Laboratory for Bioinformatics Tools – Spring 2022
Exercise 3: Multiple Sequence Alignment and Phylogenetics

Given: March 21, Due: April 11

General guidelines:

- Submit a single zip file with your code and a doc / pdf file with answers to the following questions. Make sure to write your ID number at the top of the doc / pdf. The name format of the submitted file should be "123456789_ex3.zip" – replace "123456789" with your ID number.
- Unless required otherwise, include only a single code file named "ex3.py" in the submission zip. This file should include all the code you wrote for this exercise.

Python guidelines:

- Test your code using python 3.8 or 3.9.
- Make sure to document and explain your code wherever needed.
- Code should be as efficient as possible.
- Code should be readable and well organized.
- It is advised to break code into functions wherever possible.

Part 1 – Reading a scientific paper

a. Read the 2012 paper by Collingridge and Kelly (see attached "Collingridge and Kelly 2012.pdf" file), and answer the following questions:
    - Write a pseudo code of the algorithm the authors proposed. Make sure to include the final step of the algorithm, in which the consensus MSA is constructed together with the scores of the columns.
    - Under which scenario the authors claim that *MergeAlign* notably outperforms the rest of the methods?
    - Briefly explain about the different ways the authors used to validate their method and compare it with the alternatives.
    - How did the authors select which scoring matrices to use in their evaluation?

Part 2 – Constructing phylogenetic trees and assessing statistical significance

In the following sections you will implement the UPGMA algorithm for constructing phylogenetic trees, and compare the performance of two different approaches for calculating the distance matrix input required for the algorithm.

b. Implement a python class "matrix" for representing a two dimensional matrix. In your implementation use a list of lists to represent the matrix. The constructor of the class should get the dimensions of the matrix as an input (i.e. the number of rows and the number of columns) and generate a matrix of zeros with the input dimension. The class should include two methods: "get" and "set". The first method, given two indices i and j, should return the values in the (i+1)-th row and (j+1)-th column of the matrix (i.e., the indices of the upper left cell in the matrix are (1,1)). The second, given two indices i, j and a number x, should update the value in the (i+1)-th row and (j+1)-th column of the matrix to be x.

In your implementation use "assert" (i.e., AssertionError) calls to check validity of the arguments in the constructor (e.g., the dimensions provided to the constructor must be integers > 0).

Use the "matrix" class throughout this exercise, whenever you need to represent a matrix.

c. Write a python function "upgma". Input: a distance matrix D of n sequences and a list of strings corresponding to the names of the n sequences. Output: a representation of the tree resulted by applying the UPGMA algorithm on D. Ignore the calculation of branch lengths in the output.

Represent the tree by recursively using a tuple for describing a join of two nodes. For example, for three sequences X, Y and Z, if the closest pair of sequences is X and Y then the UPGMA algorithm should return the tuple ((X,Y),Z). Note that this representation of the tree is not unique. For instance, in this example there are two equivalent representations: ((X,Y),Z) and ((Y,X),Z). Either equivalent representation is fine.

d. Write a python function "globalpw_dist". Input: a list of n strings (sequences) $S_1, \ldots, S_n$. Output: a distance matrix D, where the distance between each pair of sequences is their global pairwise alignment distance.

In order to calculate the global pairwise alignment distance of two sequences, first use the globalXX function from the Biopython package. Set the proper X parameters in the function's name such that: (1) Gap penalty (opening and extending) is -5, and (2) the similarity score for two aligned residues is based on the BLOSUM62 substitution matrix. More information can be found in the link: https://biopython.org/docs/1.75/api/Bio.pairwise2.html. Then, given the similarity scores of all pairs of sequences, convert them into distances as follows. Let S be

the matrix of similarity scores and let S_{ij} be the similarity score of sequences i, j, define the distance between these sequences D_{ij} to be $S_{max} - S_{ij} + 1$ where S_{max} is the maximal value in S.

e. Let $S_1, \ldots, S_n$ be a set of n sequences and let k be an integer greater than zero. Denote $C(S_i, \mathrm{w})$ to be the number of times that the k-mer (a word of length k) w appears in S_i (overlaps are counted as well), and denote $K(S_i, k)$ to be the set of all possible k-mers of length k that appear in S_i. We define the k-mer distance between S_i and S_j as follows:

$$D_{i,j} = \sqrt{\sum_{\mathrm{w} \in K(S_i,k) \cup K(S_i,k)} \left(C(S_i, \mathrm{w}) - C(S_j, \mathrm{w})\right)^2}$$

