Ryder Easterlin
BMI 203: Project 1
January 27, 2021

Part 1

1) For both Smith-Waterman and Needleman-Wunsch algorithms:
a) What are the parameters and variables required for algorithm initialization, execution, and termination?

For affine gap implementations, both algorithms require a protein substitution scoring matrix (such as BLOSUM62, or PAM250), a gap opening penalty, and a gap extension penalty as parameters. Both algorithms also require two protein sequences as input variables.

b) What quantities are returned?

Basic implementations of Smith-Waterman and Needleman-Wunsch should return at least a score for the alignment and the two aligned sequences. Implementations can also return the percent identity between the two sequences, the length of the alignment, and the gap percentage of the alignment, among other metrics.

c) What is the runtime complexity?

Both algorithms have a runtime complexity of *O(mn)*, where *m* and *n* are the respective lengths of the two input sequences.

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

At a broad level, both algorithms perform a dynamic programming step, where all possible alignments between the two sequences are scored according to set of recurrence rules that are largely similar between the two algorithms. The optimal alignments are then constructed in a traceback step, where a traceback matrix generated in the dynamic programming step is used to construct the alignment from back-to-front, indicating whether to match a residue pair between the sequences, insert a gap in sequence 1, or insert a gap in sequence 2. The score of the algorithm can be determined using only the dynamic programming step. For local alignment, the score of the optimal alignment is simply the largest number in the scoring matrix. For global alignment, the score of the optimal is stored in the bottom-right cell of the scoring matrix—since global alignments incorporate both full sequences in the alignment, this cell represents the alignment of the end residue/gap in each sequence.

Initialization of the scoring matrices is the first place where the two algorithms differ: for Smith-Waterman, the first row and column are set to 0; for Needleman-Wunsch, the first row and column are subjected to the gap penalty.

The recurrence rules in the dynamic programming step are also slightly different between the two algorithms. In the dynamic programming step of Needleman-Wunsch, the score of an individual cell is equal to the maximum of the optimal sequence generated in the previous iteration with either a match added to it, or a gap added into either sequence. Only a slight modification is added for Smith-Waterman: if the Needleman-Wunsch recurrence rule is negative, simply input a zero, marking to "terminate" the ongoing alignment.

Finally, the traceback step is slightly different between the two. For Needleman-Wunsch, traceback is initiated at the bottom-right cell of the matrix and continues until the top-left cell is reached, therefore encompassing the full extent of both sequences. For Smith-Waterman, traceback is initiated at the largest score in the matrix, and continues until a 0 is found in the scoring matrix. However, these traceback steps share the logic of whether to match two residues or to insert a gap into either of the aligned sequences.

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

The same logic described above is mostly followed for affine-gap implementations of the algorithms. However, since affine gaps mean that the opening of a gap is penalized more than the extension of a gap (to better reflect the biological mechanisms that result in indels), two additional scoring matrices are used to determine if it is optimal to start a gap in a subalignment of the two sequences, or to continue an ongoing gap in either of the sequences in a different subalignment. Two additional scoring matrices are required to store the scores of the subalignments with ongoing gaps in the two respective sequences.

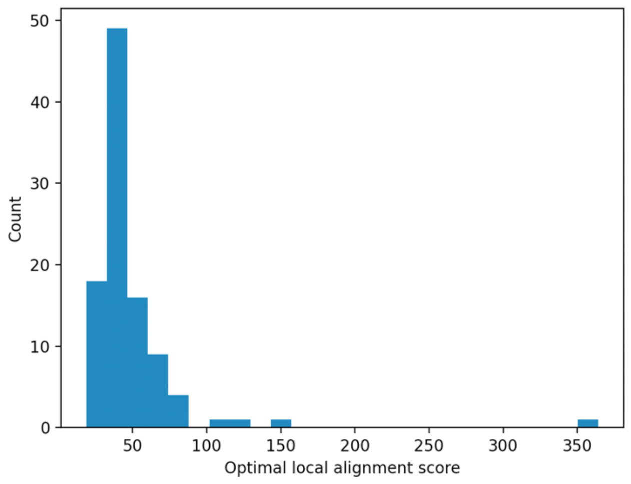
4) Write out an API (methods, variables) with which a user could interact with your class. For each method, define what it does, the arguments and data types, the functionality, and what it returns. Please add this answer to your README.md for your github repo or use full python docstrings with a tool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

API documentation was performed with Sphinx, and is in an HTML document in the Github repository.

