R Programming: Worksheet 6

1. **Straightforward examples**

   For the following, perform the below in a parallel manner, using 2 or more cores. Here you can choose between either a fork approach using `mclapply` or similar, or a socket approach using `makeCluster` and `parSapply` or similar. If you're on a Windows computer you'll need to use the sockets approach for true parallel computing.

   (a) For 100 replicates each, calculate the average of 10000 draws from a random distribution with mean 0 and standard deviation 1

   (b) Calculate the column average for each column of the built in dataset `mtcars`

2. **Bagging**

   Bootstrap aggregating, or bagging, is a technique in machine learning designed to improve the accuracy of predictors, as well as reduce the variance in the estimated predictions and help to reduce overfitting. In brief, semi-formally, given some training set D of size n, bagging generates m new training sets D_i, by sampling from D uniformly with replacement. Then, m models are fitted, one on each D_i, and prediction is done using averaging over the predictions from the m models.

   We use data from `https://www.kaggle.com/uciml/autompg-dataset` that has been lightly modified that you can download as `autompg_clean.csv` from the course website. We will try to predict `mpg` (miles per gallon) using the other variables, using linear regression, and bagging. Here we will use parallel computing to speed up the model building step.

   (a) Read in `autompg_dataset.csv`, and have a quick look at the contents. Sample 80% of the rows without replacement to form a training dataset, and make a test dataset with the remaining samples *Note that `read.csv(file)` or similar will produce a column called `X`, which is the row number. You can either use `read.csv(file, row.names = 1)` to load it in, or just generally discard it after loading in the data.*

   (b) Write a function that performs linear regression using `lm` using a bootstrapped sample of the training data. Predict `mpg` using all other variables, except `car.name`

   (c) Using parallel computing, run this function 100 times, to generate 100 models using the 100 bootstrapped training samples

   (d) *Optional* Build a function that generates the bagged estimate on the test set data, and compare the correlation between the bagged model and the truth `test[, "mpg"]`. Compare this to a doing a more conventional single linear regression using all the training data once.

3. **Bootstrap hypothesis testing**

   Here we will consider the following algorithm for comparing the means of two samples (from Efron, B.; Tibshirani, R. (1993). An Introduction to the Bootstrap.)

   See also `https://en.wikipedia.org/wiki/Bootstrapping_(statistics)` the Wikipedia entry for bootstrapping.

   Let x_1, ..., x_n be a random sample from some distribution F with sample mean $\bar{x}$ and sample variance σ_x^2. Let y_1, ..., y_m be another, independent random sample from some distribution G with mean $\bar{y}$ and sample variance σ_y^2.

i Calculate

$$t = \frac{\bar{x} - \bar{y}}{\sqrt{\sigma_x^2/n + \sigma_y^2/m}}$$

ii Create two new data sets whose values are $x_i^1 = x_i - \bar{x} + \bar{z}$ and $y_i^1 = y_i - \bar{y} + \bar{z}$ where $\bar{z}$ is the mean of the combined samples

iii Draw a random sample x_i^* of size n with replacement from x_i^1 and another random sample y_i^* of size m from replacement from y_i^1

iv Calculate the test statistic

$$t^* = \frac{\bar{x^*} - \bar{y^*}}{\sqrt{\sigma_x^{*2}/n + \sigma_y^{*2}/m}}$$

v Repeat iii and iv B times to collect B values of the statistic

vi Estimate the p-value as

$$B = \frac{\sum_{i=1}^{B} I\{|t_i^*| \geq |t|\}}{B}$$

(a) First, write an `R` function that does sampling with replacement for some vector `x`. Use `runif` as a source of randomness, and use `ceiling` as a way of rounding continuously distributed numbers to integers.

(b) What is the computational complexity of the function you just wrote? Assume that `runif` takes n time when requesting n random numbers, and that accessing an element of a vector is free.

(c) Verify the computational complexity class of your function by running it for variously sized input, and plotting input size versus run time. To get a clearer picture, you might need to average your function call for a given size over multiple runs

(d) Evaluate the computational complexity of the algorithm at the start of this question. What is its computational complexity?

(e) Write the above algorithm into R *Optional, just use the provided version of this function in the provided code*

(f) Check the computational complexity of this implementation matches its theoretical expectations by plotting run time for values of B, m and n. Is it as expected?