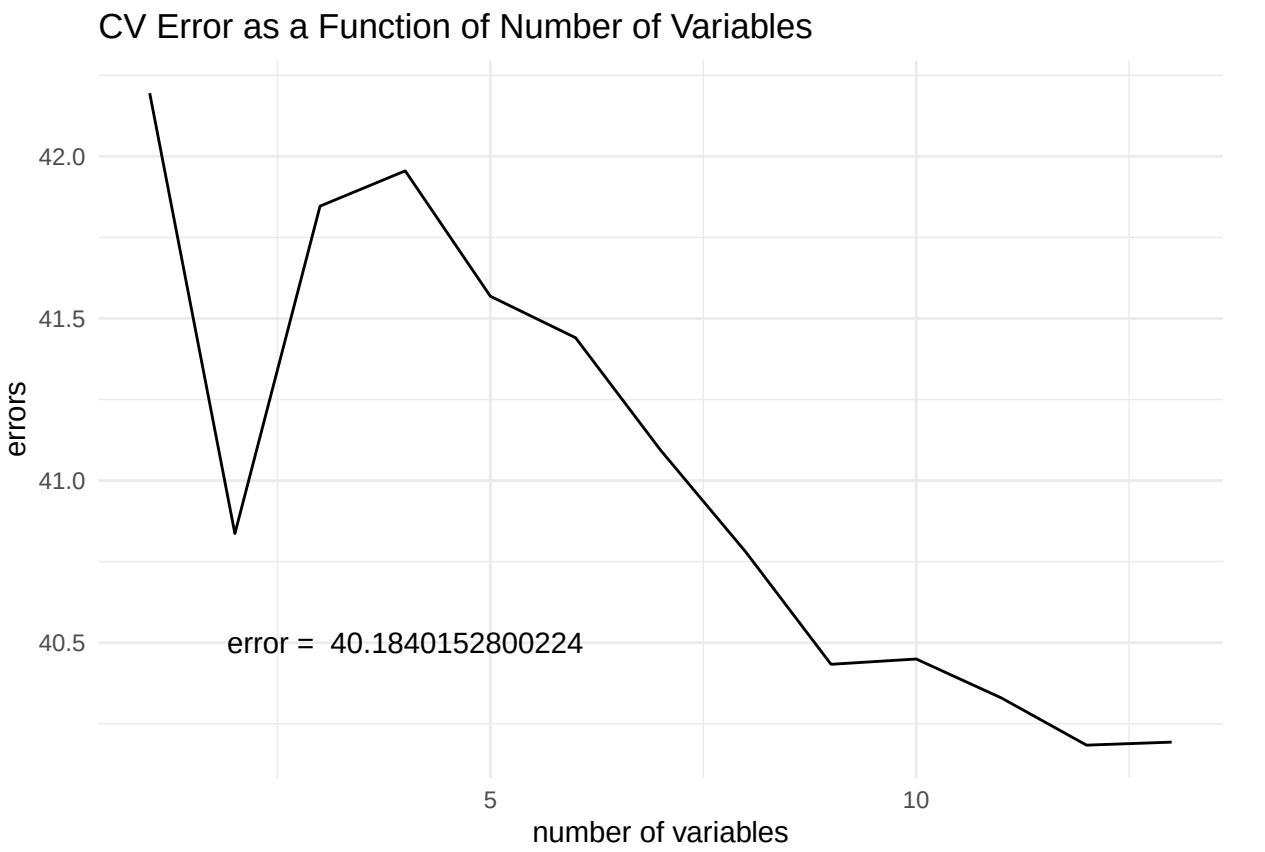
The test set MSE for linear regression is 47.22682. Now, best subset selection.

```
set.seed(11)
k = 10
folds = sample(1:k, nrow(Boston), replace = TRUE)
errors = matrix(NA, k, ncol(df)-1, dimnames = list(NULL, paste(1:13)))

predict.regsubsets <- function(object, newdata, id, ...) {
    form <- as.formula(object$call[[2]])
    mat <- model.matrix(form, newdata)
    coefi <- coef(object, id = id)
    xvars <- names(coefi)
    mat[, xvars] %*% coefi
}

for(i in 1:k){
  temp_model = regsubsets(crim~., data = df[folds != i,], nvmax = 13)
  for(j in 1:13){
    predictions = predict(temp_model, df[folds == i, ], id = j)
    errors[i,j] = mean((df$crim[folds == i] - predictions)^2)
  }
}
mean_errors = apply(errors, 2, mean)
mse_2 = mean_errors[which.min(mean_errors)]

error_df = data.frame(x = seq(1,13,by=1), errors = mean_errors)
ggplot(error_df, aes(x = x, y = errors)) + geom_path() +
  ggtitle("CV Error as a Function of Number of Variables") +
  labs(x = "number of variables") +
  theme_minimal() +
    annotate("text", x = 4, y = 40.5,
           label = paste("error = ", mse_2))
```



