IMT 573 Lab: Resampling Methods

Miloni Desai

November 21st, 2019

Collaborators

Objectives

In this lab, we explore resampling methods using the `Auto` dataset. This material is taken from James et al. (2013).

Before beginning this assignment, please ensure you have access to R and RStudio; this can be on your own personal computer or on the IMT 573 R Studio Server.

1. Download the `lab8_resampling.rmd` file from Canvas or save a copy to your local directory on RStudio Server. Open `lab8_resampling.rmd` in RStudio and supply your solutions to the assignment by editing `lab8_resampling.rmd`.
2. Replace the "Insert Your Name Here" text in the `author:` field with your own full name. Any collaborators must be listed on the top of your assignment.
3. Be sure to include well-documented (e.g. commented) code chucks, figures, and clearly written text chunk explanations as necessary. Any figures should be clearly labeled and appropriately referenced within the text. Be sure that each visualization adds value to your written explanation; avoid redundancy – you do no need four different visualizations of the same pattern.
4. Collaboration on problem sets is fun and useful, and we encourage it, but each student must turn in an individual write-up in their own words as well as code/work that is their own. Regardless of whether you work with others, what you turn in must be your own work; this includes code and interpretation of results. The names of all collaborators must be listed on each assignment. Do not copy-and-paste from other students' responses or code.
5. All materials and resources that you use (with the exception of lecture slides) must be appropriately referenced within your assignment.
6. When you have completed the assignment and have **checked** that your code both runs in the Console and knits correctly when you click `Knit PDF`, rename the knitted PDF file to `lab8_YourLastName_YourFirstName.pdf`, and submit the PDF file on Canvas.

Setup

In this lab you will need, at minimum, the following R packages.

```r
# Load standard libraries
library(tidyverse)
# Load the ISLR library
library(ISLR)
```

The Validation Set Approach

Let's start by using the `sample()` function to split (i.e. define) the set of observations into two sets, by selecting a random subset of 196 observations out of the original 392 observations. Note the training set here is 1/2 of the original dataset.

```r
#View(Auto)
# Set a random seed so that results can be reproduced
set.seed(1)

# Define the training set
train <- sample(392,196)
```

After defining the training data we can use it to fit a linear regression model. The `subset` option in `lm()` is helpful here.

```r
# Fit a linear regression model using only the training set
lm.fit <- lm (mpg ~ horsepower, data = Auto, subset = train)
```

We use the `predict()` function to estimate the response for all observations. The `mean()` function is used to calculate the MSE of the validation set. Note that the `-train` index below selects only the observations that are not in the training set.

```r
# Calculate the MSE of the validation set
mse <- mean((Auto$mpg-predict(lm.fit, Auto))[-train]^2)
mse
```

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

Try choosing a different training set. Are the results different?

```r
# New random seed
set.seed(2)



# Define new training set
train.new <- sample(392,196)

# Fit a linear regression model using only the training set
lm.fit.new <- lm(mpg ~ horsepower, data = Auto, subset = train.new)

# Calculate the MSE of the validation set
mse.new <- mean((Auto$mpg-predict(lm.fit.new, Auto))[-train]^2)

mse==mse.new
```

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

LOOCV

The LOOCV estimate can be automatically computed for any generalized linear model using the `glm()` and `cv.glm()` functions. If we use `glm()` to fit a model without passing in the family argument, then it performs linear regression, just like the `lm()` function.

```r
# Fit the model using the glm function
glm.fit <- glm(mpg ~ horsepower, data = Auto)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = mpg ~ horsepower, data = Auto)
##
```

```
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -13.5710   -3.2592   -0.3435    2.7630   16.9240
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 39.935861   0.717499   55.66   <2e-16 ***
## horsepower  -0.157845   0.006446  -24.49   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 24.06645)
##
##     Null deviance: 23819.0  on 391  degrees of freedom
## Residual deviance:  9385.9  on 390  degrees of freedom
## AIC: 2363.3
##
## Number of Fisher Scoring iterations: 2
```

```
# default behavior is the same as the liner model
lm.fit.all <- lm(mpg ~ horsepower, data = Auto)
summary(lm.fit.all)
```

```
##
## Call:
## lm(formula = mpg ~ horsepower, data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5710  -3.2592  -0.3435   2.7630  16.9240
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 39.935861   0.717499   55.66   <2e-16 ***
## horsepower  -0.157845   0.006446  -24.49   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.906 on 390 degrees of freedom
## Multiple R-squared:  0.6059, Adjusted R-squared:  0.6049
## F-statistic: 599.7 on 1 and 390 DF,  p-value: < 2.2e-16
```

