asn3

Question 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)
trains <- sample(1:nrow(College), nrow(College)/2)
train <- College[trains,]
test <- College[-trains,]
true_y <- test$Apps
```

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

```
## [1] 1343376
```

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

```
train.x <- model.matrix(Apps~., data=train)[,-1]
test.x <- model.matrix(Apps~., data=test)[,-1]
ridge <- cv.glmnet(train.x, train$Apps, alpha=0)

pred.ridge <- predict(ridge, s=ridge$lambda.min, newx=test.x)

print("Test Error")
```

```
## [1] "Test Error"
```

```
(test_error.ridge <- mean((true_y - pred.ridge)^2))
```

```
## [1] 2144380
```

```
print("Optimal Lambda")
```

```
## [1] "Optimal Lambda"
```

```
ridge$lambda.min
```

```
## [1] 347.3898
```

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

```
train.x <- model.matrix(Apps~., data=train)[,-1]
test.x <- model.matrix(Apps~., data=test)[,-1]
lasso <- cv.glmnet(train.x, train$Apps, alpha=1)
(lambda <- lasso$lambda.min)
```

```
## [1] 10.85842
```

```
pred.lasso <- predict(lasso, s=lasso$lambda.min, newx=test.x)
print("Test Error")
```

```
## [1] "Test Error"
```

```r
(test_error.lasso <- mean((true_y - pred.lasso)^2))
```

```
## [1] 1386030
```

```r
coef.lasso <- predict(lasso, type="coefficients", s=lambda)[1:ncol(College),]

print("Number of non-zero Coefficients")
```

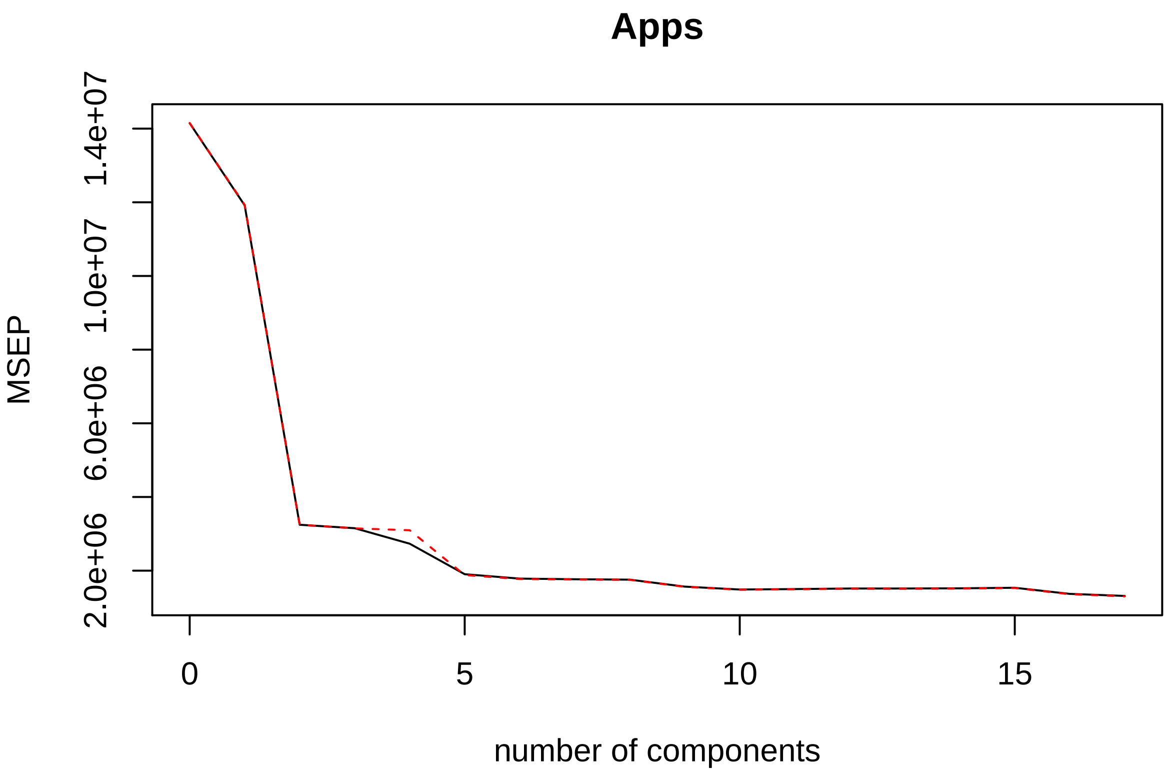
```
## [1] "Number of non-zero Coefficients"
```

