Programming Project 4: Hidden Markov Models

CS 181, Fall 2022

Out: Nov. 28
Due: Dec. 12, 11:59 PM

1 Task

In this assignment, you will implement the Viterbi algorithm for inference in hidden Markov models.

2 HMM Specifications

You will implement the Viterbi algorithm to identify the maximum likelihood hidden state sequence. You will be given a transition matrix, an initial state probability vector, and an observation sequence. Your program shall then output the maximally likely sequence of hidden states that produced the observation sequence, and on a second line, the probability of the model emitting such a sequence and following the maximally likely sequence of hidden states.

2.1 Setup

To grab the support code, see the website. We've provided an example shell script, `viterbi.sh` and a few test cases.

2.2 Reminders About Programming Language

Our Gradescope autograder is currently configured to accept solutions written in Python 2, Python 3, Java, R, and Julia.

Your solutions generally should not require the installation of any packages that do not come in the standard installations of your chosen programming language. However, if you are using Python, you will also have access to numpy and pandas.

To facilitate anonymized & automated grading, each of your solutions must be accompanied by a shell script. Make sure each problem is able to output the correct result using the shell script provided. The shell script you turn in must explicitly call your language's compiler or interpreter. If you are using Python 3, your shell script must run your program using the command "python3" rather than "python", as "python" will run your code with Python 2.

Your shell scripts should print exactly what is shown in the examples given for each problem. If you print any extra text, you will fail our autograder and lose points. This means that if you are coding in R, you may need to print text using "cat()" instead of "print()".

Be sure to print any terminal output to stdout, which is the channel that the standard print functions write to in most programming languages. If you print to stderr, your solution will be interpreted as an error

message and fail the autograder.

2.3 Shell Scripts

You must provide a shell script, `viterbi.sh`, that accepts two arguments and can be run using the following command:

```
> sh viterbi.sh config obs
```

. . . where `config` will be the path to a configuration file, described below, and `obs` will be the path to an observation sequence file.

The program you hand in should not throw exceptions for any valid inputs.

2.4 Input and Output Format

The configuration file will have the following format:

- Line one contains the number of hidden states, a comma, and the number of observations.
- Line two contains a comma separated list of hidden state names.
- Line three contains a comma separated list of observation names.
- Line four contains the initial state probabilities.
- Line five contains the transition matrix, columns comma separated and rows semicolon separated. Transitions go from the row's hidden state to the column's hidden state.
- Finally, line six contains the observation matrix, similarly separated. Each row represents a hidden state, while each column represents an observation.

In your output, all probabilities should be printed in scientific notation rounded to 4 decimal places. For example, a probability of 1 should be printed as "1.0000e+00". If you are working in Python, we recommend using exponent-format (e-format) strings.

An example model and several input sequences and solutions follow. You should make your own tests and perform additional testing, with particular emphasis on edge cases.

config

```
2, 4
A, B
A, T, G, C
0.5, 0.5
0.75, 0.25; 0.25, 0.75
0.3, 0.3, 0.2, 0.2; 0.25, 0.25, 0.25, 0.25
```

obs1

```
ATATAAAGCACCGTTGCG
```

Output:

```
> ./viterbi.sh config obs1
AAAAAAABBBBBBBBBBB
6.5325e-14
```

obs2

```
ATATAAAGCACCGTTGC
```

Output:

```
> ./viterbi.sh config obs2
AAAAAAAAAAAAAAAAA
3.7877e-13
```

obs3

```
TATAAAGCACCGTTGC
```

Output:

```
> ./viterbi.sh config obs3
AAAAAAAAAAAAAAAA
1.6834e-12
```

obs4

```

```

Output:

```
> ./viterbi.sh config obs4

1.0000e+00
```

2.5 README

You must include a README file. Include the following information:

- A description of any known bugs.
- Anything you want the TAs to know about your project.

