COMPSCI 260, Introduction to Computational Genomics **28 October 2022**

Problem Set 5

Due: 5pm on 11 November 2022

Problem 1: Love thy neighbor as thyself (15 points)

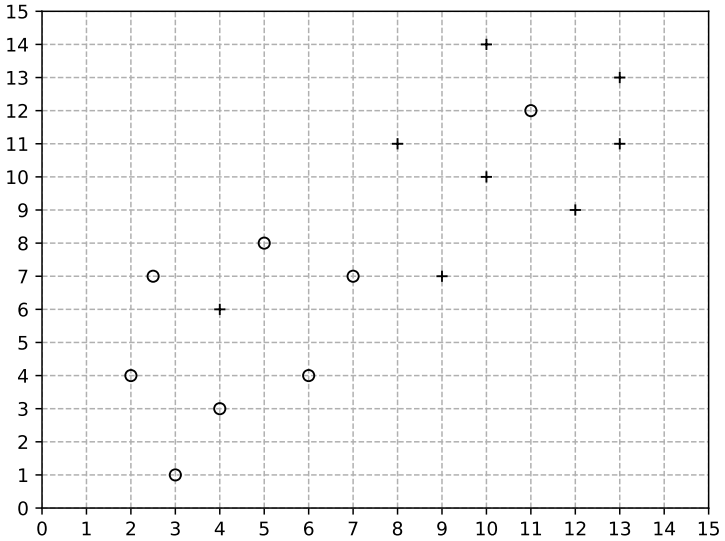
List of files to submit:

1. `README.problem1.[txt/pdf]`
2. `KNearestNeighbor.py`

Plan

One simple but powerful way of classifying is by way of the non-parametric classification strategy called 'k-nearest-neighbor'. In this approach, unlabeled samples are assigned class labels based on the labels of the k labeled samples that are closest, taking a majority vote (for this reason, when there are two classes, we always choose k to be odd so that there won't be ties).

Consider the 16 labeled, two-dimensional training samples presented in the Cartesian plot below (on the next page). Eight samples are labeled '+' and eight samples are labeled '$\circ$'. For example, these could represent the gene expression levels of two key biomarker genes (plotted on the x and y axes) from 16 patients, half of whom responded to a particular treatment ('+', the responders) and half of whom did not ('$\circ$', the non-responders).



Imagine five new patients walk into your clinic, and you measure the expression levels of the two biomarker genes for each patient. When the results come back from the lab, this is what you see:

Name	Gene expression profile	Class when $k = 1$	Class when $k = 3$
Alice	(2, 3)		
Bobby	(10, 12)		
Cindy	(8, 10)		
Donny	(4, 7)		
Ellen	(8, 7)		

a) What prognosis are you going to give each patient in terms of their response to the treatment? Consider both $k = 1$ and $k = 3$. Although k-nearest-neighbor just assigns a simple label to each case, comment on how confident you are in your prognosis in each case.

Develop

Let's scale this up a little bit. Imagine that you now have 10 biomarker genes (so the points are in 10-dimensional space, or $\mathbb{R}^{10}$; don't worry: we won't ask you to draw this). Imagine you also now have a database of gene expression values for all 10 biomarker genes for 1000 previous patients who had the good fortune of responding to your treatment (class R) and 1000 previous patients who ended up being non-responsive (class N). This data is all contained in the file `gene_expression_training_set.txt`.

This time, ten new patients walk into your clinic, and you measure the expression levels of the 10 biomarker genes for each patient. When their results come back from the lab, they are contained in the file `gene_expression_test_set.txt`.

b) Building on the skeleton code we provide in `KNearestNeighbor.py`, write Python code that will offer a prognosis for your ten new patients using $k = 5$.

For a further extra challenge, you can write your code to take k as a parameter and see how sensitive the result is to your choice of k (you should require k to be odd). If you do this, one benefit is that you can confirm your answers to part (a) pretty quickly. For a different extra challenge, you can have your code report not only the final class label assigned to your ten new patients (R or N), but also the distances and labels of the $k = 5$ closest patients in the database: this will help you assess the confidence of your prognoses.

Apply

c) **Report** the predicted prognosis class for each of your ten new patients using $k = 5$.

Problem 2: Ultrametricity and additivity (15 points)

List of files to submit:

1. `README.problem2.[txt/pdf]`
2. `UltrametricAdditive.py`
3. `any image files related to the trees you construct on paper`

Plan

In class we discussed two algorithms for building phylogenetic trees: UPGMA (unweighted pair group method using arithmetic averages) and NJ (neighbor joining). Both algorithms use distances between sequences to reconstruct the phylogenetic trees. UPGMA is guaranteed to reconstruct the correct (rooted) tree if the distance metric is *ultrametric*, while NJ is guaranteed to reconstruct the correct (unrooted) tree if the distance metric is *additive*. Here, by 'correct', we mean that distances between sequences as inferred from the resulting tree's branch lengths are consistent with those in the original matrix.