Write a python function "kmer_dist". Input: a list of n strings (sequences) $S_1, \ldots, S_n$ and an integer k>0. Output: a distance matrix D, where the distance between each pair of sequences is their k-mer distance.

f. Run the "upgma" function on the sequences in the attached "sequences.fasta" file. First, using the distance matrix calculated by the "globalpw_dist" function as an input to "upgma", and second, using the distance matrix calculated by the "kmer_dist" function as an input to "upgma". When running "kmer_dist" use k=3 as an input. Note that the names of the sequences are given in the fasta input file – the name of each sequence is given after the '>' character. Save the resulted representations of the two trees in a file called "globalpw_dist_tree.dnd" (when using "globalpw_dist") and in "kmer_dist_tree.dnd" (when using "kmer_dist"). See the attached "example.dnd" file for a tree representation.

Use the online iTOL tool for visualizing the two trees: http://itol.embl.de/upload.cgi. Use the tool's "export" option in order to generate figures of the trees and attach the figures to your submission. Name the files "globalpw_dist_tree_fig" and "kmer_dist_tree_fig". Are there any differences between the two trees (ignore branch lengths)? If so specify and explain the differences.

In the following sections we are interested in evaluating the statistical significance of the topology of our trees.

g. Construct an MSA on the n sequences $S_1, \ldots, S_n$**.** Use Biopython and the ClustalW algorithm in order to construct an MSA. For that, first download ClustalW from https://mafft.cbrc.jp/alignment/software/. Then, execute ClustalW on the input sequences in "sequences.fasta" using the ClustalW command-line wrapper of Biopython. Save the MSA result in a file called "sequences.aln.fasta" and attach the file to your submission.

h. Implement the following evaluation procedure in *eval_dist* function.

First, given an MSA file name (in *msa_aln_path* parameter) create a list of MSA aligned sequences $S'_1, \ldots, S'_n$ (that is, S'_i is the alignment of the i-th sequence in the MSA, which is S_i with potentially additional gaps representing indels).

Then, given a list of n sequences $S_1, \ldots, S_n$ (in *seq_lst* parameter):

1) Create a distance matrix, D, for $S_1, \ldots, S_n$ using the function given in *dist_func* parameter.
2) Construct a tree for D using the UPGMA algorithm. Denote the resulted tree by T.
3) For each split in T initialize a counter and set it to zero (see below how to represent it using a dictionary).
4) For i = 1:100 do:
    - Randomly select 50% of the columns in the MSA (i.e. sample without replacement).
    - Construct n new sequences $\tilde{S}_1, \ldots, \tilde{S}_n$. Construct $\tilde{S}_i$ by considering only a subset of the columns of S'_i - the columns selected in the previous step. Eventually, remove all gaps.
    - Create a distance matrix, $\widetilde{D}$, on $\tilde{S}_1, \ldots, \tilde{S}_n$.
    - Construct a tree $\tilde{T}_i$ by running the UPGMA algorithm on $\widetilde{D}$.
    - For each split in T, raise its counter by one if it exists in $\tilde{T}_i$. (See below how to determine if split exists in a tree).
5) Return a dictionary containing the counters of the splits of T.

The keys in the dictionary are tuples representing the splits in the tree T. Values are the counters of each split. For example, for three sequences, X, Y and Z, if the UPGMA tree is ((X,Y),Z) then the keys in the dictionary will be: ((X,Y),Z) and (X,Y).

Report the counter of each split from the original tree. Apply the evaluation procedure twice: first, using the pairwise alignment approach for constructing the distance matrix, and second, using the k-mers approach for constructing the distance matrix.

Briefly compare the performance of the two approaches for calculating the distance matrix. Which approach was faster and which approach provided more robust results (i.e. higher confidence in the splits of the original tree)? Explain.

i. Suggest a way to improve the evaluation of the tree when using k-mers based distance matrix. Elaborate and explain the potential problems with the approach applied in the previous section.