```r
length(coef.lasso[coef.lasso != 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
fit.pcr <- pcr(Apps~., data=train, scale=TRUE, validation="CV")
validationplot(fit.pcr, val.type="MSEP")
```

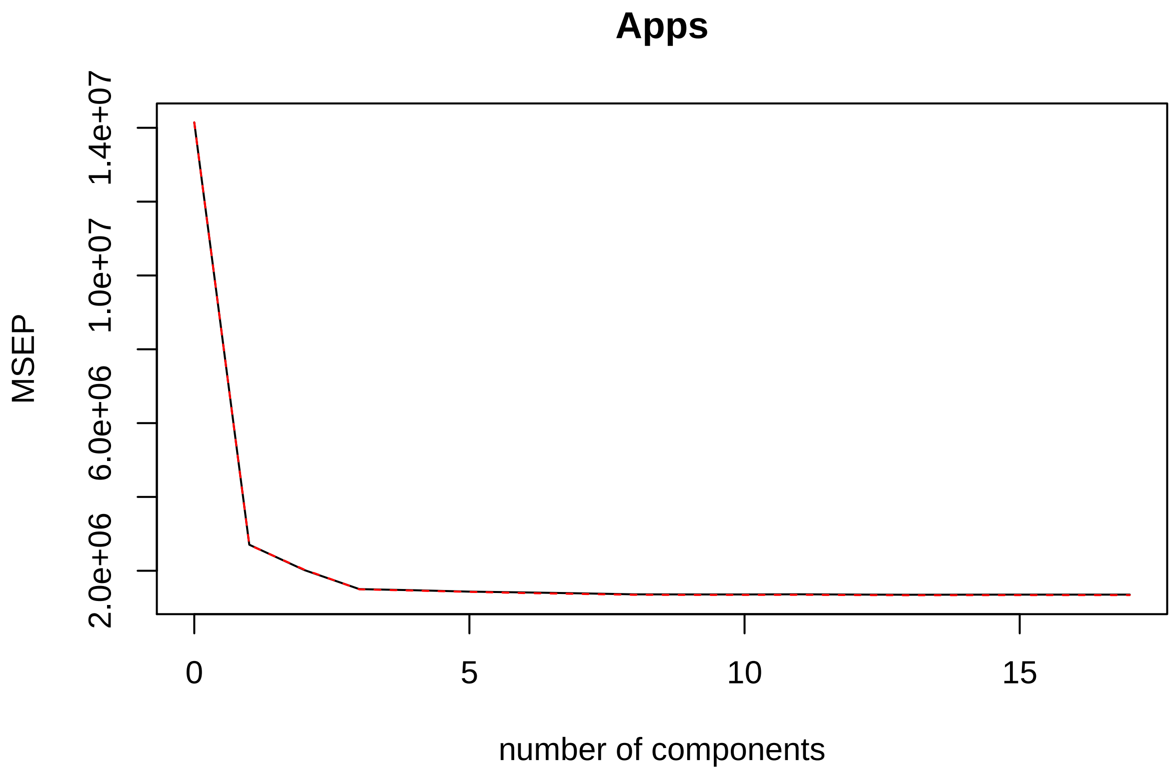



```r
#CV Score appears to me minimized at M = 16
pred.pcr <- predict(fit.pcr, test, ncomp=16)
(test_error.pcr <- mean((true_y - pred.pcr)^2))
```

```
## [1] 1548201
```

(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

```r
fit.pls <- plsr(Apps~., data=train, scale=TRUE, validation="CV")
validationplot(fit.pls, val.type="MSEP")
```

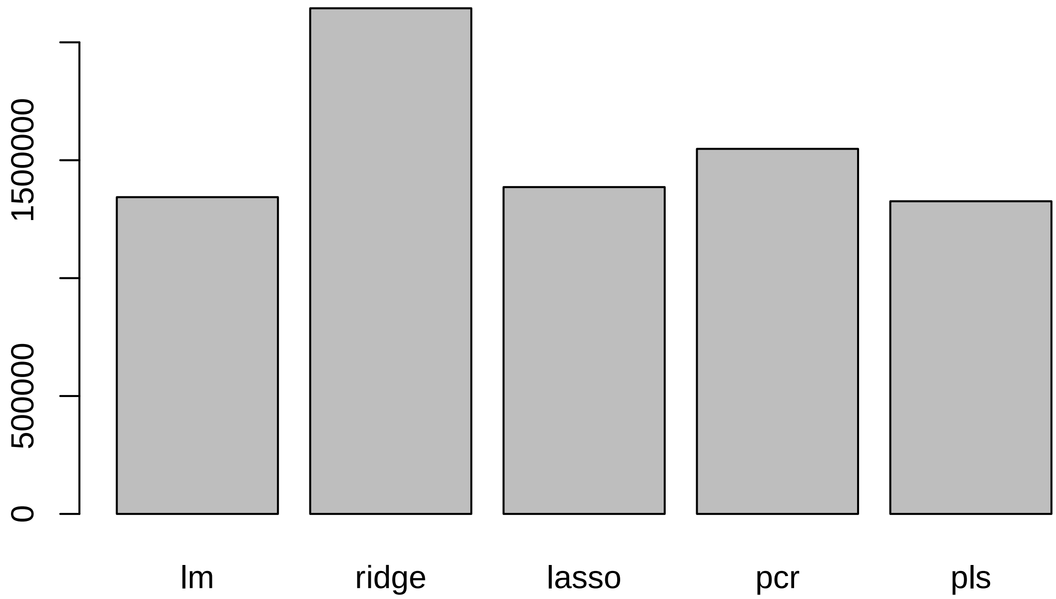



