COMPSCI 260, Introduction to Computational Genomics **25 October 2019**

Problem Set 5

Due: 5pm on 8 November 2019

Problem 1: Love thy neighbor as thyself (25 points)

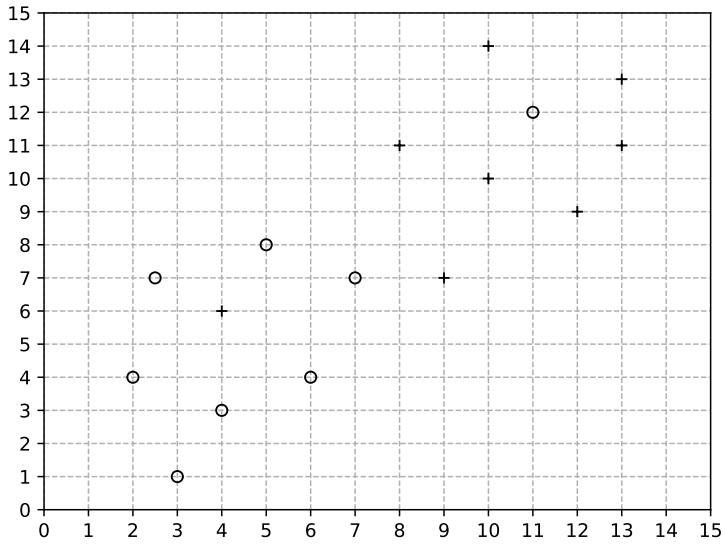
List of files to submit

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. Eight samples are labeled '+' and eight samples are labeled '○'. 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 ('○', 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 also that you now have a database of gene expression values for all 10 biomarker genes for 1000 patients who responded to treatment (class R) and 1000 patients who were 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 ten biomarker genes for each patient. When the 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 to give a prognosis for the ten new patients using $k = 5$.

Extra Challenge: For a little extra challenge, you may wish to 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 report not only the final class label assigned to the new patients (R or N), but also the distances and labels of the $k = 5$ closest labeled patients in the database: this will help you assess the confidence of your prognoses.

Check

c) **Report** the prognosis class for each of the 10 patients.

Problem 2: Ultrametricity and additivity (25 points)

List of files to submit

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

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 an *ultrametric*, while NJ is guaranteed to reconstruct the correct (unrooted) tree if the distance metric is *additive*.

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	0.9
X_2		0	0.6	0.8
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	0.8	0.4	0.6	0.8
Y_2		0	0.8	0.8	0.4
Y_3			0	0.6	0.8
Y_4				0	0.8
Y_5					0

Table 2

Develop

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

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 the two functions `is_ultrametric()` and `is_additive()`, instead of comparing distances using `dist["i,j"] == dist["j,k"]`, use `is_almost_equal(dist["i,j"],dist["j,k"])`. The function `is_almost_equal()` is defined in the Python file `UltrametricAdditive.py`, and simply verifies whether the difference between the two parameters is negligible enough that the parameters can be considered equal.

Note that each of these three functions also have a `threshold` parameter to indicate how close two distances can be 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 calls of `is_almost_equal()`.

Do *not* use a matrix structure to represent the distances between two sequences; instead use the following dictionary representation, as illustrated in the script `UltrametricAdditive.py`:

dist = {"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 the i^{th} and j^{th} sequences. We use this dictionary structure because it will be easier to modify when we merge nodes in our tree building algorithms and therefore need to remove certain distances and add new ones.

Check

b) Use your Python functions to check if the given distance metrics satisfy the ultrametricity and the additivity criteria. 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 does not 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).
2. Test the additivity criterion.
   - If one or both of them is not additive, give an example of a quadruplet that does not 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).

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

	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
BACCE			0	0.5	0.5	0.5
MOUSE				0	0.4	0.4
YEAST					0	0.3
SCHPO						0

c) 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.

Reflect

d) *What do you observe?* For the four proteins from the previous problem set, compare the relationships between them described by the reconstructed tree with the relationships you inferred based on their pairwise alignments. Additionally, how do the two fungal species fit in with the two animal species and the two bacterial species?

Problem 3: The origin of MERS (35 points)

List of files to submit

1. `README.problem3.[txt/pdf]`
2. `BuildTree.py`
3. `UltrametricAdditive.py`
4. `SpikeTree.pdf`

⋆ **Step 1:** Extracting protein relationships

Plan

It's late 2019 and you've been sent to Qatar to monitor preparations for the 2022 World Cup. After a long day of soul-crushing FIFA politics, you are enjoying a well-deserved rest at your hotel. You're relaxing in the spa with cucumbers on your eyes when you are jolted into consciousness by a sharp knock on the door. Hotel staff are streaming into the room and—owing to your reputation as an eminent scientist—are asking urgently for you to weigh in on the likely origin of the newly-discovered MERS virus: how did this particular deadly viral genome sequence arise? Was it from a mutation in an existing human coronavirus, one like SARS? Or did an animal coronavirus make the jump into a human host?

Armed with nothing but your wits, endless fluffy spa towels, and a portable Python interpreter, you set to work. You request a set of orthologous 'spike' protein sequences from a range of coronaviruses isolated from different animal species. Within seconds, the hotel bellhop waltzes in with the `CoV.fasta` file on a silver platter; you tell yourself this guy deserves a big tip.

a) The table below shows the Genbank IDs of all the spike proteins, but in his hurry, the bellhop neglected to notate the virus species from which each protein came. Fill in the table with the names of the corresponding viruses, as reported in Genbank in the "protein" database. You should also note the animal host for each species of virus.