With the best subset selection approach, the best test set MSE is 40.184, when there are 12 variables.

Using the lasso regression,

```
set.seed(11)
model3 = cv.glmnet(model.matrix(crim ~ ., train)[, -1],
                   train$crim, alpha = 1, type.measure = "mse")
pred = predict(model3, s = model3$lambda.min,
               newx = model.matrix(crim~., test)[,-1])
mse_3 = mean((pred - test$crim)^2)
mse_3
```

```
## [1] 47.92685
```

Using the lasso, the test set MSE is 47.92685.

With the ridge regression,

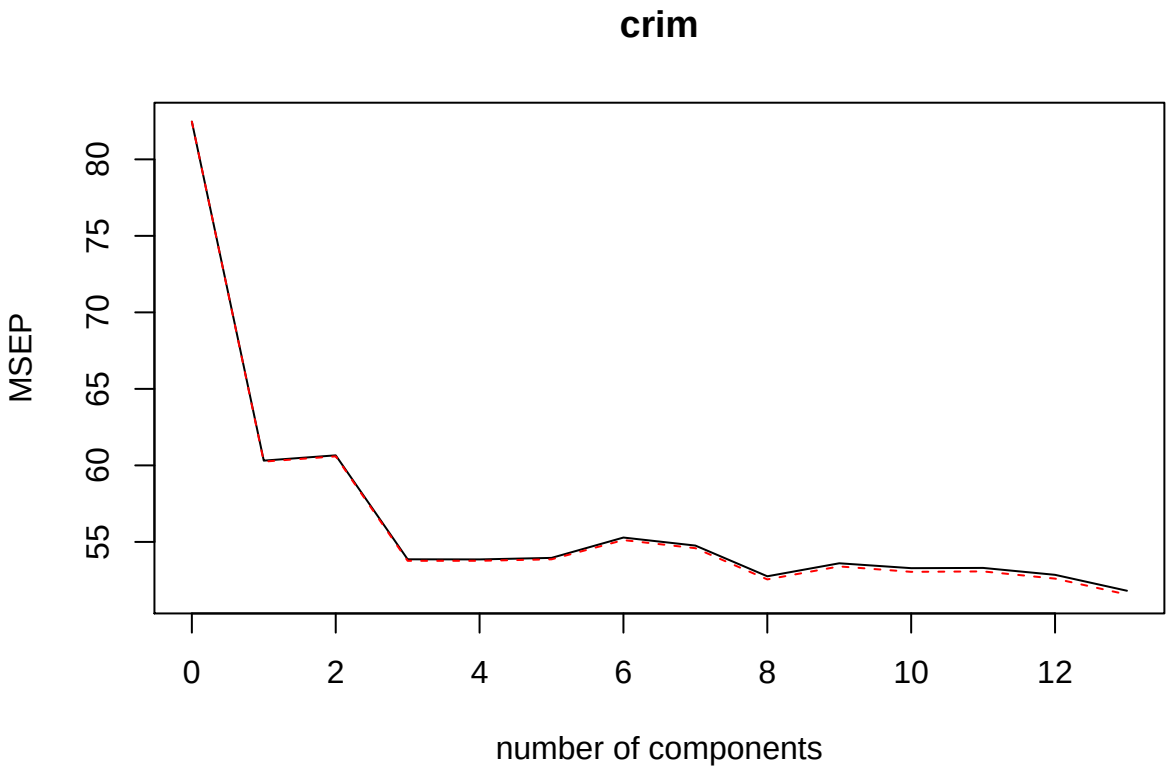
```
set.seed(11)
model4 = cv.glmnet(model.matrix(crim ~ ., train)[, -1],
                   train$crim, alpha = 0, type.measure = "mse")
pred = predict(model4, s = model4$lambda.min,
               newx = model.matrix(crim~., test)[,-1])
mse_4 = mean((pred - test$crim)^2)
mse_4
```

```
## [1] 47.85133
```

the test set MSE is 47.85133.

