Cross-Validation (Chapter 5)

Alongside model selection, which has been our focus thus far, we must conduct model assessment. For regression a measure of the success of a model is its mean squared error (MSE), the average squared residual on a test data point; for classification, it is the misclassification rate.

An approach we have encountered is to set aside certain training observations, not employing them during the fit stage, but using the resulting method on them and using their residuals to estimate the test error. Several approaches are common in this guise.

1. **Validation set approach.** In this method, (labeled) data is split into two similarly-sized parts, with one part used for training the model, and the other is used to calculate a training error. The split may be used to produce an MSE “curve”, with points \((t, \text{MSE}(t))\) arising from a parameter \(t\) which dictates the complexity of the model, and corresponding MSE estimate (in the regression case). A single curve provides an estimate of where overfitting begins. It is more revealing, however, if upon generating one curve, one reshuffles the labeled data uses the resulting training and held-back sets to generate another curve. The multiple curves obtained through iterations of this process help reveal the sort of variability in \(\text{MSE}(t)\) at a given \(t\) and whether conclusions about optimal complexity in the model drawn from just one MSE curve carry over to others.

We give an example involving the Auto data frame.

```r
set.seed(2)
train = sample(1:392, 196)  # randomly take n/2 observations to train on

lm.fit = lm(mpg ~ horsepower, data=Auto, subset=train)
attach(Auto)
mean((mpg - predict(lm.fit, Auto))[-train]^2)
% [1] 23.29559

Next, we do quadratic and cubic fits:

```r
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
```

The three estimates of test error are part of a single MSE curve, and confirm earlier findings that our best fit comes at around a degree-2 polynomial. If we change the seed and repeat these estimates a number of times we generate additional MSE curves, allowing ourselves a sense as to whether this is a property of the specific split, or carry through to most (all?) splits of data into training and validation sets.

2. **k-fold CV.** Here, one divides the labeled data into $k$ evenly-sized groups. One of these groups serves as the validation set, the source of an estimate of test MSE, after the rest have been used to train the model function. Each group is rotated through the role of validation group, producing an estimate of MSE, and the final estimate is the mean of these MSEs.

When the number of *folds* is $k=n$, we get a special case called **leave-one-out-cross-validation** (LOOCV), which has a surprising formula for the estimate of the test error

$$
CV(n) = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{y_i - \hat{y}_i}{1 - h_i} \right)^2,
$$

when the fit is least squares linear or polynomial regression. Here, $h_i$ represents the leverage of data point $i$. Nonetheless, taking $k=5$ or $k=10$ generally strikes a good balance between bias and variance.

We give an example employing the `glm()` command in concert with a function `cv.glm()` provided in the `boot` package.

```r
% Loading required package: boot
%
% Attaching package: 'boot'
%
% The following object is masked from 'package:lattice':
%
% melanoma

glm.fit = glm(mpg ~ horsepower, data=Auto)
cv.err = cv.glm(Auto, glm.fit)

names(cv.err)  # see the names of various objects returned in cv.err

% [1] "call" "K" "delta" "seed"

cv.err$delta

% [1] 24.23151 24.23114
```

The value in the `delta` component of `cv.err` has two components, but they match, so we really only need `cv.err$delta[1]`.

It is instructive to view the help information on the `cv.glm()` command.
Reading through the Arguments section of this documentation, you realize the result above has used the LOOCV approach, but is easily modified to \( k \)-fold CV for various choices of \( k \). Before doing so we generate one MSE curve via LOOCV:

```r
cv.error = rep(0, 5)  # currently an empty vector
for (i in 1:5) {
  glm.fit = glm(mpg ~ poly(horsepower, i), data=Auto)
  cv.error[i] = cv.glm(Auto, glm.fit)$delta[1]
}
plot(cv.error, pch=20, cex=.6, ylim=c(15,25), ylab="est. test MSE",
     xlab="degree of fit")
lines(cv.error, lwd=1)
```

Now let’s generate an MSE curve using \( k \)-fold CV with \( k = 10 \).

```r
cv.error.kfold = rep(0, 10)  # currently an empty vector
for (i in 1:10) {
  glm.fit = glm(mpg ~ poly(horsepower, i), data=Auto)
  cv.error.kfold[i] = cv.glm(Auto, glm.fit, K=10)$delta[1]
}
plot(cv.error.kfold, pch=20, cex=.6, ylim=c(15,25), ylab="est. test MSE",
     xlab="degree of fit")
lines(cv.error.kfold, lwd=1)
```
These same ideas carry over to the classification problem. We investigate their use in this setting in the exercises.

**Best Subset Selection (Chapter 6)**

In the lecture, we observe the process of choosing the best subset of predictors in linear regression. Here are relevant steps using R.

