PYTHON PROJECT - Implementation of HMM

Introduction:

Hidden markov model is being extensively used in various fields including bioinformatics. In bioinformatics, it has been used in sequence alignment, in silico gene detection, structure prediction, data-mining literature, and so on.

Markov model: It is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.

A **Hidden Markov Model** is a Markov Model in which the system being modeled is assumed to be a Markov Process with hidden states (or unobserved) states.HMM is one in which you observe a sequence of emissions, but do not know the sequence of states the model went through to generate the emissions. Analyses of hidden Markov models seek to recover the sequence of states from the observed data.

Transition matrix:

Transition matrix has the probability values of transitioning from one state to another.

Emission matrix:

Emission matrix contains the probabilities of the observable state given the hidden state.

Viterbi Algorithm:

Calculates the most probable state path for a HMM.

Aim:

- To understand and implement HMM by using viterbi algorithm to find out AT and GC rich regions of the given nucleotide sequence.

Before adressing the above problem, we will first try to build a simple HMM modal to find the most probable sequence using HMM, given AT/GC rich probabilities (HMM parameters).This can be done without using the vitterbi algorithm. For this we need to provide:

- Transition matrix (containing the probability of current observable state [AT-rich OR GC-rich] given the previous state)
- Emission matrix (containing the probability of the current nucleotide (hidden state) being (A or C or G or T) given the current state(AT/GC-rich)) and generate a sequence which obeys the probabilities.

Packages to be imported:

- **numpy:**In Python, numpy (Numerical Python) is a package consisting multidimensional array objects and a collection of routines for processing those arrays. Since we store the transmission and emission probabilities in a matrix we need arrays which can be created and utilised by importing the numpy package.
- **Random:**

random is a module in Python which can be imported to do random sampling.

random.sample(population, k)

Function in random package which returns a k length list of unique elements chosen from the population sequence. Used for random sampling without replacement.

Random():

Function in random package which selects a number between 0 and 1.

Input:

1. Transition matrix
2. Emission matrix
3. Length of sequence to be generated
4. Initial probability (probability of 1st position of the sequence to be AT-rich or GC-rich which are the possible observable states)

Variables:

1. Transmat
2. Emmat
3. Seqlen
4. Initprob
5. firststate
6. Obsstates
7. newseq

Output:

Most probable nucleotide sequence.

Pseudo Code:

1. The values in the transition and emission matrix to be obtained from user.
2. The first observation state of the sequence is selected using the initial probability variable (from user)
3. The variable firststate is assigned a state such that the probability of the state being selected is the same as that in initial probability variable by using random() function from the random module (which generates a random number between 0 and 1) and a series of if statements.
4. This state is stored in the list 'obsstates'.
5. The first nucleotide (hidden state) is selected using the probability values in association with the state given by the user as emission matrix.
6. As in step 2 random() is used.
7. This nucleotide is stored in the list 'newseq'.
8. The observable state (AT-rich/GC-rich) of the previous position(1st) is obtained and stored in a variable. The observable state of the second position of the sequence(AT-rich/GC-rich) is selected using the transition matrix. As in step(4) this can stored in 'obsstates' by appending it to the list.
9. The nucleotide in the second position is selected using the emission matrix. As in step(5) this is stored in 'newseq' by appending the list.
10. Steps 8 and 9 are repeated for the remaining positions in the sequence of length=seqlen.

11. Print newseq to get the sequence predicted by HMM.

Application - Viterbi algorithm to find AT and GC rich regions

Input:

1. Transition matrix (state transition probability distribution)
2. Emission matrix (observation probability distribution)
3. Nucleotide sequence (observed)
4. Initial probability

Output:

The state of the nucleotides at each position in the sequence i.e prints whether AT or GC rich.

Function: Viterbi – to backtrack and print the sequence of hidden states.

Parameters of the function:

1)nucleotide sequence

2)list of possible hidden states

3)Emission probabilities

4)state transition probabilities

List : V with an empty dictionary

Pseudo Code:

1. Get sequence from user and use split() to convert it into a list.
   Say list now has N elements.
2. Define the states(hidden) in a list (AT-rich/GC-rich).
3. Input the transition and emission probabilities.
4. Create a function and pass the list of nucleotides, states, emission & transition probabilities as arguments.
   Within the function:
5. Create a list V with an empty dictionary.
6. For the first nucleotide, create two key value pairs for both AT and GC-rich states.
7. First key 'prob' = (probability of starting with that state) *(probability of 1st nucleotide being observed given that state).
8. The second key 'prev' (which is meant to hold the previous state) is initialised with the value 'none'(as it is the first nucleotide)
9. Append an empty dictionary to the end of V for the remaining nucleotides (in a loop).
10. Now for each state multiply the each state's probability value in the previous dictionary with the probability of transitioning from that state to current state (from transition matrix) and find the maximum of the two values.
11. Multiply this probability value with the probability of getting that particular nucleotide being observed.

12. In the new dictionary previously created in the list V, create key ‘prob’ and assign the above value. Create key ‘prev’ and assign the previous state as value.

Backtracking:

Backtrack: Start from the final element of V and find the state with highest probability value. Store that state in a variable.

13. From the end of V to the beginning, backtrack by storing the states with highest probability value for each nucleotide, in the above variable.
14. Print that variable to obtain the states (whether each position is AT-rich or GC-rich).

Submitted by

Anuradha K-(119013045)

Raja Roobini S-(119013028)

Sunayanaa Sridharan-(119013040)