COMPSCI 260, Introduction to Computational Genomics **22 November 2019**

Problem Set 7

Due: 5pm on 6 December 2019

Problem 1: Viterbi decoding a genome (30 points)

List of files to submit

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

Plan

In this problem, you will implement a Viterbi decoder for HMMs and apply it to the genome sequence of an ocean-dwelling archaeon. The decoder has three major steps: initialization (already implemented in `viterbi.py`), matrix fill, and traceback. Before you begin writing your code, you **must** read the information in `durbin.pdf`, included in the zipfile you downloaded for Problem Set 7. It is important that you perform the decoding using log probabilities; the PDF file describes exactly why and how this works.

Develop

a) Implement the matrix fill and traceback steps of your Viterbi decoder in `viterbi.py`. What is required to do this is pretty straightforward for the Viterbi case (but a bit more complicated when you do posterior decoding; more on that later). After computing the Viterbi decoding path $\Pi^\star$, your program should compress it into a list of dictionaries, where each dictionary corresponds to a contiguous segment of the path in which the state remained the same; the returned list of dictionaries should therefore look something like this:

```
[
    {'start': 1, 'end': 12, 'state': 'state2'},
    {'start': 13, 'end': 20, 'state': 'state1'},
    ...
]
```

b) Write a function called `count_segments` in `viterbi.py` to count the number of segments that exist in each state. Make sure to **report** the output of this function. It could be useful to **report** other information along the way, including perhaps statistics about the length distribution of the segments for each state.

Check

We will now use the Viterbi decoder to understand one aspect of the structure of a real genome.

A standard Markov model is not likely to be a great model for a genome sequence because the base composition of most genome sequences is not homogeneous. In particular, nucleotide composition can vary regionally throughout the genome.

An interesting example is found in bacterial genomes. It turns out that the overall GC composition of bacterial genomes appears to be fairly consistent for most of the genome, but that the GC composition of the parts of the genome encoding structural RNA genes is strongly correlated with the normal growth temperature of the organism (we think this is because structural RNAs have to maintain a base-paired secondary structure; thus, if the organism lives at higher temperatures, these sequences are driven towards higher GC composition because base-paired RNA structures with more G's and C's have higher thermostability). As one instance of this, consider the genome of *Methanocaldococcus jannaschii* (its genus was recently changed from *Methanococcus*, which is what it was when it was first sequenced, so sometimes you will see that used as well). It is a hyperthermophilic (meaning it survives at extremely high temperatures), methanogenic (meaning it produces methane) archaeon (meaning it is a member of the domain Archaea) that lives in deep ocean vents, where pressures and temperatures are exceedingly intense. Overall, the base composition of the genome is strongly AT-biased, but the regions of its genome that encode structural RNA genes are strongly GC-biased. We therefore might want to segment the genome sequence of *M. jannaschii* into regions of different composition to identify locations where structural RNA genes are found. We will try to accomplish this using an HMM.

Let's assume the *M. jannaschii* genome can be modeled as a two-state HMM, with the two states being: 1 (AT-rich; the bulk genome) and 2 (GC-rich; potential locations of structural RNA genes). The parameters of this model are specified in the file `HMM.methanococcus.txt`, whose format is similar to what you saw in the last problem set.

c) Once it's ready to go, use your Viterbi decoder to decode the regions in the actual genome of *M. jannaschii*. A copy of the complete *M. jannaschii* genome is available in FASTA format in the file `bacterial.genome.fasta`. **Report** the first ten and last ten segments in your decoding.

Note: For debugging and testing purposes, you may find it useful to apply your program to the short artificial sequence contained in the file `artificial.genome.fasta`, which was generated by this exact HMM. In that file, lowercase nucleotides were generated by state 1; uppercase nucleotides by state 2.

