Chapter 6 Problem 9

Andira Putri

In this exercise, we will predict the number of applications received using the other variables in the `College` data set.

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

```
library(ISLR)
data(College) #contains 777 obs
set.seed(1)
train=sample(777,600) #600 obs for training
train.set=College[train,]
test.set=College[-train,] #177 obs for testing
```

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

- Background on Least Squares Regression

Least squares regression attempts to find coefficients that minimize residual sum of squares (RSS). Ol' reliable.

```
#fit least squares model
lin.reg=lm(Apps~.,data=train.set)
#use test set on least squares model, make predictions
lin.pred=predict(lin.reg,test.set)
#calculate test error, just the general MSE formula
MSE.lin=mean((test.set$Apps-lin.pred)^2)
MSE.lin #And our test error is...
```

```
## [1] 1101792
```

c.) Fit a ridge regression model on the training set, with the tuning parameter chosen by cross-validation, and report the test error obtained.

- Background on Ridge Regression:

Ridge regression works almost like least squares regression, but the coefficients are estimated by minimizing a quantity slightly different than least squares. Least squares attempts to minimize just RSS. Ridge regression, on the contrary, attempts to minimize:

$RSS + \lambda \sum_{i=1}^{p} \beta_i^2$

The term $\lambda \sum_{i=1}^{p} \beta_i^2$ is called a shrinking penalty, and it differentiates ridge regression from least squares. $\lambda \geq 0$ is a tuning parameter that controls the impact of the shrinking penalty on the regression coefficient estimates. When $\lambda = 0$, the penalty has no effect, and it's just a least squares problem. As $\lambda \to \infty$, the ridge regression estimates will shrink towards (but never reaches) zero since the impact of the penalty is so large. Unlike least squares, ridge regression generates several sets of coefficients from varying values of λ. Selecting a good value for λ is important, and we do this via cross-validation methods.

Noteworthy: $||\beta_2|| = \sqrt{\sum_{i=1}^{j} \beta_i^2}$ is known as an ℓ_2 norm.

```
library(glmnet)
```

```
#initialize matrices of training/test data sets
train.mat=model.matrix(Apps~.,data=train.set) #x-matrix
```

```r
test.mat=model.matrix(Apps~.,data=test.set) #y-vector
#initialize grid of possible lambda values, from 10^10 to 10^-2
lambda=10^seq(10,-2,length=100)
#fit ridge regression model (alpha=0 -> ridge regression)
ridge=glmnet(train.mat,train.set$Apps,alpha=0,lambda=lambda)
#perform cross-validation to find best lambda
ridge.cv=cv.glmnet(train.mat,train.set$Apps,alpha=0,lambda=lambda)
lambda.best=ridge.cv$lambda.min
lambda.best #And the best lambda is...
```

```
## [1] 0.01
```

```r
#Compute test error
ridge.pred=predict(ridge,test.mat,alpha=0,s=lambda.best)
ridge.MSE=mean((test.set$Apps-ridge.pred)^2)
ridge.MSE
```

```
## [1] 1101724
```

Our test error is slightly lower than that of least squares regression.

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

- Background on the Lasso:

Ridge regression isn't perfect- unlike best subset selection and forward/backwards stepwise selection, ridge regression includes all `p` predictors in the final model. Even though ridge regression shrinks coefficient estimates, it will never go down to 0, thus eliminating a predictor. Is this a problem for prediction accuracy? Not quite… but it does pose problems for interpretability. The lasso attempts to overcome this disadvantage by being able to generate parse models. Lasso coefficient estimates seek to minimize the quantity:

