415 Lab 6

April Cho

2/15/2018

Today's Objectives

1. Review how to scale data for KNN
2. Learn how to run cross validation in R
3. Learn how to run different subset selection methods for linear regression in R

Review - Scaling data

In previous lab, we learned a scale() function to standardize a data set when we run KNN. What scale() function does is to compute the mean and standard deviation of each predictor, and calculate standardized value for each data point x_i.

$$x_i^{std} = \frac{x_i - mean(x)}{sd(x)}$$

You can use scale() function if all we want to do is just fitting a KNN model on a data set. Now, what if we have training and test data? How should we scale them?

We no longer can use scale() function in this case since we want to standardize data in the same way. In other words, we need to use same mean and standard deviation to transform both training and test data.

What we have to do is to use training mean and training standard deviation to scale both training data and test data.

$$x_i^{stdtrain} = \frac{x_i^{train} - mean(x^{train})}{sd(x^{train})}$$

$$x_i^{stdtest} = \frac{x_i^{test} - mean(x^{train})}{sd(x^{train})}$$

We can do this by writing our own function.

```r
standardize <- function(train, test){
  m = apply(train, 2, mean) # colMeans(train) does the same thing
  s = apply(train, 2, sd)

  #Use train mean and sd to standardize both training and test data
  std_train <- (t(train) - m)/s
  std_test <- (t(test) - m)/s

  #returning multiple objects at once
  return(list(train = t(std_train), test = t(std_test)))
}
```

Notice that we are using a function apply(). Apply function is very useful when we want to do same operation repetitively on each column or each row. It is much faster and more concise than using a for-loop.

- The first argument is a data set that an operation

- For second argument, we can put a vector giving the subscripts which the function will be applied over. E.g., 1 indicates rows ('Run function on each row'), 2 indicates columns ('Run function on each column'), and c(1,2) indicates rows and columns('Run function on each row and each column').

- The third argument is the function to be applied. You can use a R's built-in functions or your own R function.

The function t() transposes a matrix. It was used here since operations with vector m and s is done for each column of a data set.

Let's standardize the Auto data we used for homework. As an exercise, let's choose 4 columns of Auto data as predictors.

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.2.5
```

```
set.seed(12345)
train_id <- sample(1:nrow(Auto), size = trunc(0.8 * nrow(Auto)))
Auto_train <- Auto[train_id,3:6]
Auto_test <- Auto[-train_id,3:6]

#Now let's standardize.
std_data <- standardize(Auto_train, Auto_test)
std_train <- std_data$train
std_test <- std_data$test
head(std_train)
```

```
##     displacement horsepower      weight acceleration
## 285    0.2958876  0.1657731  0.45328481    0.3537658
## 347   -0.9289344 -0.9431434 -1.06937874    0.7818406
## 299    1.4920027  0.5526044  1.08821787    0.6391490
## 349   -1.0054857 -1.0720872 -1.08701577    0.6034761
## 180   -0.6992802 -0.1436920 -0.03467301   -0.3953652
## 66     1.5015716  1.2746896  1.35747652   -0.9304587
```

```
head(std_test)
```

```
##    displacement horsepower     weight acceleration
## 10    1.8747595  2.2288736  1.0294278  -2.53573940
## 12    1.3963135  1.4552109  0.7460595  -2.71410392
## 17    0.0470956 -0.1694807 -0.2357351  -0.03863615
## 20   -0.9289344 -1.4847073 -1.3398132   1.74500903
## 22   -0.8332451 -0.3500020 -0.6402110  -0.39536518
## 23   -0.8619519 -0.2210583 -0.7048801   0.67482192
```

```
round(colMeans(std_train),4)
```

```
## displacement   horsepower       weight acceleration
##            0            0            0            0
```

```
round(colMeans(std_test),4)
```

```
## displacement   horsepower       weight acceleration
##       0.0158       0.1148       0.0181      -0.1186
```

The Validation Set Approach

We explore the use of the validation set approach in order to estimate the test error rates that result from fitting various linear models on the `Auto` data set. Before we begin, we use the `set.seed()` function in order to set a seed for seed `R`'s random number generator, so that you will obtain precisely the same results as those shown below. It is generally a good idea to set a random seed when performing an analysis such as cross-validation that contains an element of randomness, so that the results obtained can be reproduced precisely at a later time.

We begin by using the `sample()` function to split the set of observations into two halves, by selecting a random subset of 196 observations out of the original 392 observations. We refer to these observations as the training set.

```
library(ISLR)
set.seed(1)
train=sample(392,196)
```

We then use the subset option in `lm()` to fit a linear regression using only the observations corresponding to the training set.

```
lm.fit=lm(mpg~horsepower,data=Auto,subset=train)
```

We now use the `predict()` function to estimate the response for all 392 observations, and we calculate the MSE of the 196 observations in the validation set.

```
attach(Auto)
mean((mpg-predict(lm.fit,Auto))[-train]^2)
```

```
## [1] 26.14142
```

Therefore, the estimated test MSE for the linear regression fit is 26.14. We can use the `poly()` function to estimate the test error for the polynomial and cubic regressions.

```
lm.fit2=lm(mpg~poly(horsepower,2),data=Auto,subset=train)
mean((mpg-predict(lm.fit2,Auto))[-train]^2)
```

```
## [1] 19.82259
```

```
lm.fit3=lm(mpg~poly(horsepower,3),data=Auto,subset=train)
mean((mpg-predict(lm.fit3,Auto))[-train]^2)
```

```
## [1] 19.78252
```

These error rates are 19.82 and 19.78, respectively. If we choose a different training set instead, then we will obtain somewhat different errors on the validation set.

