BNFO 601: Integrated Bioinformatics (3 November 2017)
EXAM 2

The primary purpose of this exam is to serve as an educational tool. With this in mind, you should not be surprised if some of the problems go beyond your present abilities and try to connect things that may not yet have been connected. Nonetheless, most of the elements have been drawn from the modules. If you ask yourself, "*Have I seen something like this before*?" the answer is almost always "Why yes, I have".

RULES OF THE GAME

- **Available resources:** This is an open book exam. It is an open notes exam. It is an open web exam. Most important, it is an open brain exam.
- **Unavailable resources:** Needless to say, it is not an open people exam.
- ***Major exception*:** We are an open resource for this exam. You may consult with Paul.
- **Consultations may not be totally free:** You may be charged some fraction of the points allotted to a question for any advice related to that question (you'll be advised before any charge is levied). Charges are highest for general questions (like "*I don't know what to do with this question*") and lowest (or perhaps nonexistent) for specific questions (like "*I don't know how Python asks for input from the user*"). You would be well advised to exhaust your own resources before asking for advice, both to save unnecessary charges and to get clear in your mind what it is you actually need.
    - **Contact info:** Paul: pfawcett@vcu.edu, 827-0975. Paul's office is now relocated to Sanger Hall, Room B1-011; If I am unavailable or just too surly to be of use to you, you may in a real pinch also contact Jeff Elhai: ElhaiJ@VCU.Edu, 828-0794.
    - **What to submit:** **PAY ATTENTION TO THE FOLLOWING!!**

- Electronic submission is *required*. The preferred format is one .zip file containing all of your answer files! When sending attached files, ***be sure that the filenames begins with your name*** so I don't get a mailbox full of files named "Exam1".
- When asked to provide a program, submit both the program file and, if possible, at least a sample of the output. The program files should be named as indicated for each question. It's generally a good idea to submit material that shows your thought processes. **Comments** in computer programs are a definite ***"Good Thing"*** that will help determine if you were on the right track... **Items highlighted in red**

correspond to files that will be a mandatory part of your submission

- **IMPORTANT!** Your submitted folder should include ALL of the files I need for your programs to just run (even if I provided the files). Similarly, all of the file paths that you use in your programs should be relative to the folder name you submit (*i.e.* do NOT use hard coded paths). I *don't* want to mess around copying necessary files into your folder and adjusting path names in order to get your programs to run correctly -- it's a huge PITA.

- **When to submit:** Everything *must* be submitted by **10:59 AM, Wednesday, 8 November,** and *not later*. We will be starting a new module on Wednesday morning, and I want your full attention on the new material!

Sign and print your name below to indicate that you have neither received nor given aid during the preparation of your exam answers and agree to abide by the VCU Honor policy (instructor aid excepted). No collaborations! Infractions of this policy are treated with utmost seriousness:

Aravind Veerappan

The Questions

1. Position-Specific Scoring Matrices & Information Theory

A. Just as it was possible to modify the Smith-Waterman or Blast programs to accommodate protein sequences, it is possible to construct protein-based PSSM. What would be the **maximum amount of *information*** that could be theoretically encoded in one column of a *protein* PSSM? **Explain how** you derived this number.

B. What would be the **underlying assumption** about the likelihood of observing a given amino acid symbol in this instance?

C. Which of the following statements are **true or false. Explain** each answer in a short sentence:

i) Decrease in uncertainty leads to a decrease of information content.

ii) Maximum uncertainty occurs when nucleotide symbols are unequally probable.

iii) Information is equal to the uncertainty existing *before* observing a symbol.

iv) Uncertainty, also known as entropy, always increases.

v) Information gained always corresponds to the loss of uncertainty that occurs after an observation has been made.