$RSS + \lambda \sum_{i=1}^{p} |\beta_i|$

Lasso features an ℓ_1 norm; $||\beta_1|| = \sum |\beta_i|$. Unlike the ℓ_2 norm in ridge regression, the ℓ_1 norm in lasso is able to shrink coefficients down to zero when λ is large enough. λ is typically chosen through cross-validation.

```r
#coding for lasso works the same way as ridge regression
#only difference: set alpha=1
#we use the same train.mat,test.mat,and lambda grid in part (c)

#fit lasso model
lasso=glmnet(train.mat,train.set$Apps,alpha=1,lambda=lambda)
#perform cross-validation to find best lambda
lasso.cv=cv.glmnet(train.mat,train.set$Apps,alpha=1,lambda=lambda)
lambda.best=lasso.cv$lambda.min
lambda.best
```

```
## [1] 24.77076
```

```r
#Compute test error
lasso.pred=predict(lasso,test.mat,alpha=1,s=lambda.best)
lasso.MSE=mean((test.set$Apps-lasso.pred)^2)
lasso.MSE
```

```
## [1] 1144529
```

So far, our test MSEs are in the following order: Ridge Regression < Least Squares < Lasso

e.) Fit a PCR model on the training set, with M (number of components) chosen by cross-validation, and report the test error obtained, along with the value of M selected by cross-validation.

- Background on Principal Components Regression (PCR):

When you have a large number of predictors that are closely related to each other, principal components regression aims to summarize this set with a smaller number of representative variables. These representative variables are special in that they are somehow able to capture the variability in the data set. Think about it this way…each of the n observations lives in a p-dimensional space, but not all dimensions are interesting. In our case, "interest" is measured by how much the data varies in that dimension. PCR finds the most interesting dimensions for us so we're not looking at all p-predictors; the dimension of the problem is reduced as a result.

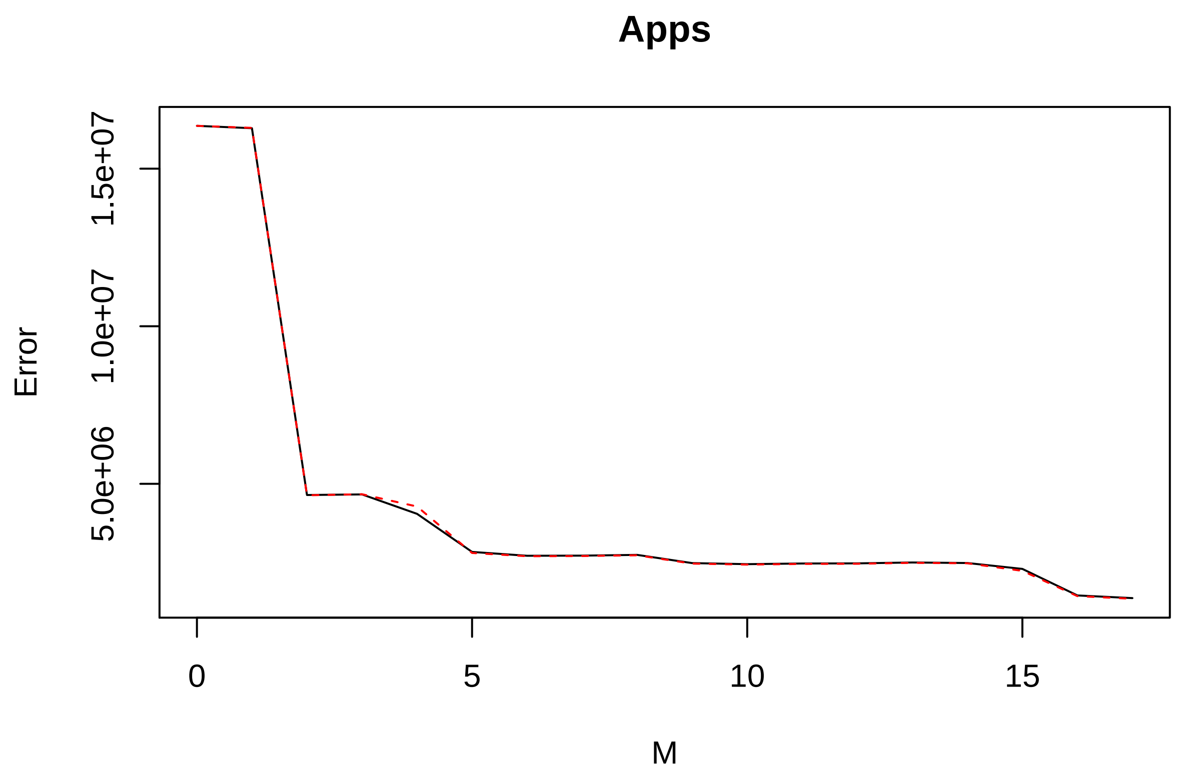
Here is a link to a beautiful visualization of principal components analysis (PCA). I found this incredibly helpful: http://setosa.io/ev/principal-component-analysis/

```
library(pls)
```