```
set.seed(2)
train=sample(392,196)
lm.fit=lm(mpg~horsepower,subset=train)
mean((mpg-predict(lm.fit,Auto))[-train]^2)
```

```
## [1] 23.29559
```

```
lm.fit2=lm(mpg~poly(horsepower,2),data=Auto,subset=train)
mean((mpg-predict(lm.fit2,Auto))[-train]^2)
```

```
## [1] 18.90124
```

```
lm.fit3=lm(mpg~poly(horsepower,3),data=Auto,subset=train)
mean((mpg-predict(lm.fit3,Auto))[-train]^2)
```

```
## [1] 19.2574
```

Using this split of the observations into a training set and a validation set, we find that the validation set error rates for the models with linear, quadratic, and cubic terms are 23.30, 18.90, and 19.26, respectively. These results are consistent with our previous findings: a model that predicts mpg using a quadratic function of `horsepower` performs better than a model that involves only a linear function of `horsepower`, and there is little evidence in favor of a model that uses a cubic function of `horsepower`.

Leave-One-Out Cross-Validation

The LOOCV estimate can be automatically computed for any generalized linear model using the `glm()` and `cv.glm()` functions. In lab5, we used the `glm()` function to perform logistic regression by passing in the `family="binomial"` argument. But if we use `glm()` to fit a model without passing in the family argument, then it performs linear regression, just like the `lm()` function. So for instance,

```
glm.fit=glm(mpg~horsepower,data=Auto)
coef(glm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

```
lm.fit=lm(mpg~horsepower,data=Auto)
coef(lm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

They yield identical linear regression models. We will perform linear regression using the `glm()` function rather than the `lm()` function because the latter can be used together with `cv.glm()`. The `cv.glm()` function is part of the `boot` library.

```
library(boot)
glm.fit=glm(mpg~horsepower,data=Auto)
cv.err=cv.glm(Auto,glm.fit)
cv.err$delta
```

```
## [1] 24.23151 24.23114
```

The two numbers in the delta vector contain the cross-validation results. In this case the numbers are identical (up to two decimal places) and correspond to the LOOCV statistic

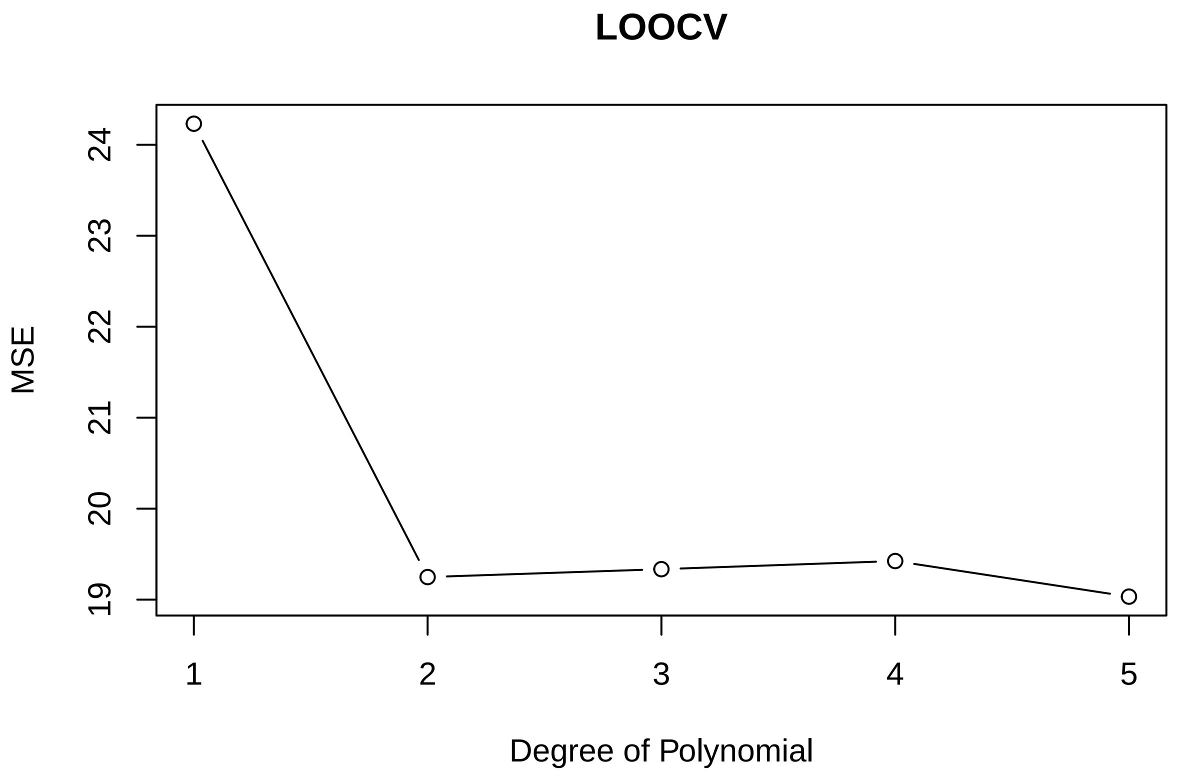
$$CV_n = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

Our cross-validation estimate for the test error is approximately 24.23.

We can repeat this procedure for increasingly complex polynomial fits. To automate the process, we use the `for()` function to initiate a *for loop* which iteratively fits polynomial regressions for polynomials of order $i = 1$ to $i = 5$, computes the associated cross-validation error, and stores it in the ith element of the vector `cv.error`.