Part II

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?



I would say this distribution looks most like a lognormal distribution, or just a left-skewed normal distribution.

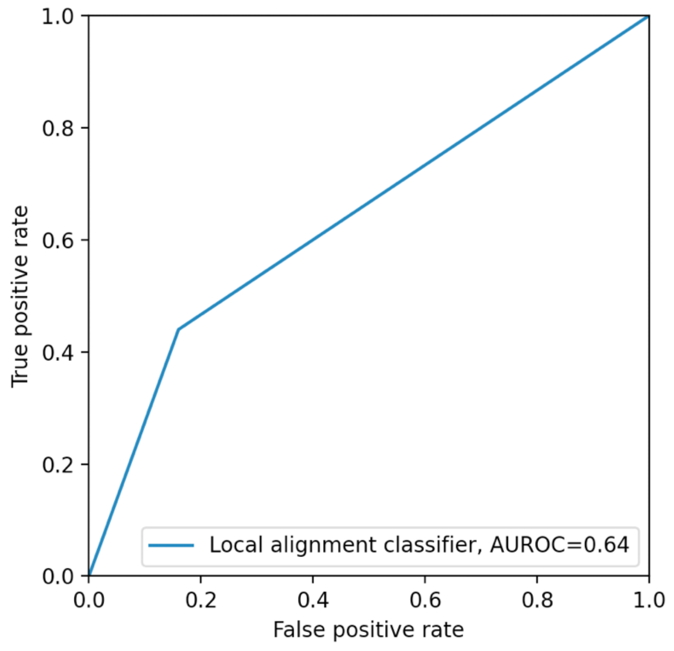
2) Generate a confusion matrix indicating the frequency of false positives, false negatives, true positives, and true negatives when using the average alignment score as a threshold. What is the threshold value, and how does the confusion matrix suggest this algorithm performed?

Threshold = 49.13

True positives = 22	False positives = 8
False negatives = 28	True negatives = 42

This confusion matrix indicates that the threshold is too high, since the number of false negatives greatly outnumbers the number of false positives, and there are a lot more true negatives than true positives. However, this does not really reflect on the alignment algorithm itself, since the threshold is a function of the alignment scores of arbitrary sequence pairs. Overall, the total algorithm of alignment and score prediction has an accuracy of 64%, compared to the 50% expected accuracy of a naïve classifier.

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].



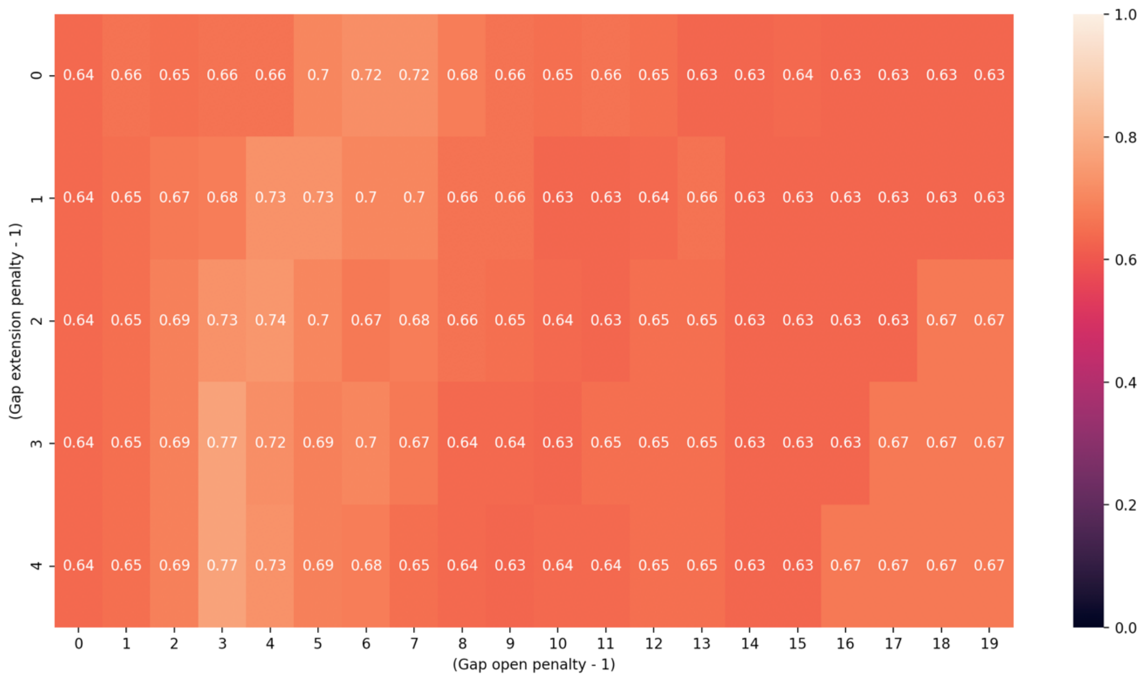
4) Determine the area under the ROC curve (AUROC). What does this value indicate about the performance of this algorithm? Can you confidently assess the performance of this algorithm with this value alone? Why or why not?