Given four sequences X_1, X_2, X_3, and X_4, Table 1 shows the distance between each pair of sequences. Let's call this distance metric D_1. Similarly, for the five sequences Y_1, Y_2, Y_3, Y_4, and Y_5, Table 2 shows a distance metric D_2.

D_1	X_1	X_2	X_3	X_4
X_1	0	0.3	0.7	0.7
X_2		0	0.6	0.6
X_3			0	0.6
X_4				0

Table 1

D_2	Y_1	Y_2	Y_3	Y_4	Y_5
Y_1	0	0.9	0.4	0.6	0.9
Y_2		0	0.9	0.9	0.4
Y_3			0	0.6	0.9
Y_4				0	0.9
Y_5					0

Table 2

Develop

a) Write the necessary code for two Python functions, `is_ultrametric()` and `is_additive()`, in the file `UltrametricAdditive.py`. These two functions will receive as input the pairwise distances for a set of sequences and then verify whether the given distances satisfy the ultrametricity or additivity criterion, respectively.

Do *not* use a matrix structure to represent the pairwise distances for a set of sequences; instead use the following dictionary representation, as illustrated in `UltrametricAdditive.py`:

dictionary of distances = {"1,2": $d_{1,2}$, "1,3": $d_{1,3}$, "1,4": $d_{1,4}$,
"2,3": $d_{2,3}$, "2,4": $d_{2,4}$,
"3,4": $d_{3,4}$}

where $d_{i,j}$ is the distance between sequences i and j. We use this dictionary structure because it will be easier to modify when we merge clusters or nodes in our tree-building algorithms (which will require us to remove certain distances and add new ones).

Since you will be working with real protein sequences, it is rarely the case that two distances (or two sums of distances) will be exactly equal. So in your two functions `is_ultrametric()` and `is_additive()`, instead of checking the equality of two distances using an expression like `dist["i,j"] == dist["j,k"]`, use `is_almost_equal(dist["i,j"],dist["j,k"])`. The helper function `is_almost_equal()` is defined in the Python file `UltrametricAdditive.py`, and simply verifies whether the difference between its two parameters is negligible enough that the parameters can be considered 'equal'.

Note that each of these three functions has a `threshold` parameter to indicate how close two distances need to be in order to be considered equal. You should use the threshold values we have provided in the skeleton code. However, you will have to pass the `threshold` argument value from `is_ultrametric()` and `is_additive()` to your calls of `is_almost_equal()`.

Apply

b) Use your Python functions to check if the distance metrics D_1 and D_2 given above satisfy the ultrametricity and/or additivity criteria and then reconstruct the appropriate tree by hand. Specifically, for each

of the two distance metrics D_1 and D_2:

1. Test the ultrametricity criterion.
   - If one or both of them is not ultrametric, give an example of a triplet that fails to satisfy the ultrametricity criterion.
   - If one or both of them is ultrametric, build the (rooted) phylogenetic tree that would be reconstructed by UPGMA by hand (i.e., on paper; submit a photo or scan, and be sure to show all your work in what you submit, not just the final tree).
2. Test the additivity criterion.
   - If one or both of them is not additive, give an example of a quadruplet that fails to satisfy the additivity criterion.
   - If one or both of them is additive but not ultrametric, build the (unrooted) phylogenetic tree that would be reconstructed by NJ by hand (i.e., on paper; submit a photo or scan, and be sure to show all your work in what you submit, not just the final tree).

c) Recall the ATP synthase we considered in the previous problem sets. The table below contains, roughly, the distance metric for those four protein sequences plus two additional sequences: one for the ATP synthase in *Saccharomyces cerevisiae* (YEAST) and one for the ATP synthase in *Schizosaccharomyces pombe* (SCHPO), both of which are fungi.

	HUMAN	ECOLI	BACSU	MOUSE	YEAST	SCHPO
HUMAN	0	0.5	0.5	0.1	0.4	0.4
ECOLI		0	0.3	0.5	0.5	0.5
BACSU			0	0.5	0.5	0.5
MOUSE				0	0.4	0.4
YEAST					0	0.3
SCHPO						0

Using the code you've written in `UltrametricAdditive.py`, determine whether this distance metric satisfies the ultrametricity and additivity criteria. Based on the results of your check, select an appropriate tree construction algorithm and build the phylogenetic tree that would be reconstructed by that algorithm by hand (i.e., on paper; submit a photo or scan, and be sure to show all your work in what you submit, not just the final tree).