Reflect

d) How many structural RNA regions (segments corresponding to state 2) do you detect with your Viterbi decoder? A file containing the positions of known transfer RNA genes is available in `tRNA.locations.txt`. *What is the relationship between these positions and those learned by your Viterbi decoder? Discuss.*

Problem 2: Posterior decoding a genome (30 points)

List of files to submit

1. `README.problem2.[txt/pdf]`
2. `posterior.py`

Plan

This problem is similar to the previous Viterbi decoding problem, except that here we will implement a posterior decoder instead of a Viterbi decoder. To do so requires that we compute the probabilities that the sequence was generated from state 1 versus state 2 for every position in the genome: If we denote the complete genome sequence as X, we will be computing $P(\pi_i = 1|X)$ and $P(\pi_i = 2|X)$ for all i. To accomplish this, we will need to implement three routines:

1. Calculation of forward probabilities: we need to compute $f_k(i)$ for $k \in \{1, 2\}$ and all i.
2. Calculation of backward probabilities: we need to compute $b_k(i)$ for $k \in \{1, 2\}$ and all i.
3. Calculation of posterior probabilities: using the forward and backward probabilities along with $P(X)$, we need to compute $P(\pi_i = k|X)$ for $k \in \{1, 2\}$ and all i. This step is already implemented in the Python file `posterior.py`.

Again, you must be sure to do all this using log probabilities or your computer is likely to crash. Descriptions for how to do this are in the excerpt from the Durbin book that we provided; however, it is slightly more complicated in the posterior decoding setting because you need to *add* log probabilities rather than simply take the maximum of log probabilities. You'll probably want to create a helper function that knows how to efficiently compute the log sum of a set of probabilities when it is passed a list of the logs of those probabilities.

Develop

a) Implement the three routines found in the `posterior.py` skeleton file.

Check

b) Once the three routines are implemented in `posterior.py`, you can use the computed posterior probabilities to produce the posterior decoding $\hat{\Pi}$ by selecting the state that is most likely at each position.[1] As before, you may wish to begin by testing your program on the sequence in `artificial.genome.fasta` before running your posterior decoder on the real *M. jannaschii* genome.

Report the first ten and last ten segments in your posterior decoding $\hat{\Pi}$ of the *M. jannaschii* genome.

Reflect

c) How many structural RNA regions do you detect now? *What is the relationship between their positions and the positions of the transfer RNA genes? Discuss.*

d) Finally, compare the results of both decoders. *Which decoder seems to work better and why?*

Problem 3: Gene prediction and identification (40 points)

List of files to submit

1. `README.problem3.[txt/pdf]`
2. `assembly_tester.py`
3. `FLAG.txt`
4. `genscan.contig0.txt`, `genscan.contig1.txt`, etc.
5. `peptides.fasta`

[1] If you wanted to take this further and display a figure plotting the probabilities as a function of the position, feel free—it may help your analysis and a Python module like `matplotlib` could be used to produce it; but in this assignment, we only require you to segment the genome (determine the contiguous segments in which the most likely state is the same for the segment) and doing that only requires $\hat{\Pi}$.

Plan

One of your research colleagues, Dr. Klum C. Mann, has taken a human chromosome and cloned a small region of it into a bacterial artificial chromosome (BAC). This cloned region was then repeatedly "shredded" into even smaller random fragments whose lengths are all known to be 2000 ± 10. Those random fragments were then sequenced an average of 500 base pairs from both ends by a DNA sequencer, resulting in 244 mated pairs of reads (488 reads in total). His task is to assemble these reads into the original BAC-cloned chromosomal region, and then analyze the genes within it.

a) Use the number of reads and their average length, along with the extra fact that the coverage is 7, to mathematically estimate the length of the original cloned chromosomal region that is about to be assembled. Show your calculations. Use the results from a previous problem set to estimate the number of gaps that are expected to remain after assembly. Again, show your calculations.