The AUROC is shown in the above plot and is equal to 0.64. This happens to equal the accuracy because the number of positive pairs equals the number of negative pairs. Because the AUROC is greater than .50, it can be said that this model is better than a random classifier, which would have an expected AUC of .50. That said, an AUROC of .64 is not particularly good generally speaking.

AUROC is a useful summary statistic to compare the performance of models, but as a metric alone does not capture the finer-grain details of what a model does well and not well. To fully assess the performance of a model, it is best to use a number of metrics, including but not limited to AUROC. F1 scores, accuracy, precision, and recall are all additionally good metrics to take into account when assessing the performance of models.

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?



- Note that the axes of the above plot are off by one

Interestingly, the best performing combinations of gap extension and gap start penalties have them relatively close to each other and at somewhat low numbers, therefore approximating or even equating to a linear gap alignment. Specifically, gap penalty schemes of (gap_open = 4, gap_extension = 5) and (gap_open = 4, gap_extension = 4) each resulted in an AUROC of .77. These schemes, especially the one where the gap extension penalty is greater than the gap open penalty, likely indicate that indels occur between the sequence pairs relatively frequently, but the indels themselves are typically pretty small.

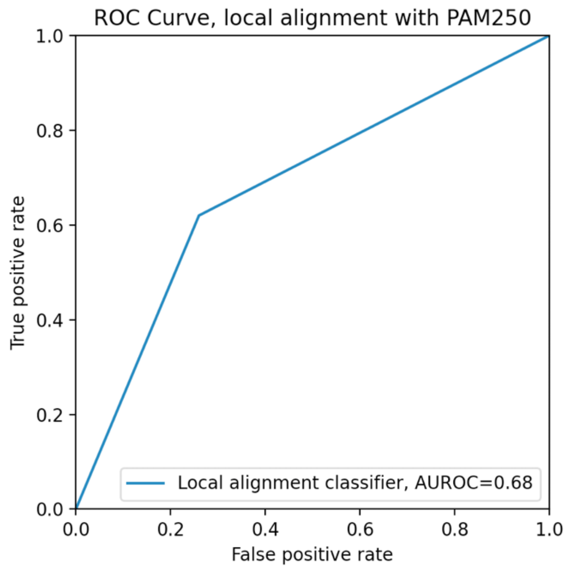
6) Using the BLOSUM50, BLOSUM62, PAM100 and PAM250 scoring matrices, evaluate the performance of the global alignment algorithm using the selected pair of best performing gap penalties. Evaluate your optional additional extension algorithm as well, using the parameters it requires.