```
cv.error=rep(0,5)
for (i in 1:5){
 glm.fit=glm(mpg~poly(horsepower,i),data=Auto)
 cv.error[i]=cv.glm(Auto,glm.fit)$delta[1]
 }
cv.error
```

```
## [1] 24.23151 19.24821 19.33498 19.42443 19.03321
```



We see a sharp drop in the estimated test MSE between the linear and quadratic fits, but then no clear improvement from using higher-order polynomials.

k-Fold Cross-Validation

The `cv.glm()` function can also be used to implement k-fold CV. Below we use $k = 10$, a common choice for k, on the `Auto` data set. We once again set a random seed and initialize a vector in which we will store the CV errors corresponding to the polynomial fits of orders one to ten.

```
set.seed(17)
cv.error.10=rep(0,10)
for (i in 1:10){
 glm.fit=glm(mpg~poly(horsepower,i),data=Auto)
 cv.error.10[i]=cv.glm(Auto,glm.fit,K=10)$delta[1]
 }
cv.error.10
```

```
##  [1] 24.20520 19.18924 19.30662 19.33799 18.87911 19.02103 18.89609
##  [8] 19.71201 18.95140 19.50196
```

We still see little evidence that using cubic or higher-order polynomial terms leads to lower test error than simply using a quadratic fit. We saw in Section 2 that the two numbers associated with `delta` are essentially the same when LOOCV is performed. When we instead perform k-fold CV, then the two numbers associated with `delta` differ slightly. The first is the standard k-fold CV estimate. The second is a bias-corrected version. On this data set, the two estimates are very similar to each other.

Subset Selection Methods for Linear Regression

1. Best Subset Selection

Let's review the best subset selection procedure.
1. Let M_0 denote the null model, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \cdots, p$:
a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
b) Pick the best among these $\binom{p}{k}$ models, and call it M_k. Here, *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $M_0, \cdots, M_p$ using AIC, BIC, C_p, or adjusted R^2.

Here we apply the best subset selection approach to the Hitters data. We wish to predict a baseball player's Salary on the basis of various statistics associated with performance in the previous year.

First of all, we note that the Salary variable is missing for some of the players. The is.na() function can be used to identify the missing observations. It returns a vector of the same length as the input vector, with a TRUE for any elements that are missing, and a FALSE for non-missing elements. The sum() function can then be used to count all of the missing elements.

```
library(ISLR)
sum(is.na(Hitters$Salary))
```

```
## [1] 59
```

Hence we see that Salary is missing for 59 players. The na.omit() function removes all of the rows that have missing values in any variable.

```
Hitters=na.omit(Hitters)
```

The `regsubsets()` function (part of the leaps library) performs best subset selection by identifying the best model that contains a given number of predictors, where best is quantified using RSS. The syntax is the same as for lm(). The summary() command outputs the best set of variables for each model size.

```
library(leaps)
regfit.full=regsubsets(Salary~.,Hitters)
summary(regfit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., Hitters)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns
## 1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "
## 2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "
## 7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "
## 8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"
##          CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 ) "*"  " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 ) "*"  " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 ) "*"  " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 ) "*"  " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 ) "*"  " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 ) "*"  " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 ) " "  " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 ) " "  "*"    " "     "*"       "*"     " "     " "    " "
```

An asterisk indicates that a given variable is included in the corresponding model. For instance, this output indicates that the best two-variable model contains only Hits and CRBI. By default, regsubsets() only reports results up to the best eight-variable model. But the nvmax option can be used in order to return as many variables as are desired. Here we fit up to a 19-variable model.

```r
regfit.full=regsubsets(Salary~.,data=Hitters,nvmax=19)
reg.summary=summary(regfit.full)
```

The summary() function returns R^2, RSS, adjusted R^2, C_p (equivalent to AIC in linear regression), and BIC for models with given numbers of predictors. We can examine these criterions to select the best model.

```r
names(reg.summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

We see that the R^2 statistic increases from 32%, when only one variable is included in the model, to almost 55%, when all variables are included. As expected, the R^2 statistic increases monotonically as more variables are included.

```r
reg.summary$rsq
```

```
##  [1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227
##  [8] 0.5285569 0.5346124 0.5404950 0.5426153 0.5436302 0.5444570 0.5452164
## [15] 0.5454692 0.5457656 0.5459518 0.5460945 0.5461159
```

