Hasan and Skyler

BNFO 420: Applications in Bioinformatics

Assignment 2: Clustal Pipeline: Aligning your Chakras

Due: 1/31/2017 before class (9:30am)

**Deliverables:** Each group will work together as a team to solve all of the following problems below. Again, this is more of an exercise to get familiar with Object-Oriented Programing in Python. Hopefully this does not bring back too many painful memories from your Java class. Search the deep recesses of your brain and awaken your inner pythonista. when in doubt “import this” to find your zen…

After class is over, you will responsible for completing the assignment as a group.

**Background:** There are eight DNA segments in the Influenza virus. Each of these segments encode for a viral protein that is critical to its transmission. It is of interest to see how these segments evolve overtime and one method of doing so is by doing a simple alignment. Clustal and other sequence alignment tools are commonly used in both the industry and academic side of bioinformatics. Like most bioinformatics software, there are two ways to run the program-- either online via a web application or locally on your computer via command-line. There are some obvious pros and cons to this, but we won't spend time discussing those right now. For this assignment you will be using the command-line version Clustal Omega to align DNA segments isolated from influenza. There are 8 segments in the influenza virus that each code for some critical viral protein. It is of interest to see how these segments evolve overtime and one method of doing so is by doing a simple alignment. Here is what we have provided to you: a working “clustal.py” program (which will have to be run on Compile) and the required files needed to get you going. The program contains 2 classes and a short main.We know that many of you are not familiar with object-oriented programming, whether it be in python or not, so we have included questions below to help guide you.

Part 1

1. In main, you will see the following line:
   1. clustalpipe = ClustalPipeline(infile,outfile,10)

In a few sentences, please describe the significance of this line? What are we doing here? Please also describe any special methods that may be called (“\_\_init\_\_”)?

1. Now in a few sentences describe the following line:
   1. clustalpipe.run()
   2. What is the relationship between this line of code and the code seen in “1a”?
2. What is the output of the “run()” method in the class ClustalPipeline()? How is this output file related to the class ClustalReport()?
3. In a few sentences describe what each method is doing in the ClustalReport class?
4. On the Compile server, run “clustal.py”
   1. HINT, HINT (please type in):

“python clustal.py Segment\_8\_seq.txtYourUserName\_Seg8.txt”

* 1. **Please turn in your output file**

Part 2

This program is functional, but………………….it seems rather limited; don’t you think?! What is stopping us from going online and using Clustal Omega? Nothing, but we are better than that! Let’s exploit the hidden magic of powers of com-pooOOoo-ter programming.

Notice how the infile in “part 1” contains only one of the eight segments the Influenza virus encodes for. That made our lives easy in many ways, because our data was already sorted. That being said, all you had to do was feed the file into the Clustal pipeline. What if we gave you a file that contained all eight segments, and told you to do an alignment for each segment? I think you know where we are going with this. Are you up for the challenge?

Automatically, you are going to need to find a way to separate and group all the sequences by segment type (1-8). Well that is exactly what you are going to do. Your new input file, “GenomicFastaResults.txt”, will find contain DNA segments from all eight segment-types. You will need a way to create eight unique fasta files for each segment-type. From there you need to create an alignment and an alignment report (just like in part 1) for each new file. Well how do you do that? Well luckily there are handy-dandy headers for each sequence. Below we included an example of a header. We bolded the part that is of interest. As some of you may know, this will require the use of regular expressions to capture what is needed. But not to worry, the format of the header is straight-forward and you will not need to employ a complicated/potentially convoluted expression.

>gb:CY103880|Organism:Influenza A virus (A/little yellow-shouldered bat/Guatemala/153/2009(H17N10))|Strain Name:A/little yellow-shouldered bat/Guatemala/153/2009|**Segment:8**|Subtype:H17N10|Host:Bat

This may seem like a large task to undertake- well we see that-- so we gave you all a super generous head start. First things first, if you don’t want to modify the clustal.py program you can simply import its classes into your new python script by copying and pasting this line at the top of your new program.

**from** clustal **import** ClustalPipeline, ClustalReport

Please see the provided code below:

**def** parse\_fasta(file):

header, DNAseq = None, [] *#creates a None type and an empty list to store the sequences that are separated over lines*

**for** line **in** file: *#iterates through lines*

line = line.rstrip() *#removes newline characters*

**if** line.startswith(**">"**): *#grabs lines starting ">"....signifies a header*

**if** header: **yield** (header, **''**.join(DNAseq)) *#If header is not a None type then returns a tuple with the header and the sequence.*

header, DNAseq = line, [] *#resets*

**else**:

DNAseq.append(line)

**if** header: **yield** (header, **''**.join(DNAseq)) *#If header is not a None type then returns a tuple with the header and the sequence.*

*#dir(object) gives what can be done with an object*

*#regex: [0-8] matches any number between 0 and 8*

**def** splitter(file): *#will return a list of filenames that have separated sequences*

filenames = [] *#creates an empty list that will be appended to below*

**with** open(file) **as** fp: *#opens file and*

**for** header, DNAseq **in** parse\_fasta(fp):

**#HMMMM some important code seems to be missing here!!!!**

return set(filenames) *#removes redundant filenames*

**Note:** This may seem really confusing at first, but with time this will all come naturally to you. Please note that there are parts missing from this code. You need to fill in those parts. You also need to consider what the “main” will look like. If this is too confusing you can try to find your own approach; however, if you do decide to stick to this approach, start by looking at the incomplete splitter function. Are any functions called within this function? Programming can be hard, and it is even harder when you are manipulating someone else’s code, good luck!