Biomedical Data Science 2020: Homework Assignment 1

DUE DATE: March 1st (Sunday) 2020, 11:59pm

Choose to do either MCDB & MBB (non-programming) or CBB & CS & S&DS (programming) assignment, depending on your academic affiliation. No late submissions will be accepted.

MCDB & MBB (non-programming)

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. . .)
2. (25pt) Align the following two sequences using the Smith-Waterman algorithm (local alignment), with the following scores: Match: 2; Mismatch: 0; Gap: -1. In addition to filling out the alignment matrix, indicate the traceback and write out the final alignment.

		A	A	A	A	C	G	C	T	T
	0	0	0	0	0	0	0	0	0	0
T	0									
T	0									
T	0									
A	0									
A	0									
T	0									
C	0									
G	0									
C	0									

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, EG, Facciotti, MT (2010). Evaluation of algorithm performance in ChIP-seq peak detection. *PLoS ONE*, 5, 7: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: Lee, D., Karchin, R. and Beer, M.A. (2011), Discriminative prediction of mammalian enhancers from DNA sequence. *Genome Research*, 21: 2167-2180.
   - 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.

CBB & CPSC & S&DS (programming)

Scripting must be done from scratch, without the use of any pre-existing packages. Standard libraries (I/O, math-related) are allowed. The programming task is to implement the Smith-Waterman local alignment algorithm for protein sequences.

Requirements:

1. Implement the algorithm from Python or R templates below. 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.
2. For a given gap penalties, the program should output the best alignment of two sequences. The default gap penalties are as follows:

   `Gap penalties: opening gap -2, extension gap = -1`
3. (50pt) The output should contain a human-readable alignment such as the following:

   ```
   T C W A
     |   |
   S C - A
   ```

   where | represents amino acid identity and - represents a sequence gap.
4. (50pt) For each sequence pair, the output must include the completed scoring matrix (including the sequences themselves) in tab-delimited format (akin to the hand-drawn DP scoring matrix), best-scoring local alignment(s) and the score. (Just to be precise, the completed scoring matrix contains the best score in the alignment up to this point.) These will constitute 90% of your grade, with the remaining 10% coming from your programming style (e.g. clear comments).
5. All submitted programs should run successfully, i.e., have no coding errors, and be well-commented.
6. (Optional extra credit) You can obtain extra 10% points for publishing your program into GitHub as a package. Users should be able to download your code with a simple package install command such as "sudo pip install git+git://github.com/author/package.git" or "install_github("author/package")". For full credit, it is required to have a clear documentation in your GitHub repository (code example, and example input and output files) as markdown.

Templates:

If you use Python, please use the following code template. Use Python 2.7.x for backward compatibilities.

```
#!/usr/bin/python
__author__ = "FirstName LastName"
__email__ = "first.last@yale.edu"
__copyright__ = "Copyright 2020"
__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

### Scripting must be done from scratch, without the use of any pre-existing packages.
### Python standard library (I/O) and numpy are allowed.
```

```python
import argparse

### This is one way to read in arguments in Python.
### We need to read input file and score file.
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):
    ### Print input and score file names. You can comment these out.
    print ("input file : %s" % inputFile)
    print ("score file : %s" % scoreFile)
    print ("open gap penalty : %s" % openGap)
    print ("extension gap penalty : %s" % extGap)

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

If you use R, please use the following code template:

```r
#!/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 We need to read input file and score file.
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) {
  ### example
  print(inputFile)
  print(scoreFile)
  print(openGap)
  print(extGap)
}

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