Reflect

d) *What do you observe?* For the four ATP synthase proteins from the previous problem sets, compare the relationships among them as described by the reconstructed tree with the relationships you previously inferred based on their pairwise alignments. Additionally, how do the two fungal species fit in with the two mammals and the two bacteria?

Problem 3: Hall monitor (30 points)

List of files to submit:

1. `README.problem3.[txt/pdf]`
2. `BuildTree.py`

3. `StudentTree.pdf`
4. `FLAG.txt`

⋆ **Step 1:** Under surveillance

Plan

As a budding computational biologist, you volunteered to do research with Duke's COVID-19 surveillance team. As you know, all Duke students were tested at regular intervals throughout the early years of the pandemic with a PCR test that detects small traces of SARS-CoV-2 genome in a sample. However, you learn from your boss that your team is conducting a trial of whole-genome viral sequencing to test if it can aid Duke's COVID mitigation efforts. To this end, she has fully sequenced the genomes of virus isolates from seven students living in the same residence hall on East Campus who reported mild respiratory symptoms nearly a week after arriving on campus at the start of the fall semester.

Ding! Your boss receives an email with fresh results from Duke's Sequencing Core and immediately forwards it on to you. Staff at the Sequencing Core must have taken COMPSCI 260 in the past because they have already assembled the myriad viral reads together into a full-length genome for each sample. Not only that, they also took the extra step of determining the spike protein amino acid sequence from each viral sample —how helpful! Now, based on the spike protein amino acid sequences, it's up to you to figure out what virus each of these students has, how related those viruses are to one another, and whether any pair of students might have transmitted a virus to one another or whether they were all infected independently.

The email from the sequencing core contains two files: `students.fasta` has the spike protein sequences associated with the viral genomes isolated from each of the seven students, while `students.aligned.fasta` contains a multiple alignment of those amino acid sequences. Note that a multiple alignment is (as it suggests) simply an alignment of multiple sequences (in this case, 7). As with pairwise alignment, it is still a long list of 'columns', but since there are 7 sequences being aligned, each column now will be 7 characters tall instead of only 2 characters tall. Within each column, each of those 7 characters is either a character from a sequence or it's a gap character. A single column is permitted to have multiple gap characters, but as with pairwise alignment, it can never be *entirely* gap characters.

A corollary of the preceding description is that if you extract any two sequences from a multiple sequence alignment and consider them by themselves, it's now possible for there to be a column with two gap characters, which wouldn't have been possible under pairwise alignment. But what we can do in that case is simply ignore any column with two gap characters in it and consider only the remaining columns.

Apply

a) Compute the pairwise distances for the seven amino acid sequences, and store them in a dictionary data structure similar to the one used in Problem 2. Your boss unearths a Python file called `BuildTree.py` she has written that contains some code for doing this within a `compute_dist()` function. **Report** your computed pairwise distances in a table easily read by a human using the provided `print_dist()` function. You should output this table when you run the `build_tree()` function, and should copy your table to your `README`.

b) Inspect the distances in your table from (a). Can you already spot any students with identical spike protein sequences? Can you be sure based solely on the table? (*Hint*: How is your ability to interpret an identical sequence affected by the way your boss's `compute_dist()` function is coded?)

c) Use the Python functions `is_ultrametric()` and `is_additive()` from Problem 2 to verify whether the distance metric computed above satisfies the ultrametricity and/or additivity criteria. Are the distances

ultrametric? Are they additive? Can we apply the UPGMA and/or NJ algorithms? Why or why not?

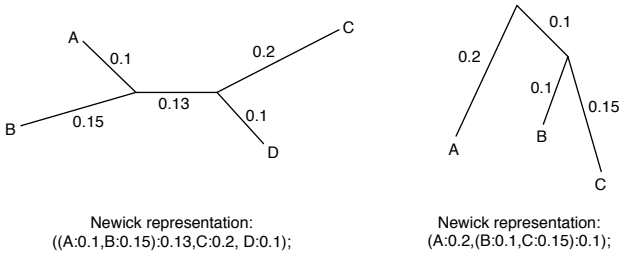
⋆ **Step 2:** A Tree Grows in . . . NJ?

Develop

d) Digging around further in `BuildTree.py`, you notice that it seems to contain some skeleton code for implementing the NJ algorithm. Unfortunately your boss never finished the implementation, so the program is missing certain important lines that actually make it work. In the section marked by the comments `# --- Begin your code ---` and `# --- End your code ---`, you will need to fill in the lines as guided by the comments, to make this program work.