```
#CV Score appears to me minimized at M = 10
pred.pls <- predict(fit.pls, test, ncomp=10)
(test_error.pls <- mean((true_y - pred.pls)^2))
```

```
## [1] 1325931
```

(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?

```
error.total <- c(test_error.lm, test_error.ridge, test_error.lasso, test_error.pcr, test_error.pls)
names(error.total) <- c("lm", "ridge", "lasso", "pcr", "pls")
## Plot Test Errors
barplot(error.total )
```




```
# Based on the plot below, It looks like there is a huge difference between the models. Other than Ridg
```

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. 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 =XB+e,where B has some elements that are exactly equal to zero.

```
set.seed(4)

p = 20
n = 1000
x = matrix(rnorm(n * p), n, p)
# Set up betas
betas <- sample(-5:5, 20, replace=TRUE)
betas[c(4,12,19,7)] <- 0
e <- rnorm(n)

y <- x %*% betas + e
```

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

```
train = sample(seq(1000), 100, replace = FALSE)
train.y = y[train, ]
test.y = y[-train, ]
train.x = x[train, ]
test.x = x[-train, ]

train_set <- data.frame(y=train.y, train.x)
test_set <- data.frame(y=test.y, test.x)
```

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

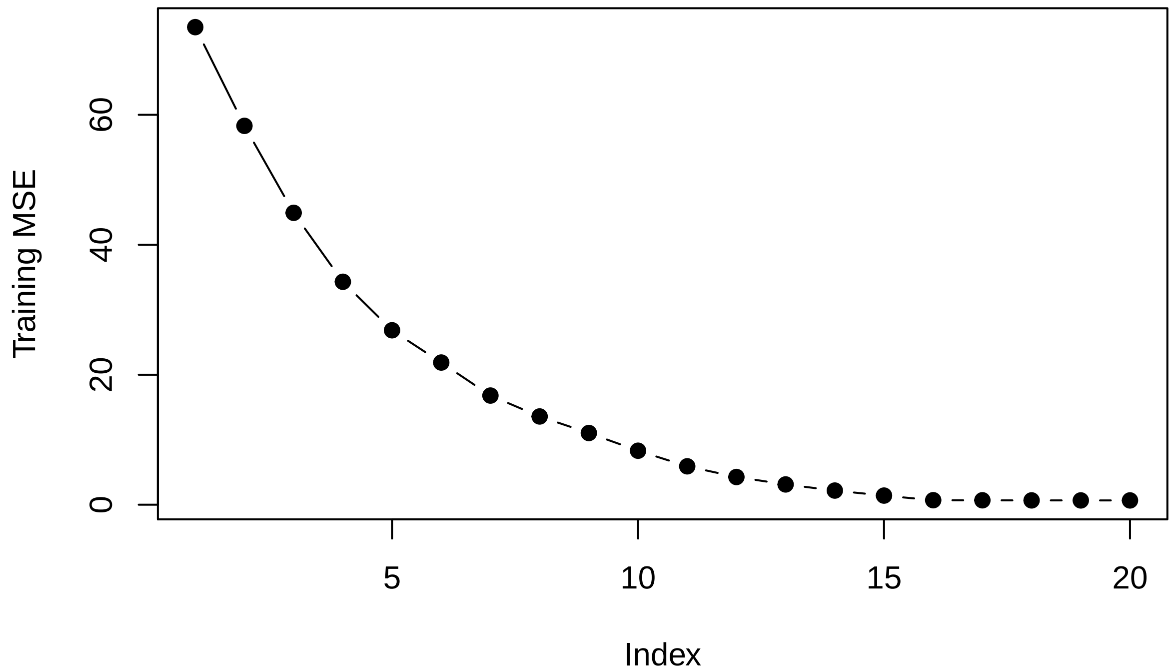
```r
# Fit model will all predictors
fit.all = regsubsets(y ~ ., data = data.frame(x = train.x, y = train.y), nvmax = p)