3 Application

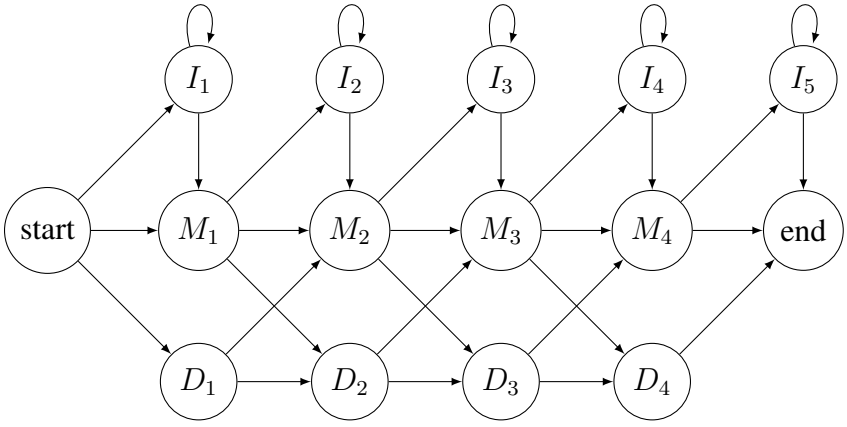
Recombination is an important source of diversity that helps populations evolve by exchanging sections of genetic material to create new genotypes. Recombination can be advantageous to all species, as it provides additional opportunities for individuals to achieve better fitness in their environment—and viruses are no exception! Viral recombination can occur when two or more viral strains infect the same host and induce their reproductive cycle using the host's cellular machinery. Since the host is doubly-infected, these viral strains have ample opportunity to exchange genetic material. Furthermore, these viruses do not even need to be of the same species to undergo recombination, as viral infection is not limited by the same barriers as sexual reproduction.

The coronavirus family is known to undergo frequent recombination. In particular, recombination has been implicated in the emergence of the SARS and MERS viruses. You suspect that a similar phenomenon may have occurred with SARS-CoV-2. In this assignment, you will be using the Viterbi algorithm to try to identify genetic recombination in a set of coronavirus samples.

This problem will ask you to implement a simplified version of a *jumping profile Hidden Markov Model* and is based on the work from this paper. It also builds upon Problem 2 from HW5. We will provide you with all the background necessary to approach this problem, but you may refer to the paper and HW5 for ideas as needed.

3.1 Profile HMMs

Recall that in HW5 we constructed an HMM using a consensus sequence obtained from multiple alignment. For a consensus sequence of length 4, our HMM looked like the following:



This is a **profile HMM**, or **pHMM**, and it can be applied to consensus sequences of any length. Recall that this HMM can produce a sequence of observations that is homologous to the consensus sequence, and that specific mutations in the homologous sequence can be represented by transitions in the HMM:

1. A match or mismatch is represented by transitioning directly to M_i, where an observation symbol is emitted with a particular frequency.
2. An insertion of some length is represented by transitioning to I_i, emitting symbols while remaining in I_i for some time, then transitioning to M_i