Preliminaries for Question 2. In class we discussed the modifications that are necessary to convert a nucleotide alignment algorithm into a protein alignment algorithm. An essential change was dispensing with the straightforward match/mismatch score or penalty and using instead an alignment score that reflects in some sense how likely a given amino acid is to mutate to any other amino acid. In the class notes and review today, we detailed the general methods by which both the PAM matrices of Dayhoff, and the BLOSUM matrices of Henikoff and Henikoff are calculated. Here, I provide you with a Python program called **`PAM.py`** (and an accompanying table of raw counts **`PAM1.txt`**) which you will find useful – it calculates a PAM matrix of the desired value of ”N” using the following basic steps, which I encourage you to fully understand by a careful dissection of the code:

1. The late Margaret Dayhoff collected pairs of very similar protein sequences that differed from each other in only about one amino acid in every one hundred residues.

2. Analyze by counting how many times a given amino acid was replaced by other amino acids to generate a so-called “Mutation probability matrix”. This is what is presented as the PAM1 Table 2 in the protein alignment and scoring notes.

3. PAM1 probabilities are converted to PAMn probabilities by multiplying the PAM1 matrix *by itself* n times. REMEMBER!!! Matrix multiplication doesn't work the way that you intuitively might guess. If you have forgotten your high school math (who hasn’t!), see the following URL for a tutorial:

http://www.zweigmedia.com/RealWorld/tutorialsf1/frames3_2.html

4. The frequency of each amino acid-amino acid transition is then rendered as an odds ratio according to the following procedure: **Note that to simplify the calculations, the method we here describe varies slightly in presentation from the method described in the notes….in practice there are a number of minor variations from author to author.**

What we do is calculate new values $\mathbf{R_{ij}}$ by taking each value $\mathbf{M_{ij}}$ in our array, and dividing it by a number $\mathbf{f_i}$ where $\mathbf{f_i}$ corresponds to the *relative* probability that a particular amino acid will undergo mutation. This captures the notion of how "exposed" to mutation a particular amino acid is. The `PAM.py` program embodies this $\mathbf{f_i}$ information in the form of a dict, `self.aa_frequencies`, which is ultimately derived (yes, well, I knicked it straight from there, OK?) from Table 22 of the *Atlas of Protein Sequence and Structure*, Suppl 3, 1978, M.O. Dayhoff, ed. National Biomedical Research Foundation, 1979. Note that these relative probabilities sum to 1.

5. From there, a log odd ratio is calculated where lod = $\log_{10}$(odds ratio).

6. Finally, the log odds ratios (the table of $\mathbf{R_{ij's}}$) are multiplied by 10. There is one more catch here. Note that our PAM1 table in the notes and in the file I provided (**`PAM1.txt`**) is not quite symmetric!! For instance $\mathbf{M_{ala->asp}}$ is not equal to $\mathbf{M_{asp->ala}}$, or more generally $\mathbf{M_{ij} != M_{ji}}$. Therefore, when creating the final dict we *average* the $\mathbf{R_{ij}}$**'s** and the $\mathbf{R_{ji}}$**'s** (So that, for instance, an ASP->ALA mutation is treated the same way as an ALA->ASP mutation).

7. Finally, the resulting numbers are truncated so that only the integer portion remains (*i.e.* 2.62 would become 26)

Question 2. Using PAM.py and PAM1.txt as a starting point, modify your Blast alignment program so that it actually works using a **PAM120** matrix we calculate (It shouldn't be necessary to actually copy and paste any text from PAM.py into your program – if you are clever you can simply import the necessary functionality from PAM.py and call whatever methods you need to calculate an appropriate PAM matrix.

The biggest difference is that normal BLASTN only has to check whether or not there is an exact match (*i.e.* does "A" == "A") and scores accordingly in a yes/no manner. But with protein comparisons, amino acids might be considered an exact, a good, a poor, or a downright bad substitution depending on which pair of

amino acids happens to be under consideration. We therefore need to *lookup* the score from the PAM dict we have calculated.

Don't forget also to implement the neighborhood threshold score to decide whether a seed is "good enough" to begin calculating forward and backward extension tables. To do this successfully, you may need to rethink the way that the existing program attempts to match query words with target words. Naturally, a first step to doing this will be identifying the part of the existing code that does this job. As always, pay attention to the comments!