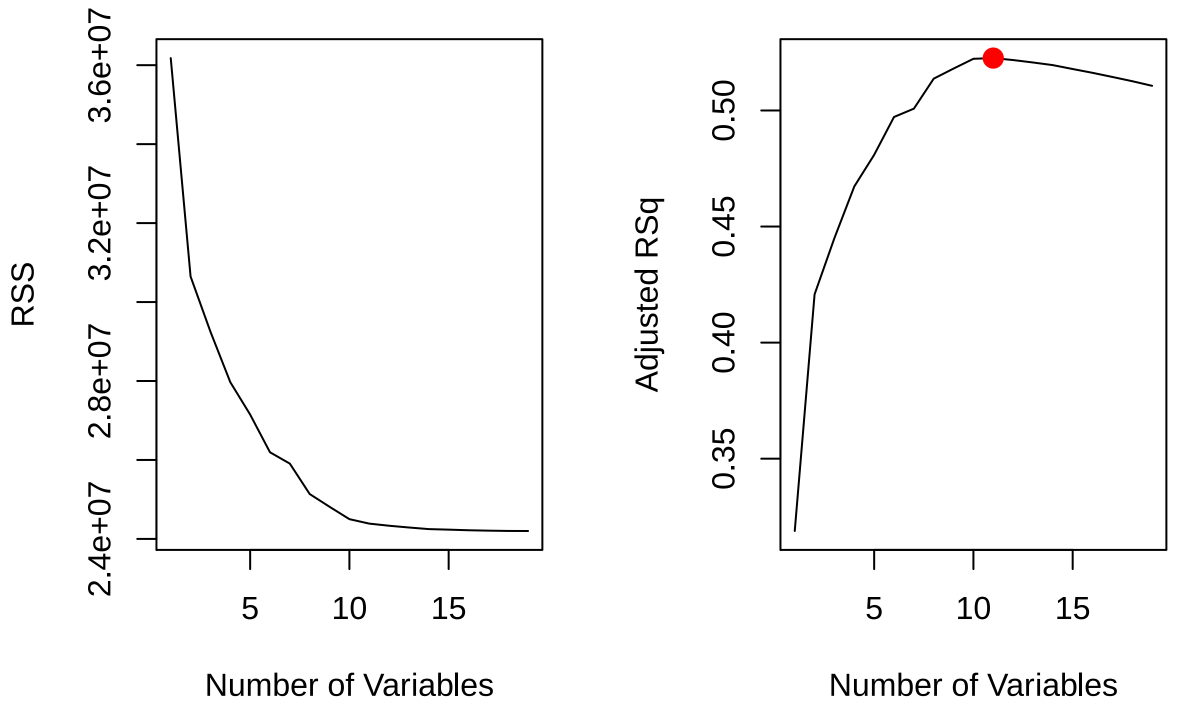
Plotting RSS, adjusted R^2, C_p, and BIC for all of the models at once will help us decide which model to select. Note the type="l" option tells R to connect the plotted points with lines.

```r
par(mfrow=c(1,2))
plot(reg.summary$rss,xlab="Number of Variables",ylab="RSS",type="l")
plot(reg.summary$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="l")
```

```r
which.max(reg.summary$adjr2)
```

```
## [1] 11
```

```r
points(11,reg.summary$adjr2[11], col="red",cex=2,pch=20)
```



We can see that a 11-variable model was chosen as the best model according to adjusted R^2. The points() command works like the plot() command, except that it puts points on a plot that has already been created, instead of creating a new plot. The which.max() function can be used to identify the location of the maximum point of a vector. We will now plot a red dot to indicate the model with the largest adjusted R^2 statistic.(i.e. model chosen by adjusted R^2)

In a similar fashion we can plot the C_p and BIC statistics, and indicate the models with the smallest statistic using which.min(). We choose a 10-variable model based on C_p and 6-variable model with BIC. Thus, we can see that the best subset selected by each criterion differs.

```r
par(mfrow=c(1,2))
plot(reg.summary$cp,xlab="Number of Variables",ylab="Cp",type='l')
which.min(reg.summary$cp)
```

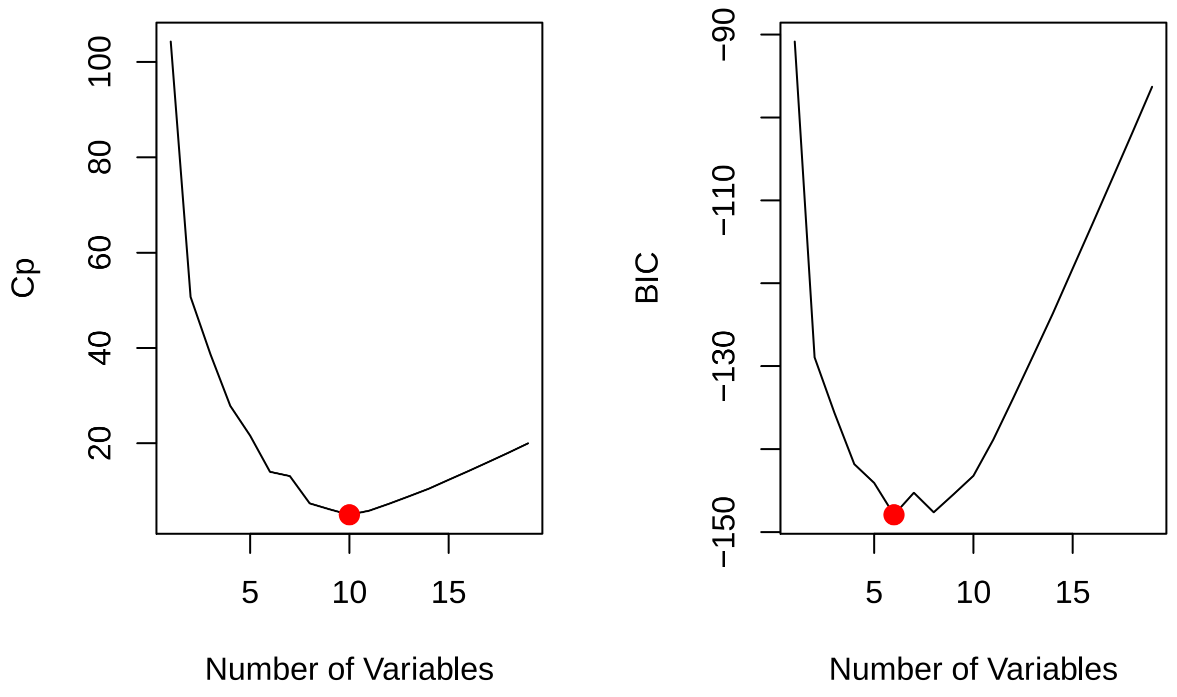
```
## [1] 10
```

```r
points(10,reg.summary$cp[10],col="red",cex=2,pch=20)

plot(reg.summary$bic,xlab="Number of Variables",ylab="BIC",type='l')
which.min(reg.summary$bic)
```