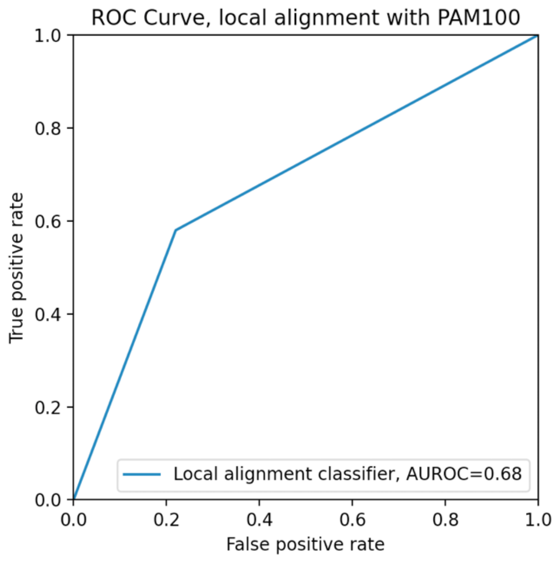
I decided to use a gap penalty scheme of (gap_open = 4, gap_extension = 5) for this question and the following question.

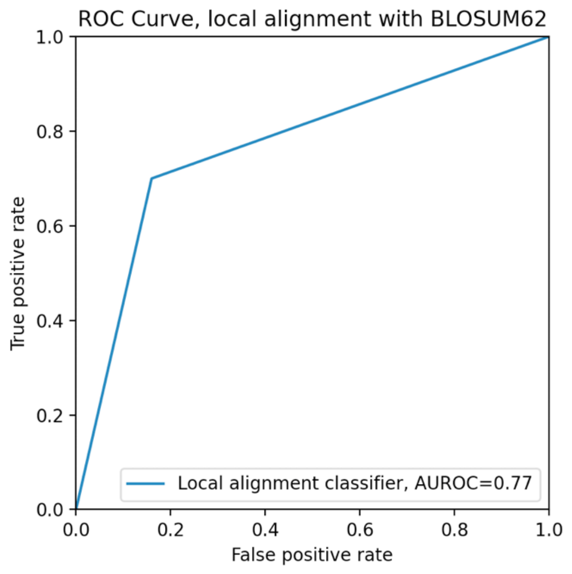
7) For each algorithm, generate a ROC plot demonstrating performance using each of the 4 matrices, given the fixed gap costs. Of all these algorithms, which performs the best as measured by AUROC?

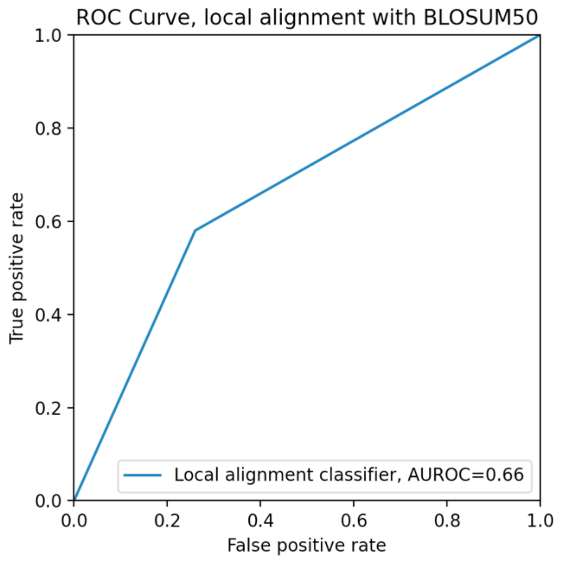
NOTE: My implementation of Needleman-Wunsch is not fully functional. It works for sequences that are largely similar but usually does not produce the optimal alignment for dissimilar sequences.

Local Alignment Results:

