Biostatistics 615 Learning Exercise #10 (10 pts)

Due by November 5th 2024 (Tuesday) 11:59pm. Use Gradescope (via Canvas) to submit an R file.

- Your submission should only contain one `R` file named `neuralNetworkNelderMead.R` that contains a function named `neuralNetworkNelderMead(p, df)`.
- Your code will be evaluated in Gradescope using 10 different test cases using an automated script. Full credit will be given if your code passes all test cases.
- You are allowed to submit multiple times before the deadline, but only the last submission will be graded. Automated feedback will be provided for each submission.
- You need to implement the function to work with arbitrary (valid) input values beyond the 10 cases tested. If you tweak your implementation so that your functions works specifically for the test cases, you will not receive any credit.
- Implement your function as efficient as you can. If your program does not finish within the time limit for each test case, you will lose the points for those test cases. Note that the official solution finishes much faster than the test cases, so this should be a reasonable time limit.

Problem 1 - Fit a Neural Network Model using Nelder-Mead (10 pts)

Write an R script `neuralNetworkNelderMead.R` that contains `neuralNetworkNelderMead(p, df)` to fit a neural network model using the Nelder-Mead algorithm. Suppose $\{X_i, Y_i\}_{i=1}^n$ are the training dataset where X_i and Y_i are both scalars. To model the relationship between Y_i and X_i, we consider a one-layer neural network model with p nodes: for $i = 1, \ldots, n (100 \leq n \leq 200)$:

$$Y_i = \alpha_0 + \sum_{j=1}^{p} \alpha_j \text{GeLu}(\alpha_{p+j} + \alpha_{2p+j} X_i) + \epsilon_i$$

where $\epsilon_i \sim \text{N}(0, \sigma^2)$ and

$$\text{GeLu}(x) = x\Phi(x) = \frac{x}{2}\left[1 + \text{erf}(x/\sqrt{2})\right]$$

where $\Phi(x)$ is the standard normal cumulative distribution function and

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

To fit this neural network model, we consider the least squares estimates by minimizing the following objective function with respective to $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{3p})^T$:

$$F(\boldsymbol{\alpha}) = \frac{1}{n} \sum_{i=1}^{n} \left\{ Y_i - \alpha_0 - \sum_{j=1}^{p} \alpha_j \text{GeLu}(\alpha_{p+j} + \alpha_{2p+j} X_i) \right\}^2.$$

To solve this problem, use the Nelder-Mead algorithm in Topic 6. Instead of implementing the Nelder-Mead algorithm from scratch, it is strongly recommended to copy the `Nelder.Mead()` function provided in https://bit.ly/615top06r. You may need to modify the code or write a new function to fit the following requirements.

- Use the same stopping criterion in the original code
- Set tolerance parameter to 10^{-5}.
- You need to specify sufficiently large number of iterations to ensure the convergence.
- The initial values of $\boldsymbol{\alpha}$ are all set to zeros.
- The input arguments of the `neuralNetworkNelderMead(p, df)` function are the number of nodes p (integer) and a data frame `df` containing two vectors: `X` and `Y`.
- The function should return a vector of the following three values:
  1. Minimum value of the objective function
  2. Number of iterations in the Nelder-Mead algorithm
  3. Number of evaluations of the objective function

Note that, in order to meet the requirements, you need to modify the Nelder-Mead algorithm described in `https://bit.ly/615top06r`. to keep track of the total count of evaluations of the objective function. Using a global variable assignment is a simple way to achieve this.

Example output of running the test code is given below:

```
> df = read.table('test.1.tsv',header=TRUE)
> head(df)
          X         Y
1 -3.141593 0.9706502
2 -3.103742 0.9392261
3 -3.065892 0.8200233
4 -3.028041 0.9437523
5 -2.990191 0.7037756
6 -2.952340 0.9143716
> neuralNetworkNelderMead(10, df)
[1] 8.918325e-03 1.895000e+03 2.504000e+03
```

The input tsv file relevant to this example is provided on the Canvas homework page as `ex10testcase1.zip`.

For all problems, you can include as many functions as you want as long as your code contains the required function.

There are specific requirements in the implementation:

- You are NOT allowed to use any functions outside the `base` package in your implementation, except for the `pnorm()` function that evaluates the standard normal cumulative distribution function. Make sure to call `pnorm()` as is, not as `stats::pnorm()` or in other ways. Use `help(...)` to check whether a function you want to use belongs to the `base` package or not.
- Your answer should be accurate up to 5 significant digits for each output values.
- Each test case must finish within 5 seconds.

You do not need to implement error handling for malformed arguments in this function.