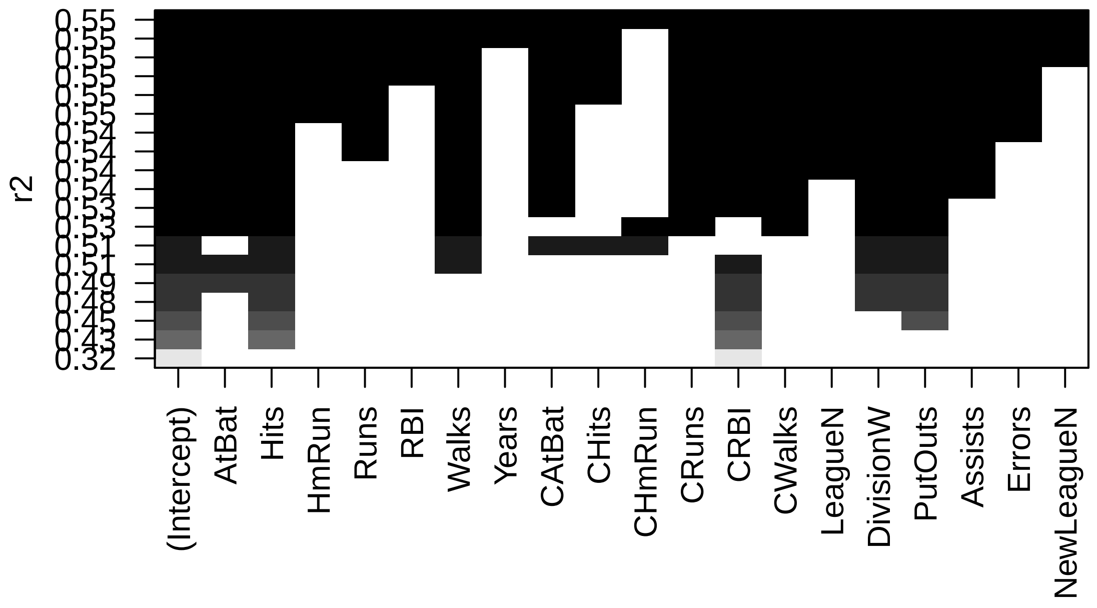
```
## [1] 6
```

```r
points(6,reg.summary$bic[6],col="red",cex=2,pch=20)
```

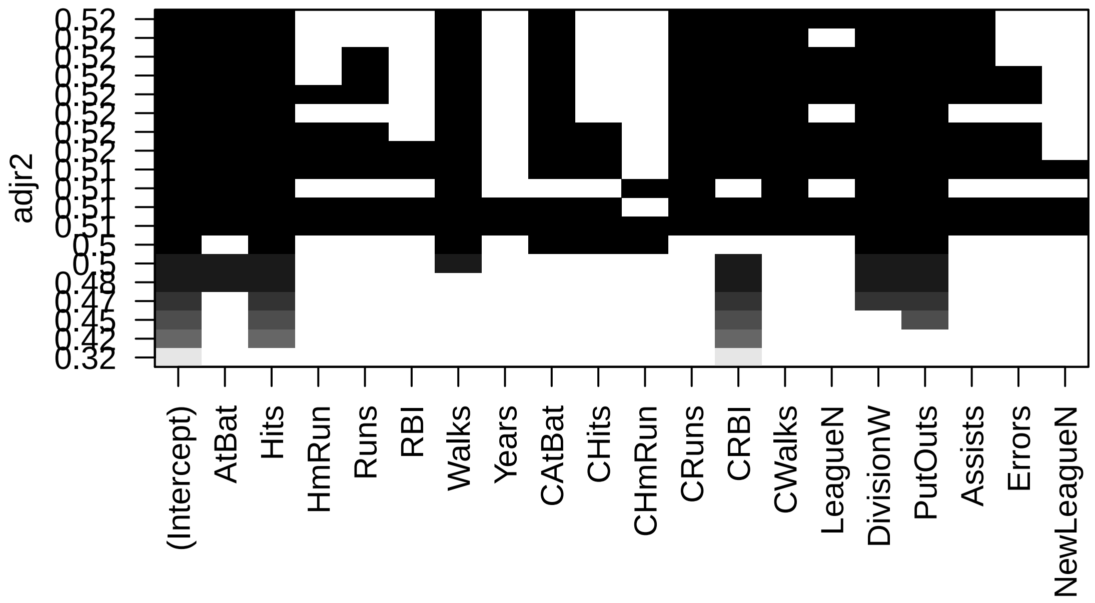


The `regsubsets()` function has a built-in `plot()` command which can be used to display the selected variables for the best model with a given number of predictors, ranked according to the BIC, C_p, adjusted R^2, or AIC. To find out more about this function, type ?plot.regsubsets.