Now when using PCR,

```
model5 = pcr(crim ~ ., data = train, scale = TRUE, validation = "CV")
validationplot(model5, val.type = "MSEP")
```



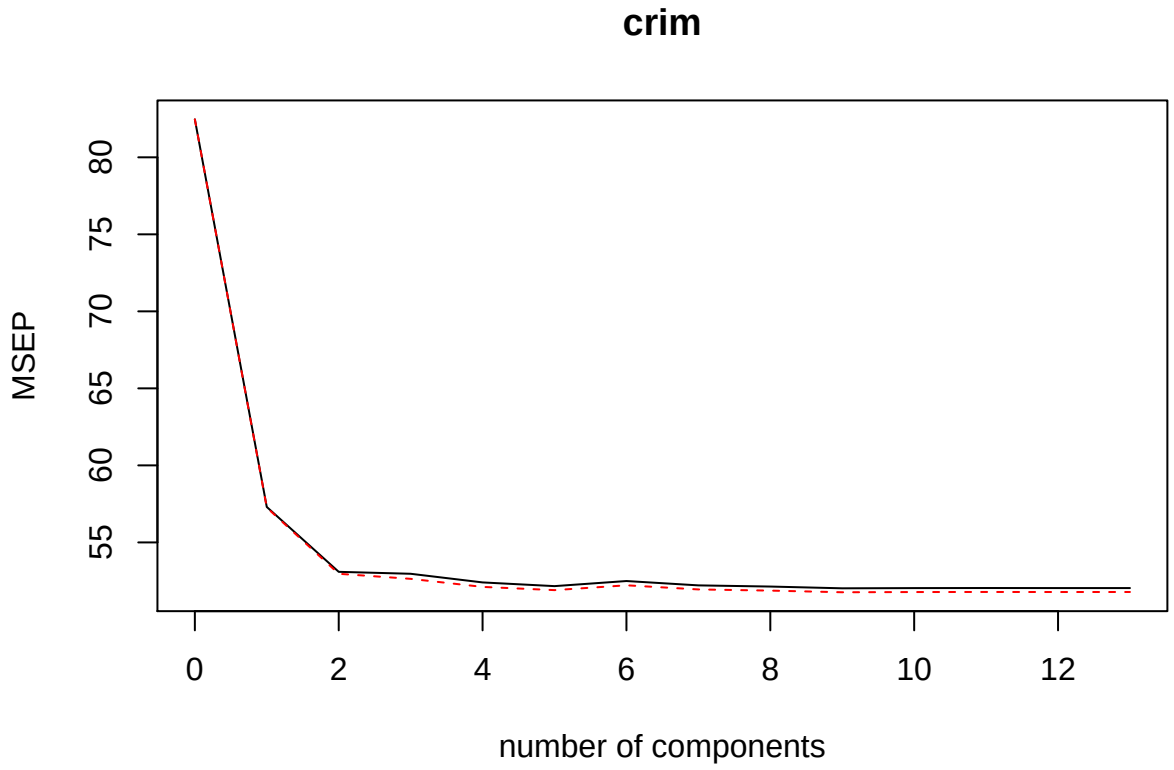

```
mse_5 = mean((test$crim - predict(model5, newdata = test[,-1], ncomp = 8))^2)
mse_5
```

```
## [1] 49.5358
```

The test set MSE when using only 8 components is 49.5358.

