Homework 5: HMMs And Models

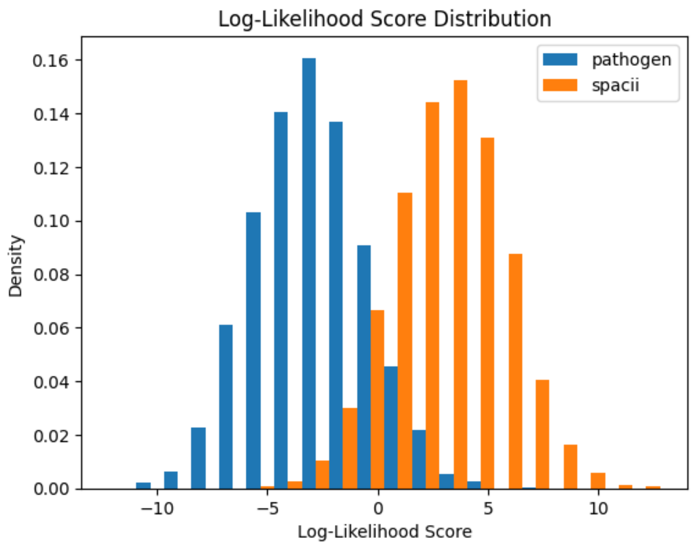
(100 points total)

Assignment guidelines

1) Submit your assignment files on canvas under module 10: COVID and Ancient Genomes
2) Please submit your code in file(s) called [name].py. Your code should be easy to open in a text editor so that someone can download and use the function you write.
3) Please submit a pdf with the answers to the questions at the bottom of the assignment (and your visualization(s))
4) Please submit a text file with the output of your code

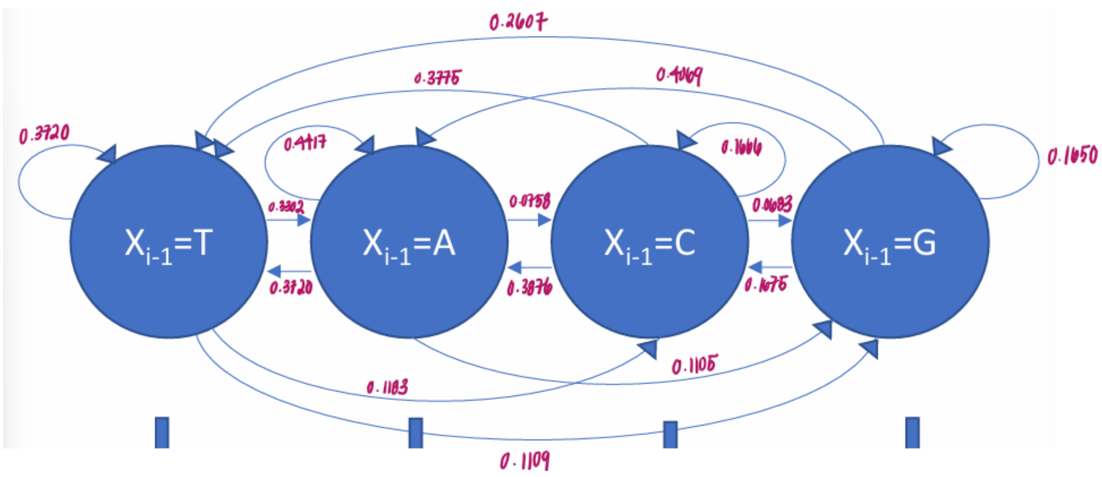
Complete the class assignment: Nucleotide Composition HMM (50 points)

Train the model on the files labeled as "training". Test the model on the Files labeled as pathogen and Spacii. In your output include a classification and a score for each sequence in both files. Also, include a plot containing the score distributions of scores for pathogen and Spacii sequences.



1) Complete the functions getLogLike and trainModel (30 points)
    a. Code meets specifications – the function exists and makes correct input and output
        i. Code includes getLogLike () function which computes the correct log-likelihood. (15 points)
        ii. Code includes trainModel() function which takes in the sequences of pathogen and Spacii and measures model parameters (15 points)

2) Draw (by hand is fine) the HMM that represents our pathogen model. Label all the transition states and indicate the emissions for each state (10 points)
    a. Emission Probabilities for Each State:
        i. T emits T: 1
        ii. A emits A: 1
        iii. C emits C: 1
        iv. G emits G: 1



3) Imagine that, due to incidental overlaps, when you assembled the Spacii genome you combined a bit of the parasite genome and your Spacii genome into a single contig. Describe how you would combine your models into a single HMM and use dynamic programming to identify a likely merge point between the Spacii and parasite sequences (no need to code this). (10 points)
    a. The single HMM will now contain two hidden states, one of which representing the Spacii model and the other representing the parasite model. Each of them will have their own emission probabilities. We can then assign transition probabilities in the case that one state switches to the other.
    b. After this, the Viterbi algorithm will find the most probable path of hidden states based on the transitions. Through the traceback, we can determine the point at which the model state is changed. This change will likely demonstrate the point in which the two genomes were merged.