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

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

```
#fit PCR model
pcr=pcr(Apps~.,data=train.set,scale=TRUE,validation="CV")
#We use the plot and summary to choose the best M
validationplot(pcr,val.type="MSEP",xlab="M",ylab="Error")
```



```r
summary(pcr)
```

```
## Data:    X dimension: 600 17
##  Y dimension: 600 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            4045     4035     2155     2160     2012     1685     1648
## adjCV         4045     4037     2153     2161     2069     1675     1644
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1650     1657     1576      1565      1572      1574      1582
## adjCV     1646     1653     1571      1561      1568      1569      1579
##        14 comps  15 comps  16 comps  17 comps
## CV         1577      1516      1207      1172
## adjCV      1574      1496      1198      1163
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       31.18    57.70    64.68    70.32    75.80    80.72    84.34
## Apps     1.01    72.29    72.30    73.85    83.83    84.48    84.49
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X       87.66    90.57     92.96     95.06     96.87     97.96     98.75
## Apps    84.49    86.01     86.21     86.27     86.27     86.30     86.49
##        15 comps  16 comps  17 comps
## X         99.37     99.85    100.00
## Apps      90.94     93.23     93.53
```

I'm thinking our optimal M is 10!

Before explaining why M=10, please note that I set a cut-off at M=15. An M value close to the total number of predictors is essentially performing least squares; no dimension reduction occurs in that case. With that being said…from the plot, 10 is the point where the error seems to be at its lowest leveling-off point. From the CV results found in the summary, if you go from 5 to 14 components, there's a dip in estimated error at 10.

```
#Compute error with M=10
pcr.pred=predict(pcr,test.set,ncomp=10)
pcr.MSE=mean((test.set$Apps-pcr.pred)^2)
pcr.MSE
```

```
## [1] 1609171
```