One good thing is that the program already contains code to output the reconstructed tree as a string in something called the Newick format. This output will be required in the next part of the problem.

The Newick format is described in detail at http://evolution.genetics.washington.edu/phylip/newicktree.html. Briefly, it is a computer-readable format for representing a tree as a string by using nested parentheses. Each *leaf* node is represented by its name (*i.e.*, a string of characters except blanks, colons, semicolons, parentheses, and square brackets). Each *internal* node is represented by a pair of matched parentheses, with its child nodes contained between the two parentheses, separated by commas. The length of a branch connecting a node to its parent can be specified by putting a colon after the node, followed by a real number representing the length of the branch. The terminating semicolon specifies where the tree representation ends. Please see the examples below.



Note that for an unrooted tree, the Newick representation is not unique. The unrooted tree on the left above could equally well be represented in many other ways, including: ((A:0.1,B:0.15):0.13,(C:0.2,D:0.1):0);

Apply

e) Once your code for implementing the NJ algorithm is working correctly, use it to reconstruct the unrooted tree representing the distances between the viruses isolated from the 7 students. **Report** the reconstructed tree in Newick format, and be sure to copy the result into your `README`.

f) As soon as you finish reconstructing the phylogenetic tree, you call your boss to show her your findings. She looks at the phylogenetic tree in the Newick format, but seems perplexed. She does not understand what the set of parentheses, numbers, and letters means. Luckily, you remember learning in your Computational Genomics class about programs that will visualize these trees. With a quick Google search, you find a program through the Interactive Tree of Life (iTOL) that draws tree diagrams and accepts Newick formatted inputs.

Use the iTOL tool to visualize the phylogenetic tree built in part (e). By default, iTOL will display a rooted tree; you may need to adjust this setting depending on your answer to part 2(b). Experiment with various levels of zoom and settings in the provided control panel until you can get a clear sense of the entire topology. Under "Export," you can save your tree in a PDF file and study it (you should also submit it, with the filename `StudentTree.pdf`). Make sure the branch labels are visible in your exported file.

g) You notice that a subset of these viruses are so close to one another in the tree that it's a bit hard to visualize their relationships. You find yourself curious about these viruses and wonder if you recognize any of them. Using the unaligned sequences in `students.fasta`, run each of the 7 sequences through protein BLAST to identify the closest hit. (To stay within BLAST's memory limit for individual jobs, it's best to submit one student at a time.) For each student, provide the BLAST accession number for the closest match, along with the virus name and the common name (sort first by max score then if necessary, by percent identity). Provide your results in your `README` in a human-readable table like the one below.

Student	Accession	Virus name	Common name of virus

Reflect

h) Given your results from BLAST, what can you say about the likely illness for each student? How does that correspond to what you see in the tree?

Apply

i) Fill in code as prompted in the function `summarize_alignment()`, and use its output to **report** the differences between the two SARS-CoV-2 genome sequences that are most similar, the two SARS-CoV-2 genome sequences that are most different, and the two genome sequences that are most different overall (you can use your distance matrix to establish these three pairs of sequences). *How would you interpret the similarities and the differences in these three pairs of genome sequences?*

j) What do these results imply about the spread of SARS-CoV-2 between students?

⋆ **Step 4:** A Spiky Subject

k) Inspecting the cluster of SARS-CoV-2 sequences on your tree, you wonder if the scope of these genetic changes might potentially include those of widely reported variant strains of the COVID-19 coronavirus. These strains have evolved mutations that increase transmission, worsen symptoms, or reduce the efficacy of some vaccines. Many of their mutations occur within the gene that encodes the "spike protein", the surface glycoprotein that allows the virus to enter human cells.

Suddenly, you have the urge to compare these SARS-Cov-2 variant spike proteins to each other to see if you can figure out which variant corresponds to which spike protein. With some insider information, you happen to know that Student 1 has an "original" version of SARS-CoV-2 spike protein. Thus for two pairs of COVID-affected students, (Student 1, Student 3) and (Student 1, Student 7) perform a global alignment of the two protein sequences, using either:

- Your very own affine global aligner from Problem Set 4, with $g = 1$, $h = 11$, and the BLOSUM62 amino acid substitution matrix, or
- NCBI's implementation of Needleman-Wunsch global alignment between two protein sequences (be sure to set the parameters the same way)

In your `README`, compare the results of your alignments to this map[1] of the different lineage of variants. Based on mutations described in the NYT article, can you identify which SARS-CoV-2 variants Student 3 and Student 7 had? Specify which variant each of these two students had and what mutations lead you to these conclusions in your `README`.

[1] If you have trouble accessing the New York Times link, we included a file called `nyt_article.pdf` for you in the zipfile.