Create an ORF generating HMM (50 points)

Create a THREE-STATE HMM that can simulate a sequence which contains Open Reading Frames (ORFs). It should run for a fixed number of emissions, x, during every run of the model. This model should be able to generate ANY possible reading frame with non-zero probability*.

If an ORF ends before reaching x emissions your algorithm should continue the sequence by starting a new ORF. Your model should include the following parameters for which you need to make up reasonable values:

k_0: a 3 by 1 matrix defining the probability of starting in each of your 3 states,

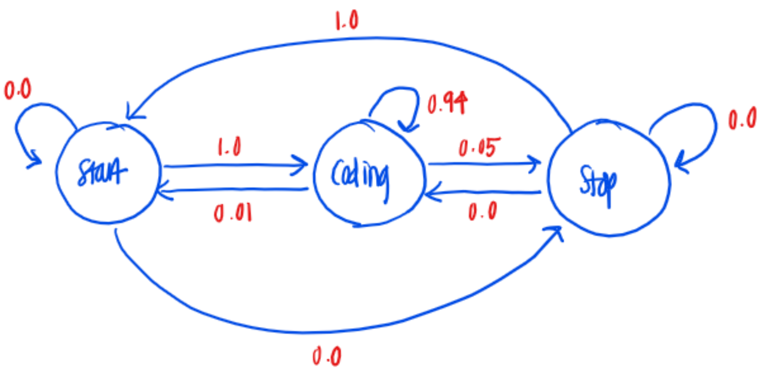
A: A 3x3 matrix containing the transition probabilities a_kl which is the probility of transitioning from state k to state l

E_k: one matrix per state representing probabilities for all possible emissions from this state.

x = the total number of emissions that you would like your model to emit per run.

Hint: What are 3 things that every ORF has?

1) Draw the HMM (by hand is fine) and justify your parameter values (25 points)
    a. Model is drawn with E_k, and A values shown on the model. Within the bounds of your parameter "x", all possible ORFs can be generated by your model. (15 points)
    b. Reasonable probability values are chosen, and an explanation accompanies the choice. (10 points)
        i. I chose the following parameters for a few reasons. The 'Start' to 'Coding' state will always occur, so in the designated 3x3 matrix of A, it is given a probability of 1. The 'Coding' state likely remains in this phase for most of the time, so I decided to keep that at 0.94. Because there is still a chance that the state changes to a start or stop codon along the way, I assigned the chances of going from 'Coding' to 'Stop' as 0.05 and 'Coding' to 'Start' as 0.01. Additionally, it is required to go back to a start codon to identify an ORF, so a probability of 1 is assigned to 'Stop' to 'Start'.
        ii. The reason I chose x to be 100 is because many functional ORFs are within the range of 100-300 codons. While x=300 may not allow all possible ORFs to be generated by the model, I went with the minimum threshold. Any value smaller than this might not result in a complete ORF. Additionally, with x=100, some sequences may contain more than one ORF and most sequences should include at least one complete one.



2) Code the model in Python**. Use it to generate 250 random sequences containing ORFs***. Measure the lengths of the 250 ORFs you generated and draw a distribution representing your result. Submit your code as “yourName_ORFHMM.py” submit the output of your code in “your_nameORFoutput.txt” and include the distribution in your pdf.
   - i. Code correctly represents your HMM from question 1 (10 points)
   - ii. ORF distribution is submitted (5 points)
   - iii. Is your distribution what you expected based on the model parameters you’ve chosen? Explain. (10 points)
     1. Yes, this distribution is what I expected based on the parameters I have chosen. For instance, if there is a 94% chance that the state is remaining in the ‘Coding’ phase, then there is more time for the ORFs to continue and increase in length. However, there is a 5% chance they end early, resulting in shorter ones as well. In our histogram, most ORFs end in the first 10-25 codons, while some extend for about 50+ codons, which gives us the tail.

* Use ATG as the only possible start. The probability may be small for some ORFs, but all should be non-0 for any ORF length of (x-1) where x is the total emissions of your model.

** It’s ok if your model generates a sequence that concludes with an incomplete ORF

*** For measuring the ORF lengths, use only the *FIRST* complete ORF in each sequence even if each sequence has more than one complete ORF.