COMP9318 Project Report

Team members:

Haiyu LYU(z5142340)

Yixi WEI(z5176186)

Q1: Viterbi algorithm for Address Parsing

Transition probability

First of all to calculate the transition probabilities, we wrote a function called `calculateTranProbability(State_File)` for this part.

We get the following data by traversing `State_File`:

1. `numOfStates`: The number of states, including BEGIN and END.
2. `states`: The name of the states, such as 'S1', 'S2'...
3. `transList`: Record the number of transition between each states.
4. `countTransTimes`: Record the number of times a state appears in total.

In order to store the transition probabilities between each state, we use the two-dimensional dictionary `transProbability`.

We can use the formula

$$A[i,j] = \frac{n(i,j)+1}{n(i)+N-1}$$

to calculate these probabilities and and put them in the `transProbability`.

n(i,j) corresponds to the data in `transList`.

n(i) the number of times from `countTransTimes`.

N is `numOfStates`.

In this part we don't smooth the transition probabilities.

Emission probability

We use `calculateEmissProbability(Symbol_File,states)` to calculate this part.

Traversing file `Symbol_File` we can get the following data:

1. `numOfSymbol`: The number of symbols.
2. `symbol:`: The name of the states, such as 'RED', 'GREEN'...
3. `emissList`: Record the number of symbol in each states.
4. `countEmissTimes`: Record total symbols in a state.

We also use 2-D dictionary to save emission probabilities, and the dictionary called `emissProbability`.

We can also use following formula to generate `emissProbability`

$$B[i,j] = \frac{n(i,j)+1}{n(i)+M-1}$$

n(i,j): data from `emissList`.

n(i): data from `countEmissTimes`.

M: the number of symbol in the `Symbol_File`.

Viterbi algorithm

We need 3 lists:

1. `probList`: Store all possible probability.
2. `pathList`: Store all possible paths.
3. `results`: Store the optimal result for each query, including path and probability.

We need to traverse the file `Query_File` and we have to initialize `probList` and `pathListeach` each time. It is necessary that using `-/&,()\s` to split each query. In this way, the length of each sequence is correct. For each query, we have different symbols. We should traverse every symbol in the query.

using formula to calculate max probabilities for state j of symbol s

$$\begin{gathered} prob(s,j) = max[prob(s-1,0) * transProb(0,j) * emissProb(j,s), \\ prob(s-1,1) * transProb(1,j) * emissProb(j,s), \\ \ldots\ldots\ldots] \end{gathered}$$

prob(s-1,0): the maximum probability of the previous symbol in the state 0. We can get this from `probList[s-1][0]`.

transProb(0,j): the transition probability of state 0 and state j. `transProbability[0][j]`

emissProb(j,s): the emission probability of symbol s in state j. `emissProbability[j][s]`

However, there are 2 situations that require attention.

1. There is no transition probability between state *i* and state *j*. We can use smoothing method to solve this problem:

$$A[i,j] = \frac{1}{n(i)+N-1}$$

2. There is no symbol *s* in state *j* :

$$B[s,j] = \frac{1}{n(i)+M-1}$$

Hence, we can use the above algorithm to calculate a maximum probability and put it in `probList[s][j]` and save the optimal path in `pathList[s][j]`.

Finally we need to find the maximum probability in `probList[-1]` and its corresponding path in `pathList[-1]`. We change our output form to `[path, probability]` and put it in `results`. The last job for this query is to initialize lists `probList` and `pathList`, then go to the next query.

After traversing all the queries, we output results.

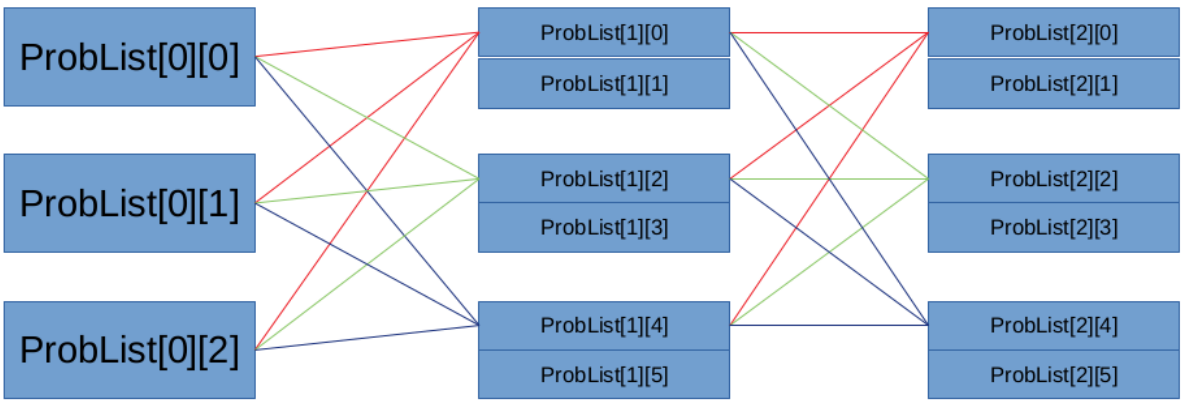
Q1 output:

```
306
307 State_File = './toy_example/State_File'
308 Symbol_File = './toy_example/Symbol_File'
309 Query_File = './toy_example/Query_File'
310 viterbi_result_1 = viterbi_algorithm(State_File, Symbol_File, Query_File)
311 for row in viterbi_result_1:
312     print(row)
```

```
[3, 0, 0, 1, 2, 4, -9.843403381747937]
[3, 2, 1, 2, 4, -9.397116279119517]
```