Note: If you're feeling stuck, maybe checking out PS3 will jog your memory.

b) Because the DNA sequencer reads from both ends of a fragment, and thus generates reads in opposite orientations, one of the reads has already been reverse-complemented so that the two reads in each pair are both reported in the same orientation. Moreover, Dr. Mann has somehow managed to ensure that *all* the pairs of reads are reported in the same orientation with respect to the original genome. In other words, we can be confident that the reported reads are all taken from the same strand of the human chromosome. This represents quite a simplification over the situation that typically prevails in a sequencing project because when our genome assembly algorithm checks for overlaps, it need not check every read with the reverse complements of the other reads. It also means that all contigs (long stretches of contiguously assembled nucleotide sequences) and supercontigs (sets of contigs with gaps between them but those gaps are "bridged" by a mated pair of reads, with one read in one contig and the other read in the other; these are sometimes also called 'scaffolds') can be correctly oriented (but not ordered) with respect to one another.

Dr. Mann had the 488 sequences written down on note cards and stored alphabetically by their names. Unfortunately, on the way to your office, he slipped and dropped the whole container of note cards! To make things worse, a pair of note cards with irrelevant reads from another sequencing project slipped out of his pocket and into the pile when he bent down to pick them up. Not one to be easily deterred, Dr. Mann gathered up all the cards and placed them in the container in a random order. Arriving to your office, he placed the container on your desk and promptly discovered that he was late for a meeting and just disappeared, leaving the container of note cards behind.

Luckily, you had a scanner with powerful OCR (optical character recognition) that you used to digitally convert the sequences and their names into one gigantic FASTA file `paired.reads.fasta`. This file now contains 245 pair of reads, including the wrong pair. You then employed a shotgun sequence assembly algorithm to assemble the reads. The output of your assembler is a list of contigs and then a list of supercontigs. Obviously, the total number of supercontigs should be less than or equal to the total number of contigs. In this case, you got two contigs, provided in files `contig0.fasta` and `contig1.fasta`. They have a single gap between them and contig0 exists downstream of contig1. [For reference, we have provided the supercontig in the file `supercontig.fasta`, but it is simply the two ordered contigs concatenated with a `<---bridged gap--->` marker in between.]

Knowing that only one gap exists between the contigs, how could one most accurately estimate the length of the gap?

Develop

c) Write a small Python program called `assembly_tester.py`. This program should take as input three FASTA files. The first should contain a set of reads. The second and third should contain the contigs, contig0 and contig1. Your program should check three things.

- First, that every read appears in the correct orientation somewhere in the set of contigs.
- Second, that whenever a mated pair of reads appears in the same contig, the distance from the beginning of the first read to the end of the last read is 2000 ± 10.
- Third, that whenever a mated pair of reads appears in different contigs (but in a single supercontig) that the first read in the pair appears before the second read in the pair.

In short, your program should verify that the assembled sequence correctly incorporates all the information in the reads. If any of the above conditions are violated, your program should report which check failed. If all of the above conditions are satisfied, your program should simply report that the assembly is consistent with the reads.

You should verify these conditions by implementing a function called `find_contig_reads()`. It should take as input the reads and the two contigs as dictionaries (loaded from `get_fasta_dict()` and return a dictionary mapping the read pair's name (removing the end distinguishing `a`/`b`) to: each read's start and end position within the contig it was mapped to (1-indexed) and the contig it was mapped as either 'contig0', 'contig1', or None. The returned dictionary should look something like:

```
{
    'seq1': {
        'contig_a': 'contig1',
        'start_a': 1,
        'end_a': 100,
        'contig_b': None
        'start_b': None,
        'end_b': None,
    },
    'seq2': {
        'contig_a': 'contig0',
        'start_a': 101,
        'end_a': 200,
        'contig_b': 'contig1'
        'start_b': 201,
        'end_b': 300,
    },
    'seq3' : {
        'contig_a': None,
        'start_a': None,
        'end_a': None,
        'contig_b': None
        'start_b': None,
        'end_b': None,
    },
    ...
}
```

Check

d) In the context of an actual sequencing project, because of sequencing errors, repeats, and chimeric cloned inserts, the assembly may not always agree with the information present in the reads. Therefore, one should not expect in general for the assembled sequence to be consistent with the entire set of reads. In such settings, an assembly testing program can be used to see how well the assembler is doing at incorporating the

read data. In the context of this problem, there is a single pair of reads that doesn't belong to the assembly. Consequently, your program `assembly_tester.py` should identify which pair it is.

