BMI 203 Winter 2020
Project #1
Due Wednesday /// 01/27/2021 by 11:59pm PST

In this assignment, you will implement two classical alignment algorithms and then evaluate each algorithm's performance with a range of parameters. There are two parts to this assignment and Part 2 requires completion of Part 1. We recommend reading through both Part 1 and Part 2 before beginning this assignment.

Project skeleton: https://github.com/ucsf-bmi-203-2021/Project1

Part 1:
Affine-gap Smith-Waterman and Needleman-Wunsch share the common hallmarks of dynamic programming algorithms. We would like efficient implementations that maximize code reuse.

With this in mind, in python, first implement a generic alignment class, *PairwiseAligner*, defining what methods you think could be shared between these two algorithms. Next, create separate classes for each alignment algorithm that inherits from *PairwiseAligner*, and implement methods unique to each algorithm.

PairwiseAligner should meet the following specifications.

- Class initialization requires parameters that describe the alignment procedure including a score matrix (read in from a separate file).
- Class contains a method *align* that accepts two protein sequences and returns an alignment.
- The scores and other related quantities for the alignment are easily accessible.

To help determine the functionality to add to your generic class *PairwiseAligner*, please answer the following questions.

1) For both Smith-Waterman and Needleman-Wunsch algorithms:
    a) What are the parameters and variables required for algorithm initialization, execution, and termination?
    b) What quantities are returned?
    c) What is the runtime complexity?

2) What functionalities in initialization, execution and termination are shared between these algorithms? Which are not shared?

3) How does affine-gap based alignment differ from linear-gap alignment in terms

of implementation?

4) Write out an API (methods, variables) with which a user could interact with your class. For each method, define what it does, the arguments and data types, the functionality, and what it returns. **Please add this answer to your README.md for your github repo or use full python docstrings with a tool such as Sphinx. This will serve as "documentation" for your TAs to review your code.**

Implement your *PairwiseAligner* class following your API.

5) *(extra credit: +5%)* Extend your implementation to include a variant of Smith-Waterman / Needleman-Wunsch from the list below (or devise your own).
    a) Global alignment permitting variable start / end points (overlap alignment)
    b) Suboptimal alignment (2nd best non-intersecting alignment)
    c) Repeat alignment (return an alignment made by repeating the query)

To confirm your implementation is functioning correctly, provide unit tests that account for the following:

- I/O of protein FASTA sequences
- I/O of a scoring matrix
- Identical sequences are correctly aligned for all algorithms implemented
- Sequence alignment scores are correctly calculated for all algorithms implemented

Part 2:
We would like to evaluate how effective these alignment algorithms are at separating "true" alignments from false or spurious alignments. Using the score associated with each alignment, we will generate a receiver-operator characteristic curve (ROC curve), as well as evaluate other metrics for algorithm performance.

Provided with the sequence data are two files indicating a set of real (Pospairs.txt) and spurious (Negpairs.txt) alignments. Use these pairs as your "true" and "false" cases.

1) With the BLOSUM50 matrix and a gap opening cost of 11 and a gap extension cost of 3, locally align these sequences and visualize the distribution of alignment scores. How would you describe this distribution?

2) Generate a confusion matrix indicating the frequency of false positives, false negatives, true positives, and true negatives when using the average alignment score as a threshold. What is the threshold value, and how does the confusion matrix suggest this algorithm performed?

3) Create a ROC plot which shows the fraction of true positives on the Y axis and the fraction of false positives on the X axis. Please take care to make your ROC

plots square, with both X and Y axes limited to the range [0:1].

4) Determine the area under the ROC curve (AUROC). What does this value indicate about the performance of this algorithm? Can you confidently assess the performance of this algorithm with this value alone? Why or why not?

Having assessed the local alignment algorithm using the BLOSUM50 matrix, we are interested in determining how adjusting the gap penalties influences its performance.

5) Once again, using local alignment, try a range of gap opening (1-20) and gap extension (1-5) costs with the BLOSUM62 matrix. Using the AUROC of each approach, determine which gap penalty performs the "best". What does this pair of values suggest about the evolution of these sequences and the likelihood of insertions / deletions?

6) Using the BLOSUM50, BLOSUM62, PAM100 and PAM250 scoring matrices, evaluate the performance of the global alignment algorithm using the selected pair of best performing gap penalties. Evaluate your *optional* additional extension algorithm as well, using the parameters it requires.

7) For each algorithm, generate a ROC plot demonstrating performance using each of the 4 matrices, given the fixed gap costs. Of all these algorithms, which performs the best as measured by AUROC?

8) Comment qualitatively on the best algorithm. What does the best performing algorithm indicate about the origin of these sequences?

To submit this assignment,

- Use Google Classroom to submit a link to your **public** Github repository that:
    - Contains a single pdf or ipython notebook labeled **Firstname_Lastname_BMI203_HW1.(pdf/ipynb)**, including your answers to the above questions in prose and associated plots.
    - Contains a README.Md that explains the layout of your repo (what code is where, how to use your implementation)
    - Contains all code relevant for this assignment
    - Is commented explaining the role of each functional code block
    - Has passed the implemented unit tests (as indicated by the github badge)
    - ***Note!*** we will only consider commits before the submission deadline.