Resampling Methods

Simone Collier

Getting Started

Start by loading the packages that have the data we need. If you need to install the packages first then run `install.packages("PACKAGENAME")` in your console before running the code chunk.

```
library(ISLR2)
library(boot)
library(dslabs)
```

We will be making use of the `Auto` data set in the `ISLR` package. It contains information about the gas milage, horsepower, and other variables for 392 vehicles.

```
attach(Auto)
```

The Validation Set Approach

We will try out the validation set approach for estimating the test error rates that various linear models fit on the `Auto` data set. We set a seed so that the results of our analysis are reproducible.

```
set.seed(1)
```

We can use the function `sample()` to split the observations into two halves for the training and testing sets. The `train` vector contains the indices of the `Auto` dataset that will be used as the training data.

```
# split the 392 observations in half so 196 observations in the training set
train <- sample(392, 196)
```

How do we fit a linear regression to the training set with `horsepower` as the predictor and `mpg` as the response?

Now we can predict the response for the complete `Auto` data set using our fitting linear model.

```
lm.pred <- predict(lm.fit, Auto)
```

Given our predictions we can compute the MSE for the test set as follows

```
# the index -train indicates all the indices that are not in train
mean((mpg - lm.pred)[-train]^2)
```

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

So our estimated test MSE for the linear regression is 23.37.

What would happen if we used a different set of observations for our training set? ***Repeat the above process using a different seed (i.e. `set.seed(2)`) and report the differences.***

Leave-One-Out Cross-Validation

We will use the function `glm()` to perform linear regression instead of `lm()` since it is compatable with the function `cv.glm()` used to compute the LOOCV estimate for generalized lienar models. `glm()` performs linear refression automatically when there is no argument for `family`.

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

The `cv.glm()` function is part of the `boot` library. The output of the function has several components. Use `?cv.glm` if you are curious to read more about them. The cross-validation results can be found by indexing using `delta`.

```
cv.error <- cv.glm(Auto, glm.fit)
cv.error$delta
```

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

The first element of the delta vector is the LOOCV estimate and the second is the adjusted estimate. We will only ever use the first, so ignore the second for the purpose of this course. So, the LOOCV estimate the test error for our linear model is 24.23.

We can fit our data using polynomials to see if we can get a better fit. The function `poly()` can be used in conjunction with `glm()` to do this. Replacing `horsepower` with `poly(horsepower, 1)` in the `glm()` we have above will yeild the same result since we are fitting a polynomial of degree 1 (linear).

```
glm.fit <- glm(mpg ~ poly(horsepower, 1), data = Auto)
cv.error <- cv.glm(Auto, glm.fit)
cv.error$delta[1]
```

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

Write a for loop that fits the `Auto` data using polynomials up to degree 10. Compute the LOOCV test error rate for each of them. Which degree polynomial fits the data the best according to the test error rate and the bias-variance trade-off?

k-Fold Cross-Validation

The `cv.glm()` can be used to perform k-fold CV using the input `K = ...` in the function. We will choose `K = 5`. We can set a seed and then write a for loop that will fit the data with up to a degree 10 polynomial.

```
set.seed(4)
# make a vector of length 10 to be filled with the test errors
kcv.errors <- numeric(10)
#for loop with 1:10 degreee polynomials
for (i in 1:10) {
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  kcv.errors[i] <- cv.glm(Auto, glm.fit, K = 5)$delta[1]
}
kcv.errors
```

```
##  [1] 24.19050 19.06920 19.20444 19.56072 18.81189 18.78161 19.22280 18.39967
##  [9] 18.77825 18.98969
```

Notice that the test error rate decline significantly between the linear and quadratic fits. With the higher order polynomials (degree 3 and up) we see little or no improvement which indicates there is no evidence that fitting with a higher order polynomial will give better results that a quadratic fit.

Repeat this process using `K = 10` and report the results.

The Bootstrap

The data set `heights` from the `dslabs` R package contains the self reported heights of 1050 individuals as well as their sex. Suppose that this data was sampled from individuals in Toronto and we want to know the mean height for the Toronto population. We can use the bootstrap method we decribed before to assign some conficence level on our estimate of the population mean from this sample.

We first need to create a function that will take in our data set and desired indices and then comoute the mean of the data on the indices.

```
mean.func <- function(data, index) {
  mean(data$height[index])
}
```

To preform one iteration of the boot strap method we sample 1050 observations for the 1050 heights while allowing observations to be repeated with `replace = TRUE`. We input this as in the index in our `mean.func()`. Make sure to set a seed for reproducibiity.

```
set.seed(5)
mean.func(heights, sample(1050, 1050, replace = TRUE))
```

```
## [1] 68.45896
```

Now we have to repeat this process many times in order to find many estimates to the mean and then compute the standard error. The funciton `boot()` from the `boot` library does this for us! All we need to do is give the data, the function used to compute our statistic and how many iterations of the bootstrap we would like to perform. In this case, let's choose 1000.

```
boot(heights, mean.func, R = 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = heights, statistic = mean.func, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 68.32301 0.001073793   0.1232363
```

The output from `boot()` tells us the desired statistic (mean) for our complete data set: `original = 68.32` as well as bootstap estimate of the standard error: `std. error = 0.12408`.

The bootstrap can also be used to estimate the variability of the coefficient estimates of statistical learning methods. We will use the bootstrap method to find the standard errors of the estimates of β_0 and β_1 for the linear regression model fit on the `Auto` data set.

Once again, we need to create a function `coef.func` which fits the reression coefficients using an input for the data and the indices of the data that are to be used in the fitting.

```
coef.func <- function(data, index) {
  coef(lm(mpg ~ horsepower, data = data, subset = index))
}
```

We can use this function inside `boot()` to compute bootstap estimates of the intercept and slope terms and their standard errors.

```
set.seed(1)
boot(Auto, coef.func, 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Auto, statistic = coef.func, R = 1000)
##
##
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 39.9358610  0.0553942585 0.843931305
## t2* -0.1578447 -0.0006285291 0.007367396
```

This gives the bootstrap estimate of the standard error for β_0 as 0.844 and β_1 as 0.007.

These exercises were adapted from : James, Gareth, et al. An Introduction to Statistical Learning: with Applications in R, 2nd ed., Springer, 2021.