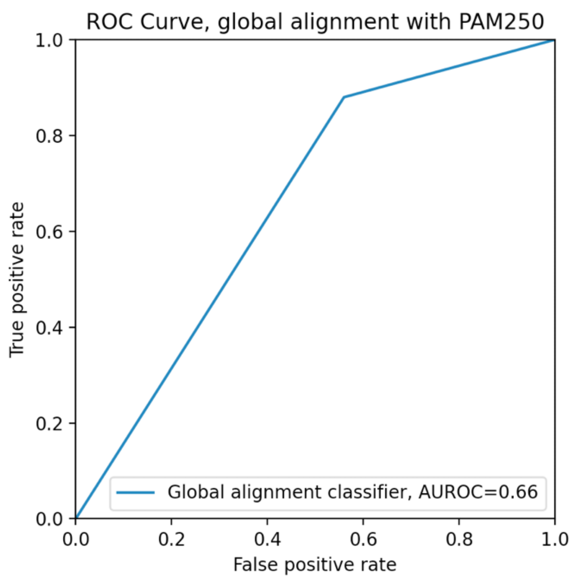


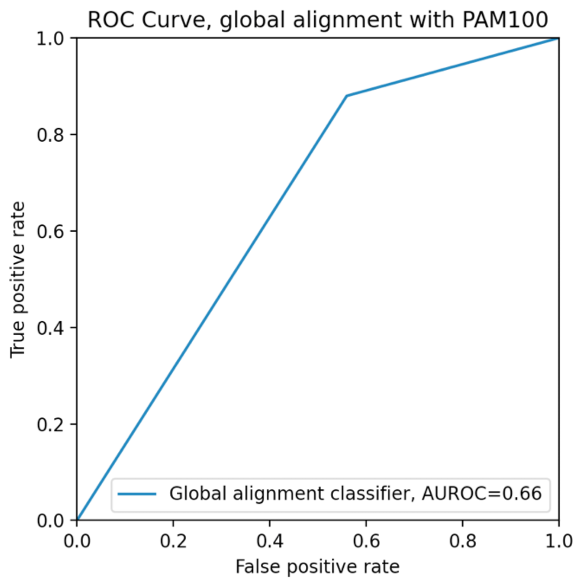


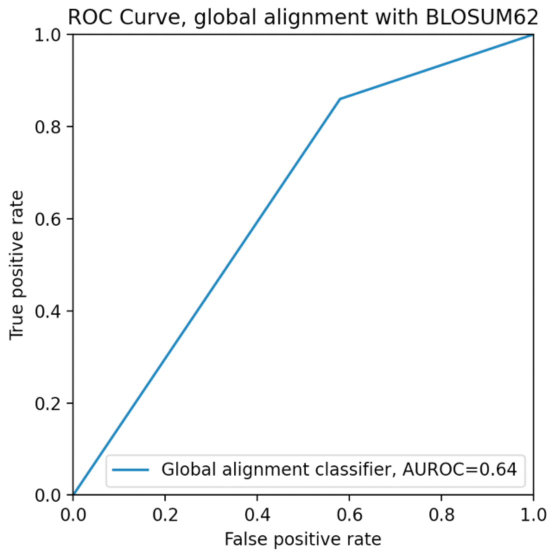


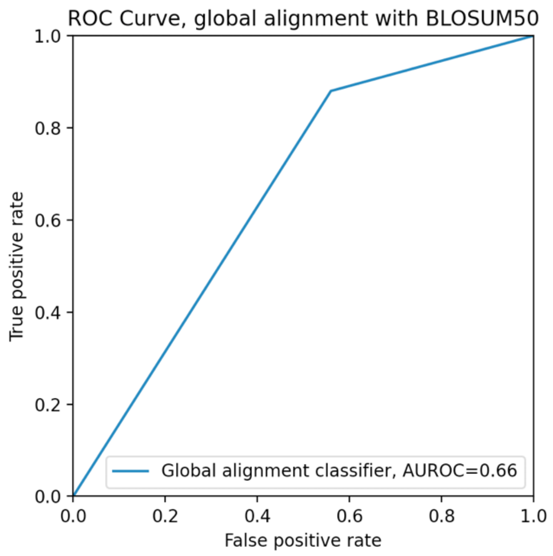


Global Alignment Results:









These results indicate that a local alignment algorithm with the BLOSUM62 substitution matrix performs best on aligning the sequence pairs provided.

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

Typically, local alignment performs better than global alignment on sequences that have conserved motifs/domains, but are evolutionarily distant. Because my Needleman-Wunsch implementation is not fully operational, it is a little hard to make judgments about the sequences themselves. That said, if these results were to hold even with a functional global alignment algorithm, then one could make the argument that these sequences have conserved domains, but likely come from organisms that are divergent.