Q2: Extending Viterbi for top-k Parsing

Q2 only needs to be modified on the basis of Q1, because the algorithm of Q1 is TOP-1. If we want to calculate TOP-K, it is necessary to save K values in each state. For example, if we are computing TOP-2, we can clearly see this algorithm in the following graph.



In code part, we use `probLevel` to save all these data. The structure of `probLevel[s][i]` is:

s: is the level of data

i: is the i-th data in s level, this is a data set. The first data in `probLevel[s][i]` is probability, like `probLevel[1][5][0]` = 0.0023133. The second data in `probLevel[s][i]` is the path. For example, `probLevel[3][2][1]` =[1,0,4].

Finally, we need to sort the probabilities in `probLevel[-1][i]` to find the largest K, record their path and probability as results.

Q2 output:

```
306
307 State_File = './toy_example/State_File'
308 Symbol_File = './toy_example/Symbol_File'
309 Query_File = './toy_example/Query_File'
310 viterbi_result_2 = top_k_viterbi(State_File, Symbol_File, Query_File,3)
311 for row in viterbi_result_2:
312     print(row)
```

```
[3, 0, 0, 1, 2, 4, -9.843403381747937]
[3, 0, 0, 0, 2, 4, -10.131085454199718]
[3, 0, 0, 0, 0, 4, -10.20007832568667]
[3, 2, 1, 2, 4, -9.397116279119517]
[3, 0, 0, 2, 4, -9.551266958946776]
[3, 0, 1, 2, 4, -9.551266958946776]
```

Q3:Advanced Decoding

By examining the query file, we found the following 116 unknown labels.

```
['MBF', 'Dumbuoy', 'RMB', 'HACC', 'Holistic', 'in', 'Lot3', 'U3', 'U5', 'Shp1', 'P.O', '6196', 'St.Kilda', 'Ganthe
aume', 'U10', 'U1', 'Burrinjuck', 'Unit5', 'Pontus', 'UNSW', 'U9', 'RMB', 'Municipal', 'Shp4', 'PMB', 'U1', 'Lvl
2', 'Ste1', 'Ste306b', 'U2', 'Locked', 'Kiosk', 'Utility', 'Services', 'Aprt2', 'Ste4', 'GPO', 'Un', 'Ossary', 'Sh
p2B', 'Shp211', 'U7', 'Surgery', 'Shp3b', 'Pathology', 'Nocklolds', 'Shp10', 'Shp1', 'Shp7', 'GPO', '5029', '800
4', 'Lvl1', 'U3', 'Shp1', 'Kiosk', 'Family', 'Institute', 'suite101', 'lvl', 'SCG', '3th', 'Ksk210', '1g', 'Lot1',
'Muggletons', '2797', '4thFlr', 'L6', 'Cootralantra', 'L1', '1stFloor', '2J', 'GPO', '805B', 'Kalamia', 'Under',
'GPO', '1623', '420a', 'Basement', 'Highways', 'Shp2', 'Linmore', 'Benmore', 'E28', 'Lv', 'Level11', 'Lvl1', 'Cumm
ingham', 'GPO', '8060', 'Lvl1', 'Exit', 'Lvl1', 'Hanger', 'Lot71', 'Lawgalls', 'Hangar', 'Lot22', 'Grahamstown',
'Locked', 'China', 'Kiosk', 'Grundy', 'Basement', 'CAE', 'Biggins', 'Hynam', '5262', 'U19', 'Casewell', 'Postal',
'Telecommunications', 'Alloah', 'U164']
```

For example, we can see 'U2' as UnitNumber. Hence, if we get a unknown label 'U2', we can increase the weight the UnitNumber to 1000. However, compared to Q1, we do not consider this case.

Q3 output:

```
293 viterbi_result_3 = advanced_decoding(State_File, Symbol_File, Query_File)
294
295 Query_File1 = './dev_set/Query_Label'
296 queryDataset = open(Query_File1).readlines()
297 ll = []
298 key = 0
299 for q in range(queryDataset.__len__()):
300     xx= re.split(r'([-/&,()\s])',queryDataset[q])
301     queryLine = []
302     for e in xx:
303         if e!=' ' and e !='\n' and len(e)!=0:
304             queryLine.append(int(e))
305     ll.append(queryLine)
306 for i in range(len(viterbi_result)):
307     for j in range(len(viterbi_result[i])):
308         if viterbi_result[i][j] != ll[i][j]:
309             key = key + 1
310 print(f'Different labels:',key)
```

```
Different labels: 115
```

115 < 134 - margin