Chapter 6 Problem 11

Andira Putri

We will now try to predict per capita crime rate in the `Boston` data set.

```
library(MASS)
data(Boston)
```

Try out some of the regression methods explored in this chapter. Present and discuss results for the approaches you consider. Propose a model (or set of models) that seem to perform well on the 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. Does your chosen model involve all features of the data set? Why or why not?

The methods introduced in this chapter are:

- Best subset selection
- Forward stepwise selection
- Backawards stepwise selection
- Ridge regression
- The lasso
- Principal components regression (PCR)
- Partial least squares (PLS)

I will use backwards stepwise selection, the lasso, and PCR.

Backwards Stepwise Selection (BSS)

BSS is a form of subset selection, which identifies a subset of the p predictors we believe are related to the response. It allows us to fit a model with a reduced set of variables using least squares. Best subset selection is not computationally feasible because it considers 2^p models. Alternatives are forward stepwise selection and backwards stepwise selection. In forward stepwise selection, you begin with a null model (no predictors) and add independent variables one at a time for analysis.In BSS, you start from the full model (all predictors), look at all k models that contain all but one of the predictors in the full model, and choose the best among them. You can only use backwards stepwise selection when p<n.

To evaluate models selected by BSS, we make use of C_p, BIC, and adjusted R^2. Without going into detail, we just need to pick models with small C_p and BIC values and a high adjusted R^2.

```
#backwards subset selection
library(leaps)
bss=regsubsets(crim~.,data=Boston,nvmax=14,method="backward")
summary(bss)
```

```
## Subset selection object
## Call: regsubsets.formula(crim ~ ., data = Boston, nvmax = 14, method = "backward")
## 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: backward
##           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 ) "*" "*"   "*"  "*" "*" "*" "*" "*" "*" "*" "*"     "*"   "*"   "*"
```

```
#amount of predictors with smallest Cp
summary=summary(bss)
which.min(summary$cp)
```

```
## [1] 8
```

```
#amount of predictors with smallest BIC
which.min(summary$bic)
```

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

```
#amount of predictors with highest adjR2
which.min(summary$adjr2)
```

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

The best models suggested by backwards subset selection are:

Y=$\beta_0 + \beta_1(zn) + \beta_2(nox) + \beta_3(dis) + \beta_4(rad) + \beta_5(ptratio) + \beta_6(black) + \beta_7(lstat) + \beta_8(medv)$ from the eight-predictor model having the lowest C_p value.

Y=$\beta_0 + \beta_1(zn) + \beta_2(dis) + \beta_3(rad) + \beta_4(medv)$ from four-predictor model having the lowest BIC value.

Y=$\beta_0 + \beta_1(rad)$ from the one-predictor model having the highest adjusted R^2.

All predictors are selected through the `summary(bss)` output. In the bottom table of the summary, the left-hand column represents the number of predictors in a model, and the * represents the variables best suited for that particular number of predictors.

```
#we test all the proposed linear models above
#split data into training set and test set
set.seed(1)
```

```r
train=sample(506,400) #400 obs for training
train.set=Boston[train,]
test.set=Boston[-train,] #106 obs for testing
#fit using least squares regression
lin.fit8=lm(crim~zn+nox+dis+rad+ptratio+black+lstat+medv,data=train.set)
lin.fit4=lm(crim~zn+dis+rad+medv,data=train.set)
lin.fit1=lm(crim~rad,data=train.set)
#make predictions
lin.pred8=predict(lin.fit8,test.set)
lin.pred4=predict(lin.fit4,test.set)
lin.pred1=predict(lin.fit1,test.set)
#calculate test errors with just the general MSE formula
#MSE of eight-predictor model
mean((test.set$crim-lin.pred8)^2)
```

```
## [1] 44.90358
```

```r
#MSE of four-predictor model
mean((test.set$crim-lin.pred4)^2)
```

```
## [1] 45.19013
```

```r
#MSE of one-predictor model
mean((test.set$crim-lin.pred1)^2)
```

```
## [1] 47.00248
```

The Lasso

If we're going to talk about the lasso, it's important to give some background on ridge regression beforehand. 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.

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
library(glmnet)
```

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

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
#initialize matrices of training/test data sets
train.mat=model.matrix(crim~.,data=train.set) #x-matrix
test.mat=model.matrix(crim~.,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 lasso model
lasso=glmnet(train.mat,train.set$crim,alpha=1,lambda=lambda)
#perform cross-validation to find best lambda
lasso.cv=cv.glmnet(train.mat,train.set$crim,alpha=1,lambda=lambda)
lambda.best=lasso.cv$lambda.min
lambda.best
```

```
## [1] 0.07054802
```

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

```
## [1] 44.96868
```

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.