# Create emppty errors list and column
errors_c = rep(0, 20)
x_cols = colnames(x, do.NULL = FALSE, prefix = "x.")

for (i in 1:p) {
  # Get i'th coefficient from full model
  coeficients_c = coef(fit.all, id = i)

  # Make Prediction Using Matrix Multiplication
  pred_y = as.matrix(train.x[, x_cols %in% names(coeficients_c)]) %*% coeficients_c[names(coeficients_c

  #Store errors in list for plotting later on
  errors_c[i] = mean((train.y - pred_y)^2)
}
plot(errors_c, ylab = "Training MSE", pch = 19, type = "b")
```



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

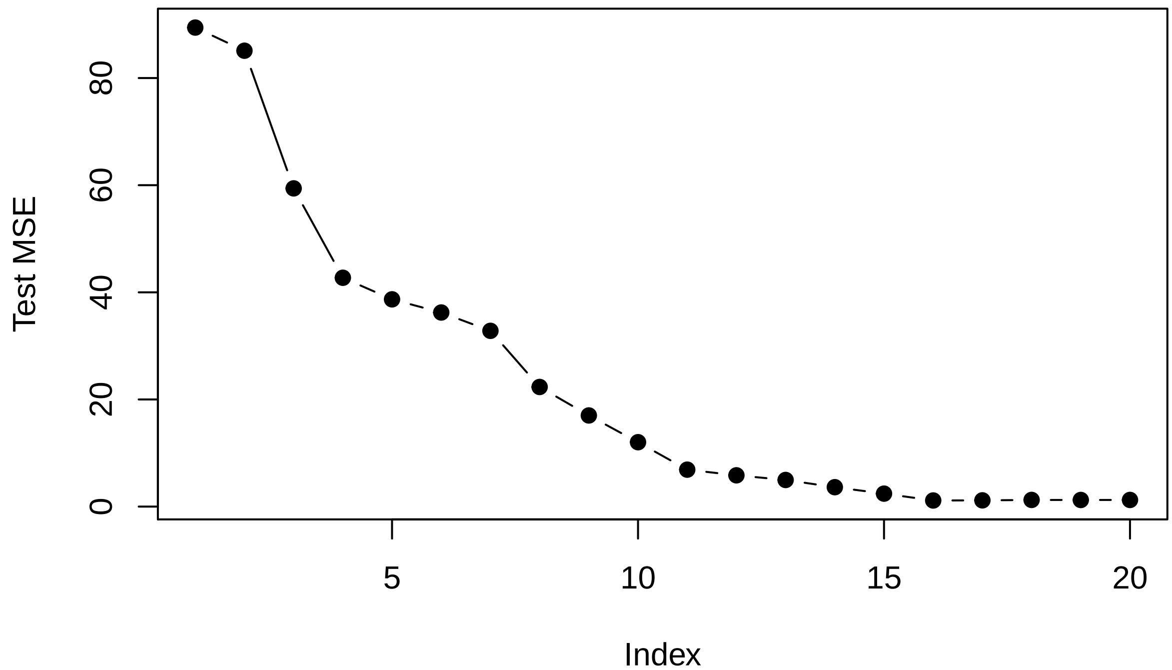
```r
# Create empty errors list and column
errors_d = rep(0, 20)

for (i in 1:p) {
  # Get i'th coefficient from full model
  coeficients_c = coef(fit.all, id = i)

  # Make Prediction Using Matrix Multiplication
  pred_y = as.matrix(test.x[, x_cols %in% names(coeficients_c)]) %*% coeficients_c[names(coeficients_c)

  #Store errors in list for plotting later on
  errors_d[i] = mean((test.y - pred_y)^2)
```

```r
}
plot(errors_d, ylab = "Test MSE", pch = 19, type = "b")
```



(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.

```r
which.min(errors_d)
```

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

```r
# The optimal number of parameters in this model is 16
```

(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.

```r
coef(fit.all, id=which.min(errors_d))
```

```
## (Intercept)         x.1         x.2         x.3         x.5         x.6
## -0.07410886  3.89337479 -0.91745510 -0.95770476  1.96585984  2.92992197
##         x.8         x.9        x.10        x.11        x.13        x.14
## -3.16132019  2.17064299  2.14765095  1.13148297 -2.92770432 -0.96705968
##        x.15        x.16        x.17        x.18        x.20
##  4.96170883  0.95643073  2.08359891 -1.96566387 -2.02506426
```