```r
Credit = read.csv("http://www-bcf.usc.edu/~gareth/ISL/Credit.csv",row.names=1)
names(Credit)
% [1] "Income" "Limit" "Rating" "Cards" "Age" "Education" "Gender"
% [8] "Student" "Married" "Ethnicity" "Balance"
```

We check a few measures on whether there is missing data.

```r
dim(Credit)
% [1] 400 11
dim(na.omit(Credit))  # same as above
% [1] 400 11
sum(is.na(Credit$Balance))
% [1] 0
```
We use the `regsubsets()` command, part of the `leaps` package, to learn the best (as measured by RSS) model at each model size, from 1 predictor to all (11) predictors. Note that, aside from balance, there are 10 columns or potential predictors, three of which are categorical. Two of them, Gender and student, are binary, and therefore one dummy variable is generated—and stands in—for each during regression. The 3rd categorical variable is Ethnicity, which has 3 values, so it generates 2 dummy variables.

```r
require(leaps)

% Loading required package: leaps

regfit.full = regsubsets(Balance ~ ., data=Credit, nvmax=11)
reg.summary = summary(regfit.full)
reg.summary

% Subset selection object
% Call: regsubsets.formula(Balance ~ ., data = Credit, nvmax = 11)
% 11 Variables (and intercept)
% Forced in Forced out
% Income FALSE FALSE
% Limit FALSE FALSE
% Rating FALSE FALSE
% Cards FALSE FALSE
% Age FALSE FALSE
% Education FALSE FALSE
% GenderFemale FALSE FALSE
% StudentYes FALSE FALSE
% MarriedYes FALSE FALSE
% EthnicityAsian FALSE FALSE
% EthnicityCaucasian FALSE FALSE
% 1 subsets of each size up to 11
% Selection Algorithm: exhaustive

<table>
<thead>
<tr>
<th></th>
<th>Income</th>
<th>Limit</th>
<th>Rating</th>
<th>Cards</th>
<th>Age</th>
<th>Education</th>
<th>GenderFemale</th>
<th>StudentYes</th>
<th>MarriedYes</th>
<th>EthnicityAsian</th>
<th>EthnicityCaucasian</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 (1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Without the *switch* \texttt{nvmax=11}, the function would not have progressed past a model with 8 predictors. I have set it to 11 because I knew we had at most 11 predictors in this dataset, but if you don’t wish to make a count ahead of time, you can set this number higher, knowing it will stop ahead of your setting if it has run out of predictors.

There are a number of things returned by the \texttt{summary()} command above. We can see a full list, then query some of the contents in the usual way employing the $ notation:

\begin{verbatim}
names(reg.summary)
\end{verbatim}

\begin{verbatim}
reg.summary$rsq
\end{verbatim}

\begin{verbatim}
reg.summary$adjr2
\end{verbatim}

\begin{verbatim}
reg.summary$cp
\end{verbatim}

We can produce a plot of RSS vs. number of predictors, such as that in the lecture.
Based on RSS, we might choose to use the model with 3 or more features. There are some tools/plots tailored to work hand-in-hand with the `regsubsets()` function, whose purpose is to assist one in choosing predictor variables. These plots are accessed by applying the `plot()` function to the information stored in `regfit.full`. Below we give one example, but see p. 246 for the syntax for others.

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

A row of blocks contains a black block when a predictor variable is included. Quite a few models produce a BIC value near -1200, including the one using all available predictors. The best one
(appearing at the top) actually has the lowest BIC, and comes from the model with 4 (in addition to the intercept) predictors: Income, Limit, Cards, and Student. To see the coefficients of the linear regression model corresponding to this collection, we use the `coef()` function on `regfit.full`:

```r
coef(regfit.full, 4)
```

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>Income</td>
<td>Limit</td>
<td>Cards</td>
</tr>
<tr>
<td>-499.7272117</td>
<td>-7.8392288</td>
<td>0.2666445</td>
<td>23.1753794</td>
</tr>
<tr>
<td>StudentYes</td>
<td>429.6064203</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Forward/Backward Selection**

With many (say, $p > 15$?) predictors, the process of finding the best model out of every one of the $2^p$ possible ones becomes computationally intractable. We have seen how forward/backward selection methods cut down on the number of possible models (down to a number on the order of $p^2$). The modifications to function calls to carry out one of these processes are minimal, in that you need to add another switch: `method = "forward"`, for instance.

```r
regfit.fwd = regsubsets(Balance ~ ., data=Credit, nvmax=13, method="forward")
fwd.summary = summary(regfit.fwd)
plot(fwd.summary$cp, pch=20)
fwd.summary$cp
coef(regfit.fwd, 6)
```