COMP 9318 Project Report

Viterbi Algorithm for address parsing

z5174903 Jiaquan Xu

z5147730 Xingchen Wang

Part1: Implementation details of Q1.

Part2: Details on how we extended the Viterbi algorithm (Q1) to return the top-k state sequences (for Q2).

Part3: Our approach for the advanced decoding (Q3).

Part 1 Addresses parsing

Let us introduce the implementation details of Q1.
First, we construct the state transition matrix and the emission matrix with 'State_File' and 'Symbol_File' in 'toy_example' as background parameters. The result is shown in the figure:

```
transition
 [[6. 2. 2. 0. 1.]
 [1. 3. 6. 0. 1.]
 [3. 6. 1. 0. 1.]
 [1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

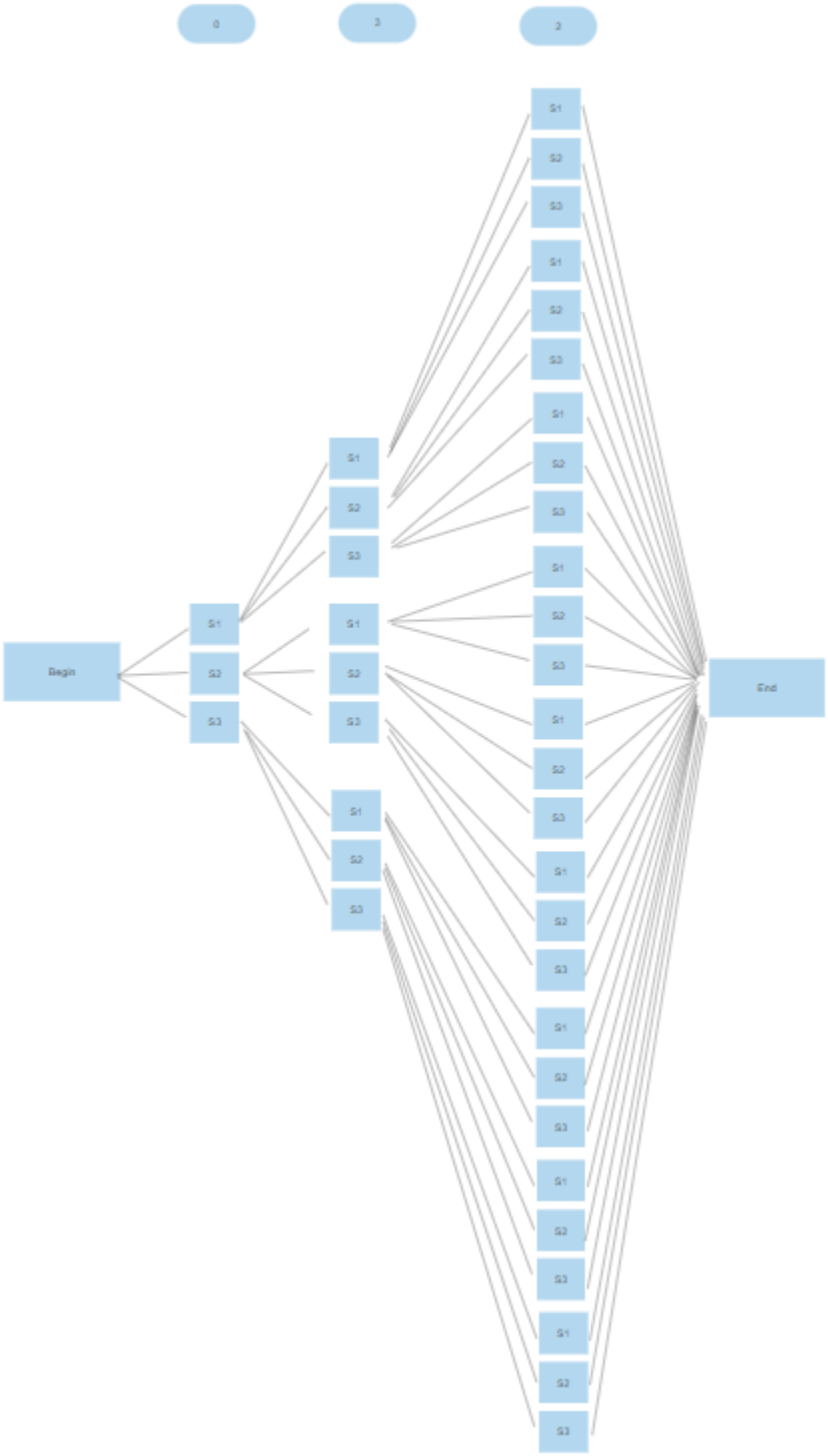
```
emission
 [[3. 2. 1. 0.]
 [1. 3. 2. 0.]
 [1. 1. 4. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Second, in order to avoid the result assigning probability ‘0’ to any state transition not seen in the training data, we perform ‘add-1’ smothing on the state transition matrix and the emission matrix. Get the following results:

```
[[0.46666667 0.2        0.2        0.         0.13333333]
 [0.13333333 0.26666667 0.46666667 0.         0.13333333]
 [0.26666667 0.46666667 0.13333333 0.         0.13333333]
 [0.28571429 0.28571429 0.28571429 0.         0.14285714]
 [0.         0.         0.         0.         0.        ]]
```

```
[[0.4 0.3 0.2 0.1]
 [0.2 0.4 0.3 0.1]
 [0.2 0.2 0.5 0.1]
 [0.  0.  0.  0. ]
 [0.  0.  0.  0. ]]
```

