Lab/HW 11: Bootstrap

Your lab/homework must be submitted in with two files: (1) R Markdown format file; (2) a pdf or html file, unless otherwise stated. Other formats will not be accepted. Your responses must be supported by both textual explanations and the code you generate to produce your result.

Bootstrap

1. [50 pt] Complete the following code to demonstrate the coverage performance of the bootstrap confidence interval for the mean of a true model that is gamma distributed `scale = 2` and `shape = 2`, 200 observations.

What would adversely impact the estimated coverage probability? Demonstrate.

```
shape <- 2
scale <- 2
n <- 200
true_mean <- shape / scale

covered <- 0
for (s in 1:1000) {
  M <- 500   # size of bootstrap sample
  stat <- ""
  smpl <- rgamma(n, shape, scale)

  for (m in 1:M) {
    idx <- ""
    stat[""] <- mean(smpl[""])
  }

  if (quantile(stat, 0.025) <= "" & "" <= quantile(stat, 0.975)) {
    covered = covered + 1
  }
}

covered / 1000
```

2. [50 pt] Estimate standard error of robust psi regression estimators.

In Lecture 8 we developed a "robust" regression based on fitting the `psi` function. In the lecture, we fitted the parameters using our gradient descent method. In the interest of time, fit the parameters here using `optim`. The optimization procedure provides us with point estimates, but we have no measure of uncertainty for them: neither a standard deviation nor confidence intervals to determine whether or not they are different from 0 with any statistical certainty. The bootstrap method allows us to do so.

Use the same data and regression model as before, data being `MASS::cats` and model begin `Hwt` (heart weight) as response and `Bwt` (body weight) as the single predictor.

a. Estimate the coefficients of a simple linear regression where the fitting is based on minimizing the mean of the psi-error.
b. Construct a bootstrap sample for the estimated coefficients of a simple linear regression where the fitting is based on minimizing the mean of the psi-error.
c. Estimate the standard deviations of each of the coefficients.
d. Construct 95% confidence intervals for the coefficients and, based on those, determine whether or not we can reject a null hypothesis that each of them is equal to zero.

Recall, the psi function is given by:

```r
psi <- function(x, c = 1) {
  return (
    ifelse(
      abs(x) > c,
      2 * c * abs(x) - c^2,
      x^2
    )
  )
}
```

And the objective function to minimize is given by:

```r
mean_psi_regression <- function(b, Y, X, c = 1) {
  return( mean ( psi( Y - ( b[1] + b[2] * X), c ) ) )
}
```

Extra Credit [40 pt]

Rcpp - The Sieve of Eratosthenes

The sieve of Eratosthenes is an algorithm for finding all prime numbers up to any given limit.

To find all the prime numbers less than or equal to a given integer n by Eratosthenes' method:

- Create a list of consecutive integers from 2 through n: $(2, 3, 4, ..., n)$.
- Initially, let p equal 2, the smallest prime number.
- Enumerate the multiples of p by counting in increments of p from $2p$ to n, and mark them in the list (these will be $2p, 3p, 4p, ...$; the p itself should not be marked).
- Find the smallest number in the list greater than p that is not marked. If there was no such number, stop. Otherwise, let p now equal this new number (which is the next prime), and repeat from step 3.
- When the algorithm terminates, the numbers remaining not marked in the list are all the primes below n.

Pseudo-code for the algorithm is given by:

```
algorithm Sieve of Eratosthenes is
    input: an integer n > 1.
    output: all prime numbers from 2 through n.

    let A be an array of Boolean values, indexed by integers 2 to n,
    initially all set to true.

    for i = 2, 3, 4, ..., not exceeding the square root of n do
        if A[i] is true
            for j = i^2, i^2+i, i^2+2i, i^2+3i, ..., not exceeding n do
               A[j] := false

    return all i such that A[i] is true.
```

In this exercise, we will implement the algorithm in R and translate it to C++ twice: using Rcpp and RcppArmadillo.

1. [10 pt] Write an R function that implements the algorithm as described above. In this implementation you will need nested loops, please keep them both explicit (that is, do not vectorize any of them).
2. [5 pt] The outer loop cannot be vectorized because the actions taken in it in step `i` depend on values that were determined in step `i-1`. However, the inner loop can be vectorized. Find a way to improve the code by vectorizing the inner loop using an R technique for vectorization.
3. [20 pt] Translate your code to C++ using the Rcpp interface and objects. Think carefully about vector subsetting in Rcpp (hint: it is often similar to R, but you have to check your work carefully)
4. [5 pt] Compare the speed of execution of all of your functions for `n = 10000`.