In this lab, we will perform linear regression using the `glm()` function rather than the `lm()` function because the latter can be used together with `cv.glm()`.

```
# LOOCV can be done with the cv.glm function in the boot package
library(boot)
# ?cv.glm

# Compute LOOCV estimate of the test MSE
cv.err <- cv.glm(Auto, glm.fit)

# Resulting object contains many different things
names(cv.err)
```

```
## [1] "call"  "K"     "delta" "seed"
```

```r
# Two nums in the delta vector contain cv results
cv.err$delta
```

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

K-Fold CV

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.

```r
# K-Fold CV can be done as well

# Compute k-fold CV estimate of the test MSE
cv.err.k10 <- cv.glm(Auto, glm.fit, K = 10)

# Two nums in the delta vector contain cv results
cv.err.k10$delta
```

```
## [1] 24.18370 24.17104
```

Now consider the following code. What does this do? #This code uses the cv.glm() function to implement k folds, with k = 10

```r
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)$delta[1]
}
cv.err.k10$delta
```

```
## [1] 24.18370 24.17104
```

What do you find? #We find that the results are the same, but the time to give results is more here.

The Bootstrap

The bootstrap approach can be applied in almost all situations!

You need the following steps:

- Create a function that computes the statistic of interest.
- Use the `boot()` function, which is part of the `boot` library, to perform the bootstrap by repeatedly sampling observations from the data set **with replacement**.

We use the `Portfolio` dataset, described in the reading.

```r
# A function that takes as input (X,Y)
# as well as a vector indicating which
# observations should be used to estimate
# alpha

alpha.fn <- function(data, index) {
  X <- data$X[index]
  Y <- data$Y[index]
  res <- (var(Y) - cov(X,Y))/(var(X) + var(Y) -2*cov(X,Y))
  return (res)
```

```r
}

# Test the function
alpha.fn(Portfolio, 1:100)
```

```
## [1] 0.5758321
```

Now we use the boot() function to produce $R = 1,000$ bootstrap estimates for α.

```r
# Boostrap estimate of alpha
boot(Portfolio, alpha.fn, R=1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Portfolio, statistic = alpha.fn, R = 1000)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.5758321 0.001093608  0.09165659
```

The bootstrap approach can be used to assess the variability of the coefficient estimates and predictions from a statistical learning method.

Below we assess the variability of the estimates for the intercept and slope terms for the linear regression model that uses horsepower to predict mpg in the Auto data set.

```r
# First create the function for the statistic of interest
boot.fn  <-  function(data, index){
  res <- coef(lm(mpg ~ horsepower,
                 data = data,
                 subset = index))
  return(res)
}

# Boostrap estimated of the standard error
boot(Auto, boot.fn, 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Auto, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 39.9358610  0.0927201519 0.820762114
## t2* -0.1578447 -0.0008216145 0.007062626
```

```r
# Recall
summary(lm(mpg ~ horsepower,
           data = Auto))$coef
```

```
##                Estimate  Std. Error   t value      Pr(>|t|)
## (Intercept) 39.9358610 0.717498656  55.65984 1.220362e-187
## horsepower  -0.1578447 0.006445501 -24.48914  7.031989e-81
```

Interestingly, these are somewhat different from the estimates obtained using the bootstrap.