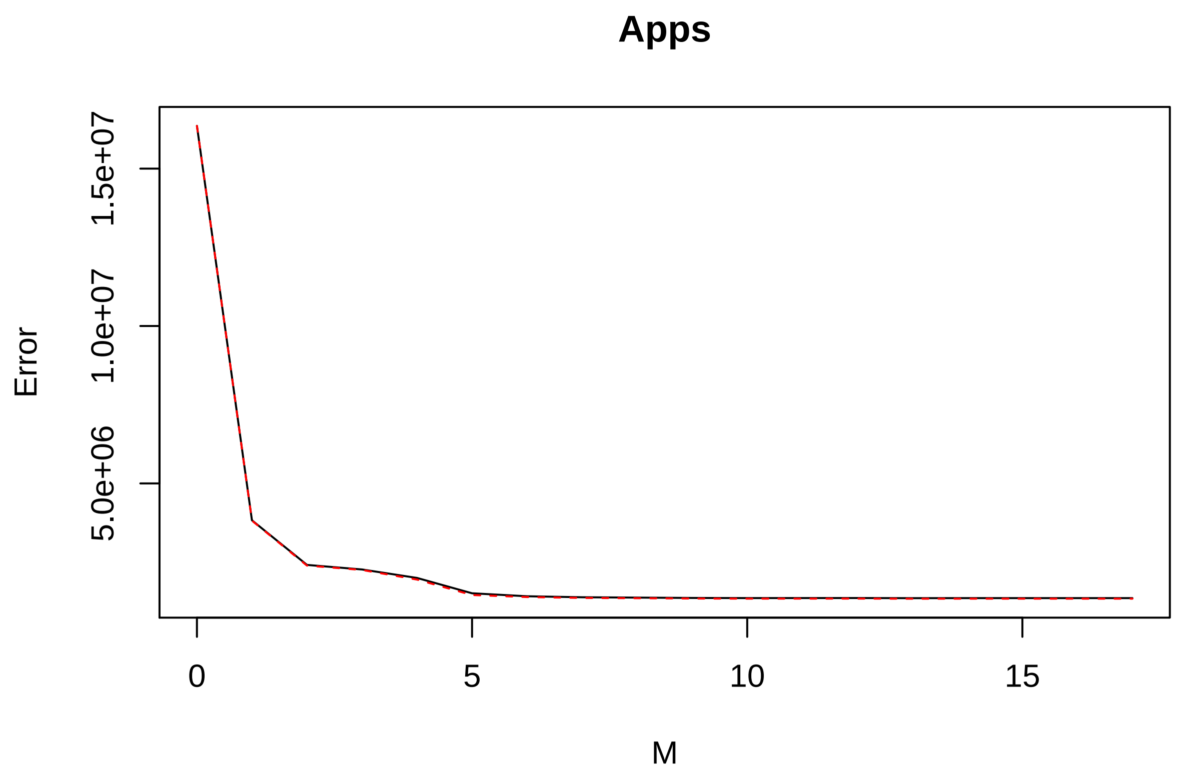
Our MSE from PCR is quite higher than our other methods thus far!

f.) Fit a PLS model on the training set, with M chosen by cross-validation, and report the test error obtained, along with the value of M selected by cross-validation.

- Background on Partial Least Squares (PLS):

Partial least squares works the same way as PCR almost, but it makes use of the response vector Y to identify new features. Basically, it's the supervised alternative to PCR. PLS and PCR are at equal levels of performance and advantages, so the decision rests on what type of data you're looking at!

```
#fit PCR model
pls=plsr(Apps~.,data=train.set,scale=TRUE,validation="CV")
#We use the plot and summary to choose the best M
validationplot(pls,val.type="MSEP",xlab="M",ylab="Error")
```



```r
summary(pls)
```

```
## Data:    X dimension: 600 17
##  Y dimension: 600 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            4045     1957     1553     1505     1413     1229     1189
## adjCV         4045     1954     1546     1500     1395     1206     1179
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1176     1172     1167      1165      1166      1166      1164
## adjCV     1167     1163     1159      1157      1157      1158      1156
##        14 comps  15 comps  16 comps  17 comps
## CV         1165      1165      1165      1165
## adjCV      1157      1157      1157      1157
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       26.45    34.74    62.77    65.08    67.25    72.08    76.70
## Apps    77.50    86.92    88.01    91.42    93.18    93.34    93.39
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X       79.97    82.64     86.73     88.31     90.78     92.44     94.02
## Apps    93.44    93.48     93.50     93.52     93.53     93.53     93.53
##       15 comps  16 comps  17 comps
## X        95.90     98.18    100.00
## Apps     93.53     93.53     93.53
```

Like in PCR, I choose M to be 10.

```
#Compute error with M=10
pls.pred=predict(pls,test.set,ncomp=10)
pls.MSE=mean((test.set$Apps-pls.pred)^2)
pls.MSE
```

```
## [1] 1100821
```

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?

Here are our test MSEs:

- Least squares - 1101792
- Ridge Regression - 1101724
- Lasso - 1144529
- PCR - 1609171
- PLS- 1100821

From least to greatest, here are the performances of each methodology in terms of minimizing error:

PLS < Ridge Regression < Least Squares < Lasso < PCR