Cross-validation and Model Selection

Lab: Cross-Validation

In this lab, we explore the resampling techniques covered in this chapter. Some of the commands in this lab may take a while to run on your computer.

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 for `R` 's random number generator, so that the reader of this book 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(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.0.5
```

```
set.seed(1)
train <- sample(392, 196)
```

(Here we use a shortcut in the sample command; see `?sample` for details.) 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 use the `mean()` function to calculate the MSE of the 196 observations in the validation set. Note that the `-train` index below selects only the observations that are not in the training set.

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

```
## [1] 23.26601
```

Therefore, the estimated test MSE for the linear regression fit is 23.27. We can use the `poly()` function to estimate the test error for the quadratic and cubic regressions.

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

```
## [1] 18.71646
```

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

```
## [1] 18.79401
```

These error rates are 18.72 and 18.79, 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] 25.72651
```

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

```
## [1] 20.43036
```

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

```
## [1] 20.38533
```

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 25.73, 20.43, and 20.39, 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 the lab for Chapter 4, 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
```

and

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

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

yield identical linear regression models. In this lab, we will perform linear regression using the `glm()` function rather than the `lm()` function because the former 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 `cv.glm()` function produces a list with several components. 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 given in (5.1). Below, we discuss a situation in which the two numbers differ. 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 which iteratively fits polynomial regressions for polynomials of order $i = 1$ to $i = 10$, computes the associated cross-validation error, and stores it in the ith element of the vector `cv.error` . We begin by initializing the vector.

```
cv.error <- rep(0, 10)
for (i in 1:10) {
  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 18.97864 18.83305 18.96115
##  [9] 19.06863 19.49093
```

As in Figure 5.4, 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] 24.27207 19.26909 19.34805 19.29496 19.03198 18.89781 19.12061 19.14666
##  [9] 18.87013 20.95520
```

Notice that the computation time is shorter than that of LOOCV. (In principle, the computation time for LOOCV for a least squares linear model should be faster than for k-fold CV, due to the availability of the formula (5.2) for LOOCV; however, unfortunately the `cv.glm()` function does not make use of this formula.) 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 5.3.2 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, as in (5.3). The second is a bias-corrected version. On this data set, the two estimates are very similar to each other.