When using PLS,

```
model6 = plsr(crim ~ ., data = train, scale = TRUE, validation = "CV")
validationplot(model6, val.type = "MSEP")
```




```
mse_6 = mean((test$crim - predict(model6, newdata = test[,-1], ncomp = 5))^2)
mse_6
```

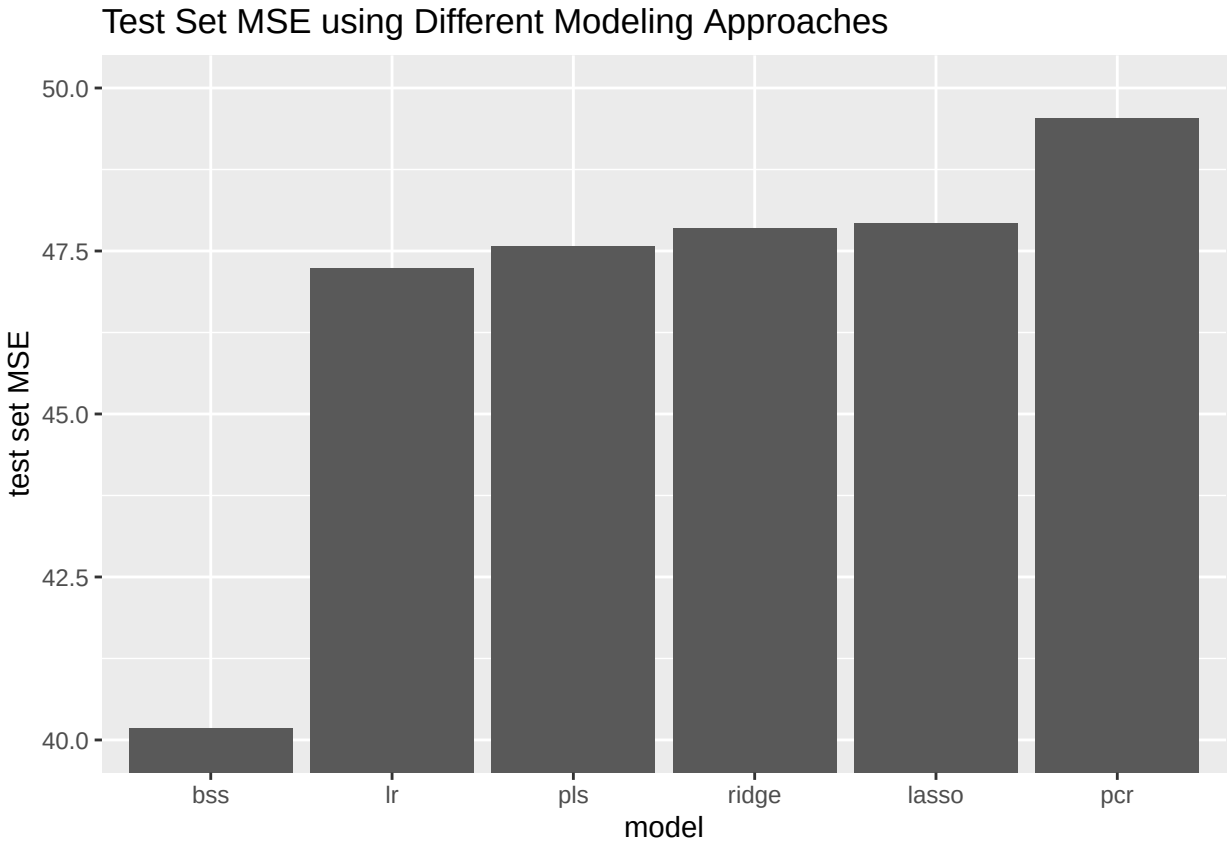
```
## [1] 47.56884
```

The test set MSE when using only 5 components is 47.56884.

(b) Propose a model (or set of models) that seem to perform well on this dataset 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.

```
mse_df = data.frame("model" = c("lr", "bss", "lasso", "ridge", "pcr", "pls"),
                    "test set mse" = c(mse_1, mse_2, mse_3, mse_4, mse_5, mse_6))

ggplot(mse_df, aes(x = reorder(model, test.set.mse), y = test.set.mse)) +
  geom_bar(stat = "identity") +
  labs(x = "model", y = "test set MSE") +
  ggtitle("Test Set MSE using Different Modeling Approaches") +
  coord_cartesian(ylim = c(40, 50))
```



The model that performed the best on this dataset was one created using best subset selection. The worst model was created using PCR followed by the lasso and ridge.

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

The model that did the best, best subset selection, used only 12 features.

All of the practice applied exercises in this document are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.