Project report

Leo Hoare (z5171844) Simone Zanin (Z5176221)

Question 1.

A standard implementation of the viterbi algorithm was followed for question 1.
First the states and symbols were passed to create the transition and emission matrix. These created a transition (N,N) matrix, and emission (N,S) matrix where N is the number of states, and S is the number of symbols. This included special cases (BEGIN, END), and both were smoothed according to add one smoothing.
Then for each query, two (N,Q) matrices were created, one containing the highest log probability to that state, the other containing the position of the previous state (path) that led to that state (Q is the length of query including start/end).
These were calculated the following way:
1. logprobs = take max of log probs on the previous query symbol (N) + transition matrix (N,N) + N matrix (working in log probabilities so + instead of x). If the symbol is unknown, uniform distribution is assumed. For each row the max is taken.
2. the path takes the index of the max logprob in the last query.
Next the paths were forced into the ‘END’ state, by adding on the logprobs to the state one more time.

The path is then built up iteratively, the first obviously being the index of end state, and traversing backwards through the matrix.
For example
query [‘BEGIN’,‘a’, ‘b’ ,’END’]
paths =
[4, 1 , 2, 5]
[4, 1 , 3, 5]
[4, 3 , 0, 5]
[4, 0 , 1, 5]
[4, 1 , 2, 5]
say the maximum logprob is line three [4,3,0,5] (from the last row of log probs matrix).
The path will be [4,1,2,5]

Question 2.

To extend the Viterbi Algorithm to Top K, only a few simple tweaks were needed.
This required for each query to keep track of three matrices:

- log probabilities (N,Q,K)
- paths (N,Q,K)
- the k in which it came from - fromstate (N,Q,K)

Firstly, rather than storing the top log probability, the top K probabilities were stored (with their paths).
Secondly, the k in which the next state was going to was remembered. This allowed for paths to take higher paths for segments and only diverge for smaller sections. For example a path could follow the top log prob for the first 8 states, then diverge. The state would know which k to follow based on the fromstate matrix.
The calculation was quite similar however, for each row of q in the query (i.e. N states), the results had to be sorted. Compared to doing all the calculation in one line as before.
To do this, I created a vector of length (N*K), which was the probability of each K appended. therefore, we could argsort these, grab the appropriate probability, the state is essentially % N, the state in which it came from is clean division // N (K). Therefore, the top k probabilities were added.

Then as before, the paths were iteratively found, the K in which the index was found, was found through fromstate as explained earlier, allowing for the path to move between different k’s.