Algorithms in Computational Biology, 2021
Exercise 2 - Programming - HMM Decoding

Due date: 21/12/22

Submission should include a tar file named $ex2.tar$, containing the following files:

- decoder.py
- Possibly other python files

HMM model

In class we discussed using HMMs for finding hidden states in sequence data. In this exercise you will implement the forwards/backwards algorithms for calculating liklihood as well as the Viterbi and Posterior decoding methods for identifying likely hidden states of the input sequences. In this exercise we will assume there are always **2** hidden states. The input sequence will be a sequence with a generic alphabet size.

Recall an HMM model with n states ($S = \{s_i\}_{i=1}^{n}$) that emits data from an alphabet of $\Sigma = \{\sigma_j\}_{j=1}^{m}$ can be described with parameters $\theta = (\tau, e)$ where:

- τ - transition probability matrix ($n \times n$), where $\tau_{i,j}$ is the probability of moving from state s_i to state s_j
- e - emission probability matrix ($n \times m$), where $e_{i,j}$ is $e_{s_i}(\sigma_j)$: the probability of s_i to emit letter σ_j.

States: It will be convenient to add special start and end states besides the two actual hidden states.

- B_{start}, B_{end} are the start/end states. The model is forced to begin in B_{start} and to end in B_{end}. These special states do not emit any value (see details below).
- sate 1 - in the example of fair/loaded dice this can be thought of as the "fair" sate

- state 2- in the example of fair/loaded dice this can be thought of as the "loaded" state

Transitions: Since we only have two hidden states our tranistion matrix can be fully described with two numbers:

- p is the probability of transitioning from state 1 to state 2.
- q is the probability of transition from state 2 to state 1.

Initial Start Probabilities: Besides for the tranisiton probabilities there is the "Steady state" tranisitions or the probability to start at state 1/2. In the lectures this was reffered to as P_0. Since we only have two hidden states this can be fully described with one number. For the sake of simplicity we will refer to $p0$ as the probability to start at state 1.

- $p0$ is the probability of the sequence starting at state 1.
- $1 - p0$ is the probability of sequence starting at state 2.

Emissions: Emission probabilities will be provided as input in the form of a matrix.

Implementing Start states: In order to enforce the model to begin at B_{start}, we recommend the following trick:

- End: add the special character ^ to the alphabet, and append it to the beggining of the input sequence. Edit the emissions matrix such that

$$e_{S_i}(\hat{}) = \begin{cases} 1 & S_i = B_{start} \\ 0 & S_i \neq B_{start} \end{cases}$$

  Don't forget to remove the special character from your output

HMM implementation

In this exercise you will implement the decoder.py program, with 4 possible algorithms: Forward, Backward, Posterior decoding and Viterbi. Write a program that reads the model probabilities, initializes a new HMM model, and runs these algorithms.

Input:

- The code receives an initial emission matrix: initial_emission.tsv - defines the emission probabilities of the states for each letter in the alphabet. The first row will describe the letters of the alphabet. For example in the fair/loaded dice example the first row will contain the letters: 1, 2, 3, 4, 5,

6. There will be two rows after that, corresponding to the first and second hidden state. So row 2 in the 3rd column will contain the probability for the first state to emit the 3rd letter.

- Transition probabilities are passed as command line arguments $p, q, p0$.
- The sequence, as a string (seq).

1 Probability of Observations

Using the HMM model it is possible to evaluate the (log-)likelihood that some sequence of observations came from the model.

Implement the **Forward** algorithm. The function should receive a sequence, transition, start probabilities, and emission probabilities, and generate the Forward table. Use this table to find the log-likelihood of the sequence.

It should be possible to invoke the forward algorithm from the command line using the following syntax:

```
python3 decoder.py --alg forward seq initial_emission.tsv p q p0
```

The program should print the log-likelihood of the sequence. The Forward table should not be printed.

Similarly, implement the **Backward** algorithm.

Running the following command should print the same output as the forward command (i.e. the log-likelihood):

```
python3 decoder.py --alg backward seq initial_emission.tsv p q p0
```

For example, if the input sequence is 266, the probabilty of moving from fair to loaded, p, is 0.01 and the probability of moving from loaded to fair, q, is 0.05, the start probability, $p0$, is 0.5, the emissions are uniform for fair and 0.1 for rolling 1,2,3,4,5 and 0.5 for rolling a 6 when the state is "loaded" then the output is simply (output at least 2 digits precision):

-4.26

2 Posterior Decoding

Using the Forward and Backward algorithms, you can now compute posterior probability for each state of the i'th position in the sequence X, $P(S_i = s|X)$. Use this to find the most likely state for each index, given the sequence.

Running the following command:

```
python3 decoder.py --alg posterior seq initial_emission.tsv p q p0
```

Should print the most probable state for each index, found by the posterior decoding.

The output format should be as follows (for $p = 0.01, q = 0.05, p0 = 0.5$ and the provided *initial_emission.tsv* file):

```
11111111111111111111111111111111112222222222222222
12453634212453134212453421245313426666666266666661

22222222222222211111111111111111111111111111111
66666664666666612453134212453134212653134212 4
```

Blocks of 2 lines of length up to 50 characters followed by an empty line. Bottom row should be the given sequence and top row the corresponding hidden state ('1' - first state, '2' - second state). Do not print the special states (B_{start}, B_{end}) or characters (^, $).

3 Viterbi Decoding

Given the model parameters (i.e. transition probabilities p, q, start probability $p0$ and emission probabilities) and a sequence of observations $X = (x_i)_{i=1}^{n}$, we want to find the most likely assingment for the hidden states that emitted X. That is, which positions in the sequence are part of a motif. In this section you are required to retrieve the most probable path for the hidden states given the emitted sequence.

It should be possible to invoke the viterbi decoder from the command line using the following syntax:

```
python3 decoder.py --alg viterbi seq initial_emission.tsv p q p0
```

The program should print the most probable path of hidden states that could emit the sequence seq, using **Viterbi** algorithm. The output **format** is identical to the Posterior decoding format above.

Are the results identical to the posterior decoding? Think why?

Tips and implementation issues

Running time

- Your program should run fast. Your Forward & Backward methods will run many times on the next exercise. Use vectorized numpy operations instead of nested loops as often as possible.

Underflow

- The probabilities in the Viterbi/Forward/Backward tables decrease with the length of the input sequence, and it might cause underflow. If the sequence is long enough (e.g. >200bp), It's likely the last columns of the tables will be all zeros, and the output will be incorrect. The solution we suggest for this problem is working in log space. Replace multiplication with addition and addition with logsumexp.

- It is recommended to first implement the algorithms in a straightforward way, with nested loops and without logspace. Make sure it's correct for short sequences (5-10 characters), and then change it to log space and vectorized operations.
- You are allowed to use non-standard libraries for reading / writing files (e.g. pandas, Bio).