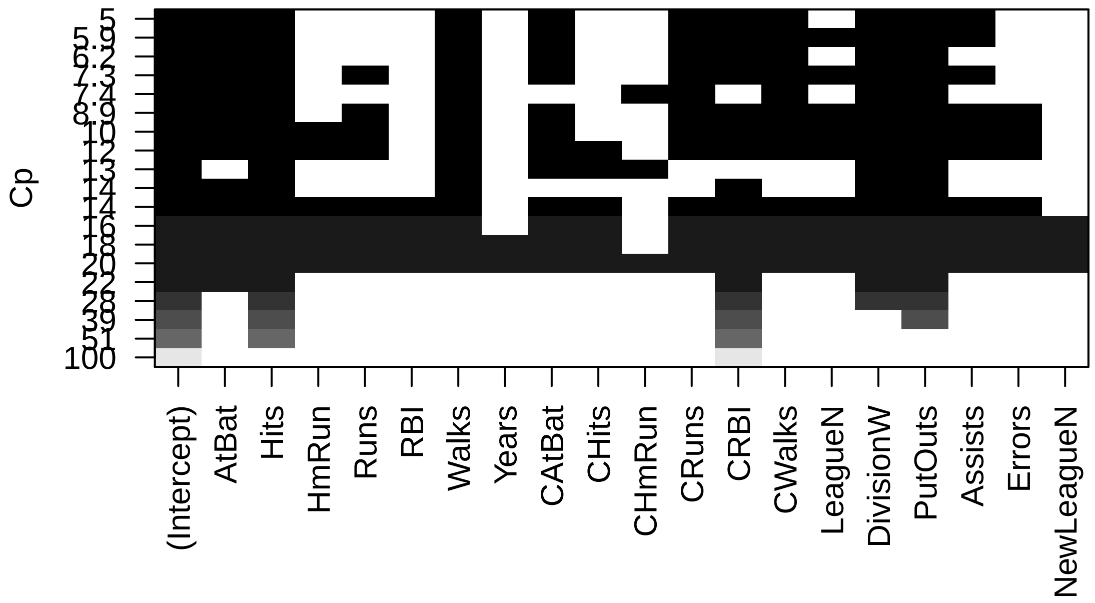
```r
plot(regfit.full,scale="r2")
```



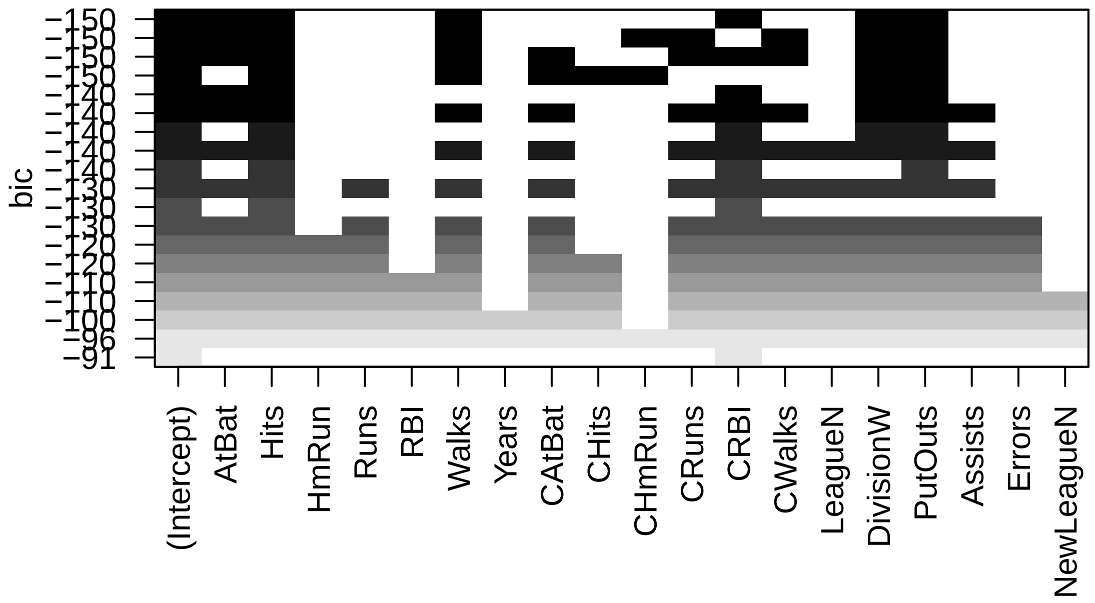
```r
plot(regfit.full,scale="adjr2")
```



```r
plot(regfit.full,scale="Cp")
```



```r
plot(regfit.full,scale="bic")
```



The top row of each plot contains a black square for each variable selected according to the optimal model associated with that statistic. For instance, we see that several models share a BIC close to -150. However, the model with the lowest BIC is the six-variable model that contains only AtBat, Hits, Walks, CRBI, DivisionW, and PutOuts. We can use the `coef()` function to see the coefficient estimates associated with this model.

```
coef(regfit.full,6)
```

```
##  (Intercept)        AtBat         Hits        Walks         CRBI
##   91.5117981   -1.8685892    7.6043976    3.6976468    0.6430169
##    DivisionW      PutOuts
## -122.9515338    0.2643076
```

2. Forward and Backward Stepwise Selection

For computational reasons, best subset selection cannot be applied with very large p. In that case, stepwise methods, which explore a far more restricted set of models, are attractive alternatives to best subset selection.

