Biomedical Data Science 2021 Homework 1

Due: April 9th (Friday) 11:59pm EST

Submission should be done in Canvas

MCDB & MBB (non-programming)

Submit a single file answering the following questions:

1. (25pt) Multiple sequence alignments (MSA) cannot be efficiently handled using purely dynamic programming. Choose one existing MSA software and describe how it implements MSA. (for example, Muscle, clustalW, Kalign, MView, T-coffee, etc)

2. (25pt) Align the following two sequences using the Smith-Waterman algorithm (local alignment) with the following scores. In addition to writing out the alignment matrix, indicate the traceback and write out the final alignment.

Match: 2; Mismatch: -1; Gap: -1

Sequence 1: ATTTGCAGT

Sequence 2: GTATGGTTGA

3. (25pt) ChIP-seq is a common method to determine protein-DNA interaction on a genome-wise scale. The exact sites of binding must be inferred from sequence reads of the DNA that is purified along with the protein of interest. Describe an algorithm for determining protein-DNA binding sites from ChIP-seq data. See the following citation for a list of example algorithms:

Wilbanks, Elizabeth G., and Marc T. Facciotti. "Evaluation of algorithm performance in ChIP-seq peak detection." *PloS one* 5.7 (2010): e11471.

4. (25pt) Machine learning approaches have become extremely useful in the analysis of biological data. After reading the paper referenced below, answer the following:

Ghandi, Mahmoud, et al. "Enhanced regulatory sequence prediction using gapped k-mer features." *PLoS Comput Biol* 10.7 (2014): e1003711.

- What are the researchers trying to predict/infer?
- What information is being used for the prediction? What is the logic behind using these data?

- What preprocessing steps are used to prepare the data for machine learning?
- What is the model the researchers use?
- How do the researchers evaluate their predictions? Were they effective? What biological insight was gained?

5. (Optional extra credit) You can obtain extra 10% points for completing the programming assignment correctly.

CBB & CPSC & S&DS (programming)

Submit a single script file including all the codes and comments. All supplementary files could be find here or in Canvas files.

Requirements

- Implement the Smith-Waterman algorithm with Python or R. You may use the templates below. All scripts must be done from scratch. Standard libraries (I/O, math-related) are allowed. Except Numpy and pandas in python, or similar auxiliary functions in R, other pre-existed libraries are **NOT** allowed.
- The program should automatically read in the similarity matrix file called “blosum62.txt” and input sequences in a file called “input.txt”, where each line is a sequence.
- For a given gap penalties, the program should output the best alignment of two sequences. The default gap penalties are as follows: opening gap -2, extension gap = -1
- The output should contain (1) Sequences, where the input is shown (2) Score matrix, where the completed scoring matrix are shown in tab-delimited format (akin to the hand-drawn scoring matrix) (3) Best alignment output as well as the alignment score. Please see file sample-output.txt for details of output format. These will constitute 90% of your grade, with the remaining 10% coming from your programming style (e.g., clear comments).
- All submitted programs should run successfully and be well-commented. Please comment with a Usage line and example line (see template below for examples). We will test and execute the script exactly as you write out in the example line.
- For example, if you have the following in your code:

```
## Usage: python hw1.py -i <input file> -s <score file>
## Example: python hw1.py -i input.txt -s blosum62.txt
```

Then running `python hw1.py -i sample-input.txt -s blosum62.txt` should give you the exact file output as sample-output.txt (by “exact” we mean using bash `diff -E -b file1 file2` result in no output). And we will run your code with `python hw1.py -i input.txt -s blosum62.txt` and the output of this will be compared to our answer (again with `diff`)

- (Optional extra credit) You can obtain extra 10% points for publishing your program into GitHub as a package. Users should be able to install your package with a simple command such as "pip install git+git://github.com/author/package.git" or "install_github("author/package")". For full extra credit, it is required to have a clear documentation in your GitHub repository (code example, and example input and output files) as a README markdown file.

Templates

If you use Python, you could use the following code template. Recommend Python 3.x but it would be good as long as the output is the same as expected.

```
#!/usr/bin/python
__author__ = "FirstName LastName"
__email__ = "first.last@yale.edu"
__copyright__ = "Copyright 2021"
__license__ = "GPL"
__version__ = "1.0.0"

### Usage: python hw1.py -i <input file> -s <score file>
### Example: python hw1.py -i input.txt -s blosum62.txt
### Note: Smith-Waterman Algorithm

import argparse

### This is one way to read in arguments in Python.
parser = argparse.ArgumentParser(description='Smith-Waterman Algorithm')
parser.add_argument('-i', '--input', help='input file', required=True)
parser.add_argument('-s', '--score', help='score file', required=True)
parser.add_argument('-o', '--opengap', help='open gap', required=False,
default=-2)
parser.add_argument('-e', '--extgap', help='extension gap', required=False,
default=-1)
args = parser.parse_args()

### Implement your Smith-Waterman Algorithm
def runSW(inputFile, scoreFile, openGap, extGap):
    ### calculation
    ### write output

### Run your Smith-Waterman Algorithm
runSW(args.input, args.score, args.opengap, args.extgap)
```

If you use R, you could use the following code template.

```
#!/usr/bin/env Rscript

### Usage: Rscript --vanilla hw1.R <input file> <score file>
### Example: Rscript --vanilla hw1.R input.txt blosum62.txt
### Note: Smith-Waterman Algorithm

### This is one way to read in arguments in R
args = commandArgs(trailingOnly=TRUE)

if (length(args)<2) {
stop("At least two arguments must be supplied (inputFile, scoreFile).n",
call.=FALSE) } else if (length(args)>=2) {
    # default gap penalties
    args[3] = -2
    args[4] = -1 }

## Specifying author and email
p <- c(person("FirstName", "LastName", role = "aut", email =
"first.last@yale.edu"))

## Implement your Smith-Waterman Algorithm
runSW <- function(inputFile, scoreFile, openGap = -2, extGap = -1) {
    ### calculation
    ### write output
}

## Run the main function and generate results
runSW(inputFile=args[1], scoreFile=args[2], openGap=args[3], extGap=args[4])
```