Report the pair of reads that doesn't belong to the assembly.

e) **Report** the number of reads where one mate maps to contig0 and the other maps to contig1. Additionally, for each pair of reads that map this way, **report** the possible length of the gap between the two contigs, using the knowledge that these two reads are taken from the ends of a single fragment of constrained length.

f) Based on the various values reported in the previous problem, calculate the size of the gap that exists between the two contigs. Show your work.

g) Dr. Mann returns to your office precisely when you finish writing your `assembly_tester.py` program. "Fine timing," you say. He smiles weakly, but goes on to say that he's a little unclear about how to find all the genes contained in this sequence. "We can't simply apply an ORF-finder here, because of that whole exon-intron-splicing thing, but there must be some HMM-based tool we can use." With that, he turns and leaves your office, mumbling something about having to teach a class. The only word you can make out as he trudges down the hallway is GENSCAN.

Find the GENSCAN web server at MIT (`http://hollywood.mit.edu/GENSCAN.html`) and use it for this problem [many of the links there are broken, but the tool still works; the paper describing GENSCAN is available from the Resources tab on the COMPSCI 260 webpage]. Notice that the GENSCAN HMM is not constrained to start in the state corresponding to background nucleotide distribution (state N) but can start in any state (i.e., the initial probability for every state is nonzero). *Explain why this fact might be relevant when you go to submit your specific assembled sequence.*

h) Submit your assembled sequence to GENSCAN. If you have more than one contig, you will need to submit each contig separately (GENSCAN cannot leverage the information that contigs are bridged into supercontigs, so just submit the contigs one at a time and ignore the fact that they may be organized into fewer supercontigs). Save the results for each contig into a text file (call them `genscan.contig0.txt` and `genscan.contig1.txt`). Submit these files.

Reflect

i) Just as you are finishing up with the results from GENSCAN, Dr. Mann pops in his head through your door and asks, "How's it going with GENSCAN?" You try to show him the output on your screen, but he seems to be in a great hurry as usual. "Could you just write up an explanation for me and include it with the results?" *Please explain to Dr. Mann how to interpret the GENSCAN output* (the goal here is to understand it yourself enough to explain it to someone else).

j) Now, across all the contigs you submitted, let k be the total number of protein-coding genes predicted by GENSCAN. What is k? For each of those k predicted genes, how many exons are predicted? Save all the predicted peptide sequences in one FASTA file `peptides.fasta`.

k) Are the predicted number of exons typical of genes in the human genome or atypical?

l) Once again, Klum's timing is impeccable: as soon as you have prepared the `peptides.fasta` files, he's knocking at your door again. It's almost as if he's got some preternatural sense for keeping you maximally busy (this is what grad school is like). Anyway, he tells you that he thinks your peptides may be related to one another. But he's also worried about the fact that each may not be a full protein. So he asks you to study them in relation to one another, initially as peptide fragments, and then to figure out the corresponding full-length proteins. He starts singing a ditty with valines and tryptophans as the main characters so you tune him out as you get to work.

Submit the k peptide sequences to BLAST to determine their identities, using `blastp` and the

UniProtKB/Swiss-Prot protein sequence database. Given that we know these genes are from the human genome, are the top hits returned by BLAST believable? Which of the peptide sequences represent full-length proteins, and which represent fragments? How does this relate to their locations in the two contigs?

m) You should observe that three of these peptides return BLAST matches that are similar to each other. In what sense are the three proteins related to one another? What do these particular types of protein do in the cell?