We can also use the `regsubsets()` function to perform forward stepwise or backward stepwise selection, using the argument method="forward" or method="backward".

```
regfit.fwd=regsubsets(Salary~.,data=Hitters,nvmax=19,method="forward")
summary(regfit.fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "forward")
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: forward
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns
## 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 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"
##           CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 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 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

```r
regfit.bwd=regsubsets(Salary~.,data=Hitters,nvmax=19,method="backward")
summary(regfit.bwd)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters, nvmax = 19, method = "backward")
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: backward
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns
## 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 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"
##           CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 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 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

For instance, we see that using forward stepwise selection, the best one-variable model contains only CRBI, and the best two-variable model additionally includes Hits. For this data, the best one-variable through six variable models are each identical for best subset and forward selection. However, the best seven-variable models identified by forward stepwise selection, backward stepwise selection, and best subset selection are different.

```r
coef(regfit.full,7)
```

```
##   (Intercept)          Hits         Walks        CAtBat         CHits
##    79.4509472     1.2833513     3.2274264    -0.3752350     1.4957073
##        CHmRun     DivisionW       PutOuts
##     1.4420538  -129.9866432     0.2366813
```

```r
coef(regfit.fwd,7)
```

```
##   (Intercept)         AtBat          Hits         Walks          CRBI
##   109.7873062    -1.9588851     7.4498772     4.9131401     0.8537622
##        CWalks     DivisionW       PutOuts
##    -0.3053070  -127.1223928     0.2533404
```

```r
coef(regfit.bwd,7)
```

```
##   (Intercept)         AtBat          Hits         Walks         CRuns
##   105.6487488    -1.9762838     6.7574914     6.0558691     1.1293095
##        CWalks     DivisionW       PutOuts
##    -0.7163346  -116.1692169     0.3028847
```

3. Choosing Models Using the Validation Set Approach and Cross-Validation

Recall that we wish to choose a model with low test error, not a model with low training error. We just saw that it is possible to choose among a set of models of different sizes using Cp, BIC, and adjusted R^2, which indirectly estimate test error by making an adjustment to the training error to account for the bias due to overfitting. Now, we will directly estimate the test error using either a validation set approach or a cross-validation approach.

In order for these approaches to yield accurate estimates of the test error, we must use only the training observations to perform all aspects of model-fitting -including variable selection. Therefore, the determination of which model of a given size is best must be made using only the training observations. This point is subtle but important. If the full data set is used to perform the best subset selection step, the validation set errors and cross-validation errors that we obtain will not be accurate estimates of the test error.

Suppose we have a sequence of models M_k indexed by model size $k = 0, 1, 2, ..., p$. For both validation set approach and cross-validation approach, we estimate the model size k for which the estimated test error(validation error or cross-validation error) is smallest. Once k is selected, we return a model $M_{\hat{k}}$

Validation set approach

In order to use the validation set approach, we begin by splitting the observations into a training set and a test set. We do this by creating a random vector, train, of elements equal to TRUE if the corresponding observation is in the training set, and FALSE otherwise. The vector test has a TRUE if the observation is in the test set, and a FALSE otherwise. Note the ! in the command to create test causes TRUEs to be switched to FALSEs and vice versa. We also set a random seed so that the user will obtain the same training set/test set split.

```
set.seed(1)
train=sample(c(TRUE,FALSE), nrow(Hitters),rep=TRUE)
test=(!train)
```

Now, we apply regsubsets() to the training set in order to perform best subset selection.

```
regfit.best=regsubsets(Salary~.,data=Hitters[train,],nvmax=19)
```

Notice that we subset the Hitters data frame directly in the call in order to access only the training subset of the data, using the expression Hitters[train,]. We now compute the validation set error for the best model of each model size. We first make a model matrix from the test data.

```
test.mat=model.matrix(Salary~.,data=Hitters[test,])
```

The model.matrix() function is used in many regression packages for building an “X” matrix from data. Now we run a loop, and for each size i, we extract the coefficients from regfit.best for the best model of that size, multiply them into the appropriate columns of the test model matrix to form the predictions, and compute the test MSE.

```
val.errors=rep(NA,19)
for(i in 1:19){
   coefi=coef(regfit.best,id=i)
   pred=test.mat[,names(coefi)]%*%coefi
   val.errors[i]=mean((Hitters$Salary[test]-pred)^2)
}
```

We find that the best model is the one that contains ten variables.

```
val.errors
```

```
##  [1] 220968.0 169157.1 178518.2 163426.1 168418.1 171270.6 162377.1
##  [8] 157909.3 154055.7 148162.1 151156.4 151742.5 152214.5 157358.7
## [15] 158541.4 158743.3 159972.7 159859.8 160105.6
```

```
which.min(val.errors)
```

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

```
coef(regfit.best,10)
```

```
## (Intercept)       AtBat        Hits       Walks      CAtBat       CHits
## -80.2751499  -1.4683816   7.1625314   3.6430345  -0.1855698   1.1053238
##      CHmRun      CWalks     LeagueN   DivisionW     PutOuts
##   1.3844863  -0.7483170  84.5576103 -53.0289658   0.2381662
```

This was a little tedious, partly because there is no predict() method for regsubsets(). Since we will be using this function again, we can capture our steps above and write our own predict method.

```
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
  }
```

Our function pretty much mimics what we did above. The only complex part is how we extracted the formula used in the call to regsubsets(). We demonstrate how we use this function below, when we do cross-validation.