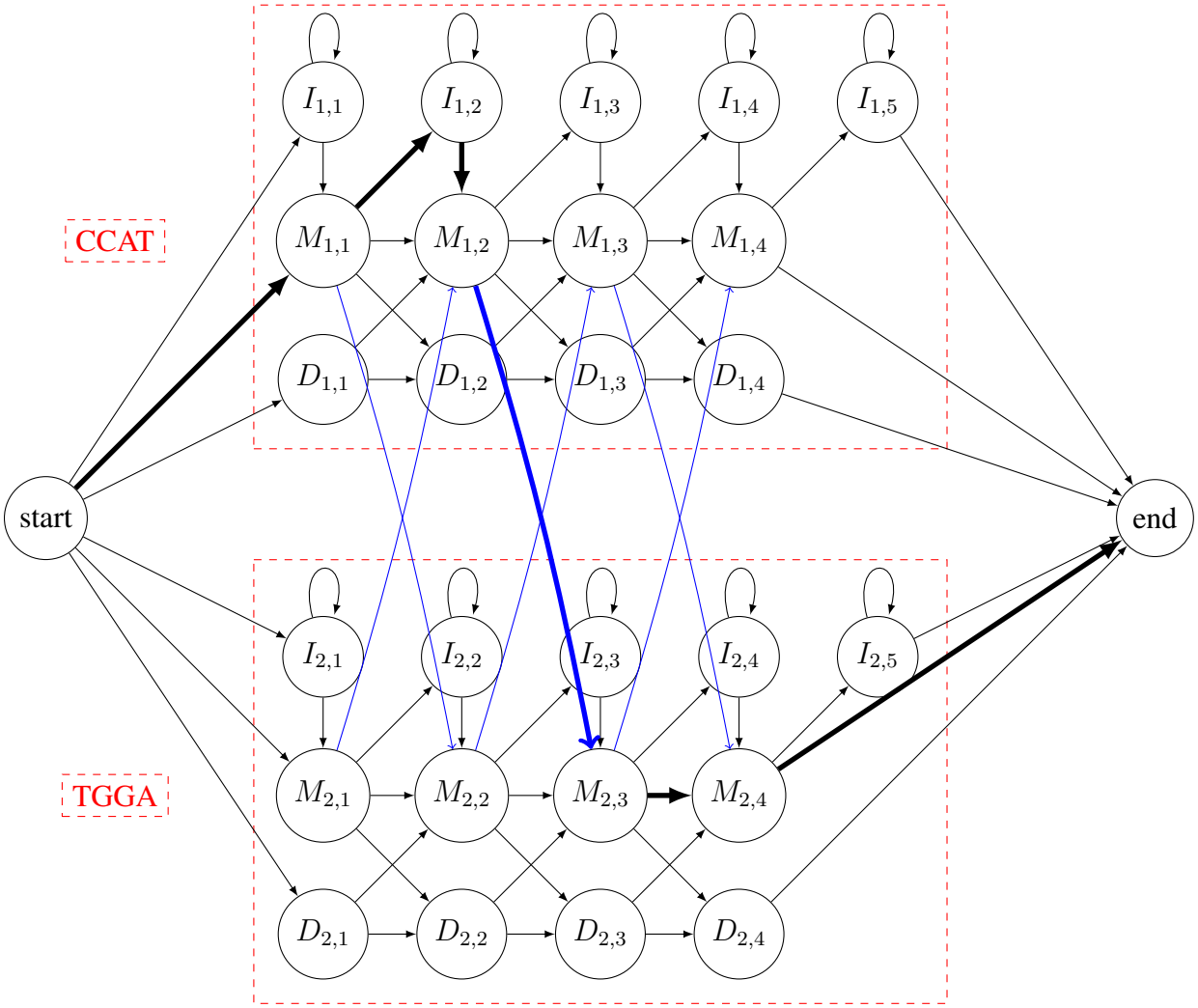
3. A deletion is represented by transitioning to D_i, where a gap character is emitted.

This setup is great for inferring whether an observation sequence is derived from a singular consensus sequence, but in this case we really want to detect recombination; in other words, we want to infer whether an observation sequence is derived from several consensus sequences. That's where jpHMMs come in!

3.2 Jumping Profile HMMs

A **jumping profile HMM**, or **jpHMM**, is a combination of multiple profile HMMs in parallel, with transitions between the pHMMs. Intuitively, these inter-pHMM transitions represent an observation sequence *jumping* from one pHMM's consensus sequence to another pHMM's consensus sequence.

To illustrate the use and construction of a jpHMM, consider the following scenario. Imagine we have two reference sequences which we suspect have undergone recombination: `CCAT` and `TGGA`. We need to construct pHMMs for each, so we will designate pHMM 1 to represent `CCAT` and pHMM 2 to represent `TGGA`. Let's say we would like to determine whether or not the sequence `CTCGA` is recombinant from our two reference sequences. We would construct the following jpHMM:



Each state is denoted by its corresponding mutation and a pair of integers: the index of the pHMM followed by the index of the position in the profile's sequence. The blue lines indicate some of the possible inter-pHMM transitions we added when combining our profile HMMs into a jpHMM. The bolded lines indicate the result of running the Viterbi algorithm on our jpHMM with the observation sequence `CTCGA`. So the most probable sequence of hidden states for the observation sequence is

$$M_{1,1} \to I_{1,2} \to M_{1,2} \to M_{2,3} \to M_{2,4}$$

You'll notice that by focusing on the first indices of each state in the sequence, we find that for the first three symbols, the HMM is in a state from pHMM 1, while for the final two symbols, the HMM is in a state from pHMM 2.