Index	Protein ID	Virus name	Animal host
1	AAK83356.1		
2	AAL57308.1		
3	AAL80031	Porcine hemagglutinating encephalomyelitis virus (HEV)	
4	AAP41037.1	SARS coronavirus Tor2	human
5	AAR01015.1		
6	AAV49723		
7	ABD75513.1		
8	AFS88936.1		
9	ATW75480.1		
10	AUM60024.1		
11	CAA01637.1		
12	CAA71056.1		
13	YP_209233.1	Murine hepatitis virus strain JHM	

Develop

Just as you finish filling in the table with the names of the viruses, the bellhop rushes back in to bring you another file, `CoV.aligned.fasta`, which contains a multiple alignment of the 13 spike proteins (apparently, the bellhop knows how to use ClustalW). You give him another big tip for his efforts.

b) Write a Python function called `compute_distance` that takes as input the indices of two proteins, as given in the table above, and returns the distance between the two proteins, computed from the multiple alignment the bellhop provided. Here, you decide to define the distance between two aligned proteins to be the number of mismatches in the alignment, divided by the total number of matches and mismatches (*i.e.*, all positions where there are no gaps in either of the sequences): it's quick-and-dirty, but time is of the essence here. Include this function in the `BuildTree.py` file.

Check

c) Compute the pairwise distances for the 13 proteins, and store them in a dictionary data structure similar to the one used in Problem 2. **Report** your computed pairwise distances in a format that is usefully read by a human.

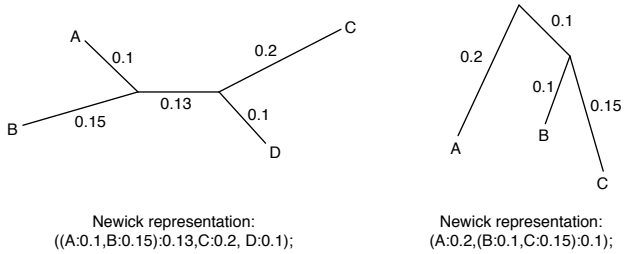
⋆ **Step 2:** NJ tree construction

Develop

d) Luckily you find in an obscure folder on your computer a Python file called `BuildTree.py` which seems to contain skeleton code for implementing the NJ algorithm. You can definitely use it to reconstruct the (unrooted) phylogenetic tree of the 13 proteins. Unfortunately 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 ---` fill in the lines as instructed by the comments, to make this program work.

The program already contains code to output the reconstructed tree in 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, the Newick format is a computer-readable format for representing trees 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 the children nodes 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 can also be represented as: ((A:0.1,B:0.15):0.13,(C:0.2,D:0.1):0);

Check

e) **Report** the reconstructed tree in Newick format, and be sure to copy the result into your `README`.

As soon as you finish reconstructing the phylogenetic tree of the spike proteins, you call the hotel staff to show them your findings. They look at the phylogenetic tree in the Newick format, but seem perplexed. They do not understand what the set of parentheses, numbers, and letters means. Luckily, you remember the "Phylip drawtree" program that you used in your Computational Genomics class. You quickly Google the name of the program and find the web server: http://www.sacs.ucsf.edu/cgi-bin/drawtree.py.

f) Use the online tool "Phylip drawtree" to visualize the phylogenetic tree built in part (e). The tool will output a PostScript file which you can download and open and convert to a PDF if you have Adobe

Acrobat or Mac Preview or another related program. If you don't have a program capable of opening and converting a `.ps` file, you can also make use of a web server operating as a front end for Ghostscript (*e.g.*, `http://ps2pdf.com/convert-ps-to-pdf`) to convert it to a PDF. Either way, save the tree in a PDF file named `SpikeTree.pdf`, and study it (you should also submit it).

Reflect

g) *What are you going to tell the hotel staff? How do you think the MERS virus arose?*

h) Looking back at the list of spike proteins you note that SARS is one of the spike proteins. Interestingly, your phylogenetic tree will also give you information about how SARS originated. *How do you think the SARS virus arose? Is there any similarity in the symptoms of these two diseases (SARS and MERS)? According to the phylogenetic tree, is SARS the other human coronavirus that MERS is most similar to?*

Problem 4: Don't worry yourself sick (15 points)

List of files to submit

1. `README.problem4.[txt/pdf]`

A friend of yours, Murray Worry, is something of a hypochondriac. One night, while watching the news, Murray learns about a rare disease estimated to occur in only one of every 50,000 people. Despite the odds, he insists you accompany him to get a proper medical test in order to rule it out. After an excruciating wait at the clinic, you and Murray are finally welcomed back to see Dr. Mel Practus. In an effort to reassure him, Dr. Mel indulges Murray and orders the test because it is known to be 99.9 percent accurate (specifically, if one has the condition, there's a 99.9% chance that the test detects it, and if one doesn't have the condition, there's a 99.9% chance that the test does not detect it). After a quick swab from the inside of Murray's cheek, Dr. Mel excuses herself from the room to carry out the diagnostic test. Moments later she returns and informs you both of the bad news: the test has come back positive for this extraordinarily rare disease.

a) Might there be some quantitative way you can comfort your friend? Think carefully about how likely it is that Murray actually has the disease, given that he tested positive.

b) How do you explain your answer to part (a) in light of the fact that the test is 99.9% accurate? What would have to be true about the diagnostic test in order to observe a more reasonable false positive rate?

c) What if it turns out that development of this rare disease were the result of very specific environmental factors? Does your conclusion from part (a) still hold? Why or why not?