Cross validation approach

We now try to choose among the models of different sizes using crossvalidation. This approach is somewhat involved, as we must perform best subset selection within each of the k training sets. Despite this, we see that with its clever subsetting syntax, R makes this job quite easy. First, we create a vector that allocates each observation to one of $k = 10$ folds, and we create a matrix in which we will store the results. By running `rep()` function on vector 1:k, we try to make a vector with equal number of 1's, 2's, $\cdots$. By running `sample()`, we shuffle the fold-assignment for each data point.

```
k=10
set.seed(1)
folds=sample(rep(1:k,length=nrow(Hitters)))
table(folds)
```

```
## folds
##  1  2  3  4  5  6  7  8  9 10
## 27 27 27 26 26 26 26 26 26 26
```

```
cv.errors=matrix(NA,k,19, dimnames=list(NULL, paste(1:19)))
```

Now we write a for loop that performs cross-validation. In the jth fold, the elements of folds that equal j are in the test set, and the remainder are in the training set. We make our predictions for each model size (using our new predict() method), compute the test errors on the appropriate subset, and store them in the appropriate slot in the matrix cv.errors.

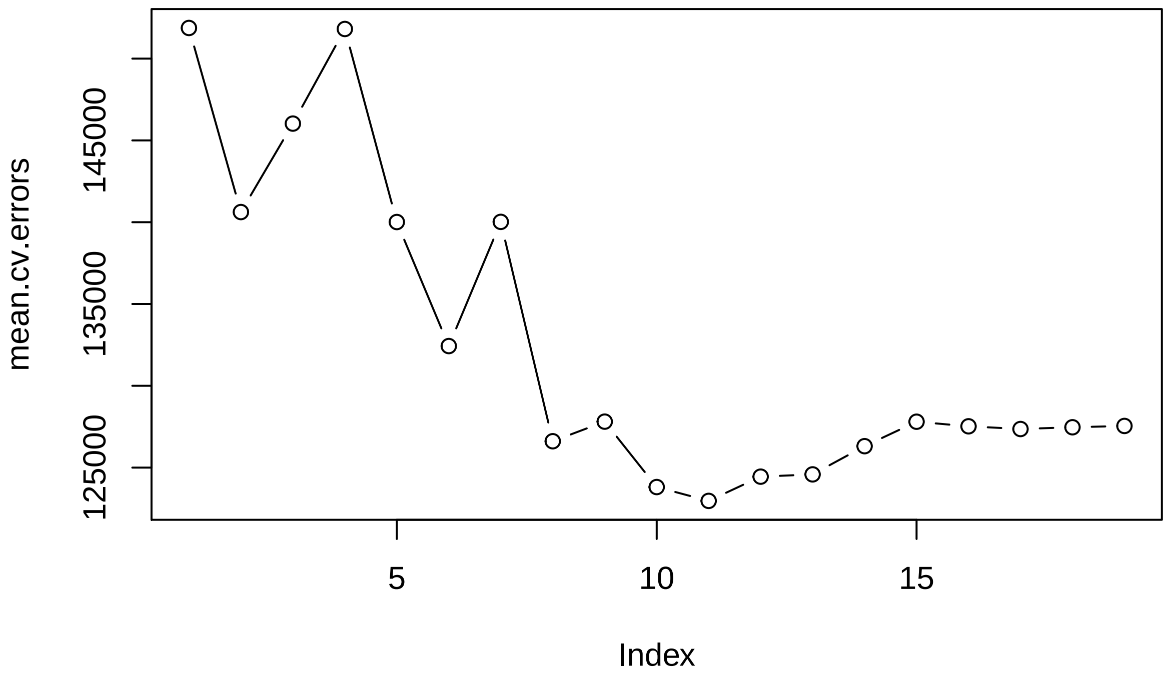
```
for(j in 1:k){
  best.fit=regsubsets(Salary~.,data=Hitters[folds!=j,],nvmax=19)
  for(i in 1:19){
    pred=predict(best.fit,Hitters[folds==j,],id=i)
    cv.errors[j,i]=mean( (Hitters$Salary[folds==j]-pred)^2)
    }
  }
```

This has given us a 10×19 matrix, of which the (i, j)th element corresponds to the test MSE for the ith cross-validation fold for the best j-variable model. We use the apply() function to average over the columns of this matrix in order to obtain a vector for which the jth element is the crossvalidation error for the j-variable model.

```
mean.cv.errors=apply(cv.errors,2,mean)
mean.cv.errors
```

```
##         1         2         3         4         5         6         7         8
## 151871.5 140620.2 146026.6 151809.5 140011.0 132429.2 140020.6 126612.3
##        9       10       11       12       13       14       15       16
## 127810.3 123811.2 122966.2 124447.9 124585.2 126310.6 127805.1 127524.9
##       17       18       19
## 127359.1 127466.2 127546.2
```

```r
par(mfrow=c(1,1))
plot(mean.cv.errors,type='b')
```



We see that cross-validation selects an 11-variable model.