COMP9318 Project Report

Kuan-Chun Hwang   z5175539

Question 1

Question 1 focuses on the the implementation of Viterbi algorithm to get the most likely state sequence for an observed address and its log probability. This involves three main steps:

1. Preprocessing the data file.
2. Compute transition probabilities and emission probabilities with smoothing.
3. Running the Viterbi algorithm itself to get the result.

Preprocessing

Preprocessing is an important step as the data structure chosen to store the data into will have significant impact on the rest of the implementation. Furthermore, the accuracy of preprocessing will obviously have a direct impact on accuracy of our result.
To preprocess the state file, we store the a dictionary that consist of state IDs as the key and their actual name as the value because IDs are what is used in the rest of the program and we can simply use it to get their meaning back. We also store the frequencies using a numpy arrive such that the it is a 2D numpy array with dimension of state from and state to.
To preprocess the symbols file, we also do something similar with the state file where we store the frequencies using a 2D numpy array with dimension of state and symbol emitted. However, there is another small difference such that instead of storing the symbols as IDs as the key and what the actual symbols are as values, we use the symbols themselves as the key and IDs as the value. This is because we need to be able to convert symbols from observation into IDs later on and this structure is much more convenient.
After preprocessing the symbols file, we convert all the queries in Query File to their corresponding symbol ID form to be used later.

Computing the Transition and Emission Probabilities

Computing the transition and emission probabilities is important as it has direct impact on the accuracy of the Viterbi algorithm. Here, we used add-1 smoothing because it is a straightforward approach. We are required to do smoothing here because we need to shift some probability of seen state and symbols to unseen state and symbols because if we calculate the probabilities with raw training data, we eliminate the probability of unseen state transition and symbol emission such that when we never account them. We do this smoothing to make the probabilities more realistic.
To calculate the transition probabilities, we first initialise a 2D numpy array with all 0s. The dimension of this array is the number of state, N, on both sides. We iterate through the 2D numpy array and at each i and j, we compute the transition probabilities using the smoothing formula provided from the spec:

transition probability from i to j +1 / sum of frequency of i transmitting to another state + N -1

However, we also check if state j is BEGIN or state i is END or state i is BEGIN and state j is END because we know that the state transition to BEGIN and state transit from END is always impossible so

we do not want to do smoothing for these cases. Also, we cannot transit from BEGIN to END right away so we do not do smoothing for this case as well.
Next we compute the emission probabilities as well and following similar steps with transmission probability, we initialise a 2D numpy array and iterate through using the formula provided from the spec. Here, we also skip the state the BEGIN and END as they have no symbols to emit and do not require any smoothing.

Viterbi Algorithm

For the Viterbi algorithm, we use 2 2D numpy array to store probabilities, T1, and the path, T2. The dimension of these array are the number of state and the length of the observation plus begin and end sequence. We then compute the log transition probability from BEGIN to all of the state for the first column inside the probability matrix, T1. We then compute the first transition from BEGIN to all state with addition to the log probability that they emit the first symbol. The first step is a little different from the rest since we are only transiting from BEGIN such that it is the only previous state possible. As a result, the first column of T2 stores BEGIN state's ID only and second column of T2 stores the log probability of each state after transiting from BEGIN to first state.
In all subsequent step, we compute their maximum log likelihood by adding the log probability we stored for last state in the previous step and adding the log transition probability from all last state to the current state and adding the log emission probabilities of all current state to emit the observed symbol. Here we use log as number gets too small during probability multiplication, precision can be lost and we approach 0 very quickly. Nevertheless, at each step, we compute all combination of last state's transition to current state then we take the maximum probability and the state it came from and store it into our probabilities matrix, T1, and path matrix T2.
Finally, we reach the end of observed sequence where we need to transit from the last state to END state. This is simply done by adding the probability of all the state to their transition probability to END state as they can all only transit to END state.
We then perform backtracking where we look at the last column of the probability matrix T1, and get the argmax, which is the index with the largest probability. This largest probability is our result for the most likely state sequence log probability. We then backtrack using T2 by looking at the corresponding index since it tells us where the state came from. After this, we recursively iterate backward and at each step, we look at the current value in T2, which tells us the state it came from and then look at the previous column's value at that index until we reach the first column. In each step, we add the value we see in T2 into our path. Also note that since the path must start with BEGIN and END, we add the END state's ID before we start and BEGIN state's ID when we finish. Finally, we reverse it to get the final path.

Question 2

In question 2, we are required to get the top k state sequences instead of the just best state sequence in question 1. The preprocessing part and transition and emission probability computation are all the same. However, this time we are required to store all top k result for each state in time rather than just storing the top result. As a result, our T1 and T2 matrix becomes three dimensional with the new dimension being k. Furthermore, in our path matrix, T2, we store the state it came from and also the k result it came from.

We follow a similar manner to question one where we first compute the transition from BEGIN to each first state and in this first case, the top k probabilities are all the same and they could all only come from BEGIN. Afterwards, similar to question one, but this time we use an extra for loop to loop through the top k result of the probability from the last state's previous time step's top k result and compute the probabilities. We add all of these probabilities into an array along with the state and k they came from. We then sort this array to get the top k result and put the relevant information into our T1 and T2 matrix. We do this for all subsequent observation until we reach the end, where we do our transition from last state's top k to END state.
From this, we perform backtracking again, except this time, it is a 3D backtracking since we need to go to the k where each of the result came from as well. We take the last slice of our T1 matrix as it contains all the information on the last probabilities and backtrack from all of the result. The reason we cannot use top k result here is because there is a chance that our k result and k+1 result are ties and taking top k here will eliminate the possibility of considering the k+1 result without finding out its path to see if it should be included instead of the k result. From this, we can get all the top k result's indices for us to perform back tracking. We sort them by their probabilities first as that is what the question requires. In our backtracking, we look at the i and j value which are the corresponding state and k that was used to reach that state. The rest is similar to question one such that we keep repeating this step and look at the iterate backwards to get our path.
After getting all the paths, we sort them by their probabilities first then the reverse of their path IDs as the question require us to do that to break ties in the result.

Question 3

In question 3, we are required to improve the accuracy of our Viterbi implementation. One approach would be to improve the smoothing method as the add-1 smoothing used isn't that great since too much of the probability mass is moved, which reduces accuracy as you get an incorrect probability distribution that does not normalize into 1.
The alternative smoothing method we choose to implement was absolute discounting as it is a discounting method instead of a correction method. It takes off the probability from seen one and redistribute it to the unseen training data which keeps the probability distribution normal. We approached this by changing how smoothing was done when computing the transmission probability.
In absolute discounting, we set a constant d that we subtract off every non zero probability and for probabilities that are 0, we compute their unigram probability and and apply a normalization onto it to get the smoothed probability.
After getting the smoothed probability, the rest of the implementation is the same as question 1.