Let's take a look at the first observation sequence 'Red Yellow Blue' of 'Query_File' in 'toy_example'. The ID corresponding to the observation sequence appears as '0 3 2', Q1 asks that we need to calculate the most likely hidden state sequence corresponding to this observation sequence.
From the 'begin' state to the 'end' state, the hidden state corresponding to this observation sequence is actually 3*3*3=27, as shown in the following tree diagram:

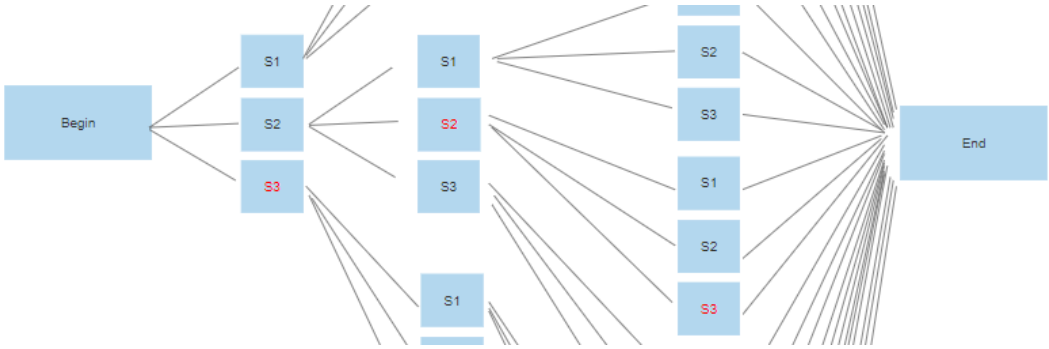


This is all possible outcomes. But in fact, we don't need to calculate these 27 sets of

data. Because we choose according to the Viterbi rules during the calculation process, some paths are terminated. So, every round, we perform up to 3*3=9 operations.
Next, let's select the wanted line in the graph and finally get a path. Because we want the best of them, we have the following Viterbi rules:
max(P(transition(a to b))* P(emission(b to O))*P(a))
'a' and 'b' means the state before and next
'O' means the observed state
Like this:

```
value[i][a] + math.log(transition[a][b]) + math.log(emission[b][query[0]])
```

So, every time we have this calculation and only save the maximum.
At last, we get the max one from 3 results, and the best rote is '2 1 2' .



The result is:

```
[3, 2, 1, 2, 4, -9.397116279119517]
```

Part 2 Top-k parsing

The second problem is the extension of the first problem. Their difference lies in the number of choices in the final path. In the first question, we only choose one that maximizes the result. The second question asks us to choose the maximal result of the first k.
In the process of implementation, let's analyze where we need to expand. Going back to the example above, in the complete tree diagram, we said that there are a total of 27 results, but each round is calculated up to 9 times, because we may have to discard a lot of paths in each round, so the last round is Take the maximum value from the three values to get the final probability and path. The key issue is also in this place. We can't simply select the k maximum values to go down in each round of calculations. If so, the real best path of the k is likely to be simplified in the previous round of operations, so there is no subsequent operation.
Considering that data processing will face a huge amount of data, it is unreasonable to calculate all the paths (27 in the above example) and then sort the first k. So, we still need to simplify the maximum number of operations and ensure that the correct results are not excluded.
Our approach is to select the first k results for each state after each Viterbi rule operation (when less than k, all are selected). The reason for this is that k is retained in comparison to the total number of results from each round, which avoids repetitive

problems (if the k does not achieve the goal, then the n is less likely to be As). 'Repeat' means that, for example, in the second round, this path was due to rule, but in the last round, it should be one of the largest k. In other words, the current results do not represent the end. It similar the dynamic programming problems.

Part 3 Advanced Decoding

This is the result of Question 3 advanced decoding, reducing total number of incorrect labels by 20.

```
compare result:
Q1_label_difference 134
Q3_label_difference 114
```

We compare output sequence of states generated by Viterbi algorithm in Question 1 with Query_Label and turned out that there are 134 difference in total.
Recall that we used the add-1 smoothing method when calculating the transition probability and the emission probability in Question 1. In fact, there are many other smoothing techniques that perform better than add-1 smoothing. Such as Addictive smoothing, Good-Turing estimate, Jelinek-Mercer smoothing, Katz smoothing, Witten-Bell smoothing, **Absolute discounting**, Kneser-Ney smoothing.

In Question 3, we used **Absolute discounting** instead of add-1 smoothing to smooth emission rate. Here is the formula to compute the emission rate by using this strategy.

$$B(state, symbol)\begin{cases} B(state, symbol) without\ smooth - p & if\ B(state, symbol) > 0 \\ \dfrac{vp}{N - v} & if\ B(state, symbol) = 0 \end{cases}$$

B(state, symbol) is the emission probability.
V is the number of symbols that emitted > 0 times in this *state.*
p = 1/(Ts+v), where *Ts* is total number of symbols emitted in *state.*

The reason why this smoothing method has better performance:

For the dev_set test, the number of symbols is over 40 thousand. We assume there is a state that only emits very little symbols with very little times. Emission frequency is like this: E = [1, 2, 3, 0, 0, 0, 0, 0, 0, 0 ……, all others are 0]. If we smooth all symbols with 1, the probability for emitted symbols is 6+3/6+40000, while the probability for never emitted symbol is 40000/6+40000, which is much larger than probability of emitted symbols. This unbalance result would cause poor behavior in comparing emissions. In contrast, by using absolute discounting smoothing, the

proportion of emitted symbols will always be dominated. The distinction between different symbols will also increase.

In conclusion, the method we used in Question 3 Advanced decoding is to change add-1 smoothing method to **Absolute discounting smoothing method**. The Viterbi calculation part remain the same.