Submit your program as "`YourName_BLAST_Prot.py`". I have provided are two files, **`quoll.txt`** and **`numbat.txt`** representing putatively orthologous imaginary protein sequences from two closely related (and dastardly cute) Australian marsupials. **Show the output** of your program when run with these as query and search sequences respectively. Do you buy the argument that these two proteins may be orthologs? **Explain.**

Note that for the purposes of the exam, you are encouraged to play around with different values for each of the key parameters, but stick with gap opening and extension scores that are the same for each, as these are not properly differentiated in the existing code base. I recommend that you experiment with various word lengths and neighbourhood thresholds. 3 or 4 might be a good starting place for word length. For word length three, starting threshold values starting around 10-15 might be about right? Experiment. **Describe the effect on your output as you vary these!**

Question 3. Modifications to **`MyCluster.py`**

3a. If you haven't done so already in class, modify the clustering program **`MyCluster.py`** so that it is fully functional (*i.e.* calculates a Pearson correlation coefficient instead of Euclidian distance, correctly implements the depth first search to untangles the nodes, and produces `.cdt` and `.gtr` files that can be meaningfully loaded and used in the `Java TreeView` software package). This is just the starting point for the rest of the questions, so take the time to get this right first!!

3b. With a properly running version of **`MyCluster.py`** as a starting point, change the program yet again so that it implements BOTH of “Single linkage clustering” and “Complete linkage clustering”. If it makes your life simpler, you can completely ditch the simple average clustering that occurs by default. But the user should be able specify whether it is to perform single or complete clustering by passing an appropriate argument to initializer of the `Tree` class.

Both of these approaches will require that each node will have an attribute that somehow *keeps track* of all the genes that it represents. Implementing linkage clustering variants is a bit more challenging than just a weighted averaging procedure. For linkage clustering we need to keep track of not only how many genes a node represents, but also the *identity* of the genes that a node represents. Think deeply about the fact that the single and complete linkage procedure works in a fundamentally different way than the average linkage approach! For one thing unlike average clustering, single linkage does NOT require us to make up new data for each non-gene node that is created. This is because the distance between any two nodes is given by what amounts to a *proxy*…in this case the nature of the proxy depends on whether we are doing single or complete clustering. In single linkage clustering, the distance between the closest pair of actual genes in the respective nodes serves as the proxy distance for the nodes as a whole (refer to the graphic in the powerpoint).

Think of it this way… say you wish to know the distance between NODE3 (which happens to represent GENE23, GENE1, and GENE4) and NODE5 (which happens to represent GENE12 and GENE2), you would have to consider the similarity of all of the following pairs: ('GENE23', 'GENE12'), ('GENE23', 'GENE2'), ('GENE1', 'GENE12'), ('GENE1', 'GENE2'), ('GENE4', 'GENE12'), ('GENE4', 'GENE2'). *One* of these pairs will have the highest correlation (smallest distance) and will therefore be the proxy distance that we use to represent the distance between NODE3 and NODE5 under single linkage clustering (complete linkage would be the pair with the largest distance).

But here's something critically important! Note that when we finally decide to make a new node (***always*** based on the node pair with the smallest node-to-node distance regardless of which method is used!), the new node will represent ALL of the genes from the underlying nodes that we joined! For instance, in our example, if we happened to create some new NODE6, it would represent GENE23, GENE1, GENE4, GENE12, and GENE4.

A couple of final giveaways – if you are somehow maintaining a list of genes that a node represents, reflect first on the fact that genes form a basis case…they are simply nodes that represent only themselves (*i.e.* they are leaves). It turns out there is already a mere snippet or two of code in the program that is hinting at this direction, although nothing is implemented correctly or fully. Can you figure out the bits that I'm referring to? Anyway, once you get it all figured out and working, **submit your modified program as `Yourlastname_Linkage_cluster.py`**

3c. Test your program with the small ***ratiodata.txt*** file. Once you have it working, **run it on the *BacillusData2.txt* file, and INCLUDE IN YOUR SUBMISSION the resulting `.gtr` and `.cdt` files. Do these files work with `Java TreeView` or not?**

3d. Explain why it is often desirable to cluster and view the same data using a number of different methods. Do you think single linkage a *more* or *less* valid approach to clustering than the average linkage method? What do you notice about the resulting treeviews when you run your program under the single linkage vs. simple average? Or single linkage vs. complete linkage?