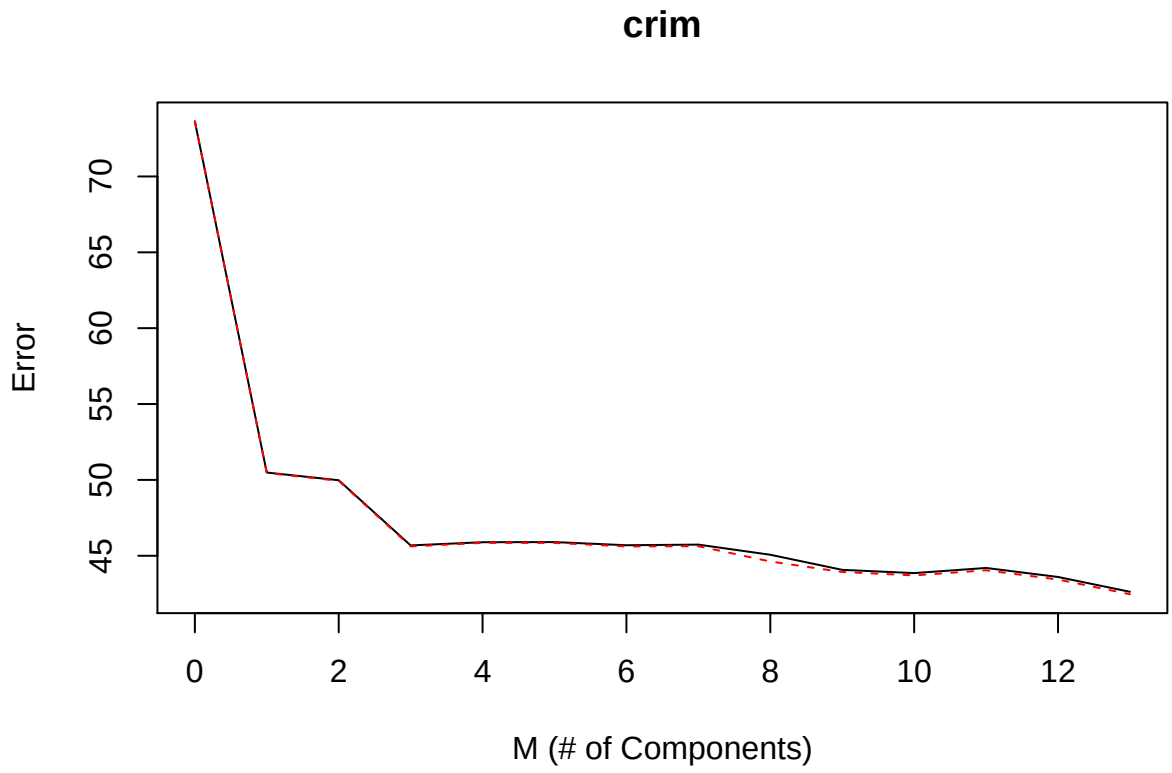
Here is a link to a beautiful visualization of principal components analysis (PCA). I found this incredibly helpful: http://setosa.io/ev/principal-component-analysis/

```
library(pls)
```

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

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

```
#fit PCR model
pcr=pcr(crim~.,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 (# of Components)",ylab="Error")
```




```
summary(pcr)
```

```
## Data:    X dimension: 400 13
##  Y dimension: 400 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           8.581    7.105    7.069    6.759    6.774    6.775    6.760
## adjCV        8.581    7.103    7.067    6.754    6.771    6.771    6.754
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       6.763    6.713    6.639     6.623     6.648     6.603     6.529
## adjCV    6.755    6.680    6.628     6.611     6.637     6.591     6.517
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       47.22    60.48    69.24    76.14    82.83    87.83    91.07
## crim    31.81    32.62    38.56    38.58    38.67    39.65    40.33
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X       93.31    95.44     97.12     98.54     99.54    100.00
## crim    42.84    43.24     43.62     43.71     44.74     45.95
```

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

```
pcr.MSE
```

```
## [1] 47.2354
```

Our errors for each method (obtained through cross-validation) are shown below:

BSS 8-predictor model: 44.90358

BSS 4-predictor model: 45.19013

BSS 1-predictor model: 47.00248

The Lasso: 44.96868

PCR: 47.2354

The best model is the 8-predictor model chosen via backwards subset selection.