Since the most likely sequence of hidden states jumps between our two consensus sequences, we would conclude that our observation sequence `CTCGA` is recombinant between our reference sequences!

3.3 Your Task

Our set-up script should have provided you with a file, `application.txt`, containing 5 short amino acid sequences. These are small fragments of the receptor-binding domain in various spike proteins, each of which highlight a particular **SLiM**, or **S**hort **Li**near **M**otif: the RGD motif (Arg-Gly-Asp). We've separated these into potential parental sequences and sequences which are suspected of being recombinant. Your task is to determine whether these 3 suspects are recombinant using the Viterbi algorithm.

You are free to design your HMM however you like. There is no one right answer; answers which are biologically reasonable and fully justified will receive credit. Some points to consider:

- **Hidden states:** The example diagram above includes the full range of hidden states in a profile HMM. What is the purpose of including I states? M states? D states? Considering the *lengths* of the sequences we've provided, are all of these states necessary? If you were to exclude some states, what limitations does that decision introduce?
- **Inter-profile transitions (conceptually):** How would a transition between one pHMM's I state and another pHMM's I state be represented in an observation sequence? I to D? How would a transition between one pHMM's M state and a non-neighboring M state of another pHMM be represented?
- **Inter-profile transitions (in practice):** How does the frequency of inter-pHMM transitions affect your program's ability to transition between profiles *precisely*? How high should you set these frequencies to enable "jumps"? How low should you set them to prevent excessive "jumping"?
- **Emission probabilities:** Many amino acids share certain chemical properties which are relevant to their function. How can you assign emission probabilities to account for the physiochemical similarities and differences between amino acids? Additionally, the SLiM of interest is the RGD motif, which you may be able to locate in the sequences. Should you assign more weight to a correct amino acid sequence in this region?

You should submit and report the following in a file named `application.pdf`:

- A reproducible description of your workflow. You should provide enough detail for us to trace your steps exactly from start to finish. Please tell us if you wrote any new scripts. If you save your outputs to a file, mention it. If you run any commands in the shell, tell us which commands exactly. If you create new files or edit files by hand, we should know that too. The point is that it should be clear exactly how you generate outputs from data.
- A justification for any HMM design choices you make. This includes how you came up with transition probabilities and emission probabilities in general. You do not need to answer every design question above, but they may be useful for organizing your thought process and explanations.
- Whether you believe any of the samples contain (1) recombinant material from both parental sequences, (2) nonrecombinant material derived from one of the parental sequences, or (3) nonrecombinant material with little relation to either parental sequence. Justify your conclusions.
- ***Optional:*** If you believe there may be better groupings for parental sequences and potential recombinants, tell us your alternative grouping and why!

We've left you with a lot of freedom in approaching this task and we've introduced some new conceptual ideas for HMMs. This offers a lot of opportunity to be creative with your design, but we also recognize that having this freedom can be daunting. Come to TA hours if you want some feedback on your thought process or if you have questions about the concepts we've presented!

4 Handin

To hand in this assignment, upload your shell scripts along with all files needed to run your code on Gradescope. If you have any subdirectories or folders that you would like to preserve in your handin, you will need to compress your submission into a zip file and upload the zip file to Gradescope. However, to ensure that the autograder runs your handin properly, make sure that all shell scripts are present at the root of your handin; in other words, do not place your shell scripts inside any folders.

When you hand in your solution, our autograder will immediately run on your submission. We have made the test cases provided in the project handout immediately visible to you so that you can ensure your solution runs properly with the autograder. If you fail the autograder and cannot determine why, notify a TA and we will help you to diagnose the problem. If your final submission is not compatible with our autograder, it will be very difficult for us to give you credit for this assignment.

5 Grading

We will grade your handin using pre-generated test cases with a certain number of points allocated per test case. Test edge cases extensively!