50.007 Machine Learning Project

Zhou Yutong 1003704
Prashanth Nair 1003639
Joey Richie L Tan 1003599

Results are already run and can be viewed from the jupyter folder.

Part 1:
Annotations done individually.

Part 2:
To write a function that estimates the emission parameters, we are required to first load the training set into a dataframe. For each word, we place it in a dictionary, with the row having the input of the number of unique y state, column as each word x, where all entries are 0. We will then calculate the following:

- y_count_dic storing count(y) in a dictionary
- emission_count_table storing all count(y → x) in a dataframe
- emission_param_table storing all the emission parameters in a dataframe

The emission_param_table follows the formula of emission_count_table/y_count_dic,

$$e(x|y) = \frac{\texttt{Count}(y \rightarrow x)}{\texttt{Count}(y)}$$

Due to the issue of certain words that appear in the test set but not in the training set, a special token word #UNK# is introduced. This way, a column for UNK is added at the end of the table. We would then perform the same steps as earlier, generating y_count_dic, emission_count_table and emission_param_table with k = 0.5.

Iterating through every word in the set, replacing the unknown words in the test set with #UNK#, we would then be able to generate the highest emission probability and compare it with the answer to verify the performance of the predictions.

Our results are as follows:

EN

```
#Entity in gold data: 13179
#Entity in prediction: 18650

#Correct Entity : 9542
Entity  precision: 0.5116
Entity  recall: 0.7240
Entity  F: 0.5996

#Correct Sentiment : 8456
Sentiment  precision: 0.4534
Sentiment  recall: 0.6416
Sentiment  F: 0.5313
```

CN

```
#Entity in gold data: 700
#Entity in prediction: 4248

#Correct Entity : 345
Entity  precision: 0.0812
Entity  recall: 0.4929
Entity  F: 0.1395

#Correct Sentiment : 167
Sentiment  precision: 0.0393
Sentiment  recall: 0.2386
Sentiment  F: 0.0675
```

SG

```
#Entity in gold data: 4301
#Entity in prediction: 12237

#Correct Entity : 2386
Entity  precision: 0.1950
Entity  recall: 0.5548
Entity  F: 0.2885

#Correct Sentiment : 1531
Sentiment  precision: 0.1251
Sentiment  recall: 0.3560
Sentiment  F: 0.1851
```

Part 3:

Here we are required to create a function that estimates the transition parameters. Hence instead of focusing of the emission state of earlier, we would be looking at the transition state. The loading of the data slightly differs in the sense that each "next line" or "\n" will be replaced with a "START" and a "STOP", to facilitate the identification of when the sentence starts and stops.

For the transition_param_table, the rows will represent the current state and the columns will represent the next state, hence the "START" state would appear in rows while "STOP" appears in columns. We would then iterate through the states from "START" till the state before "STOP", as there would not be any transition after "STOP".

The formula used here is as follows:

$$q(y_i|y_{i-1}) = \frac{\text{Count}(y_{i-1}, y_i)}{\text{Count}(y_{i-1})}$$

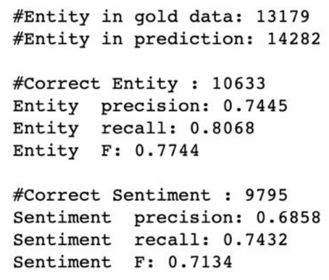
Following which is the introduction of the Viterbi algorithm. Similarly, we would first create a 2D table, pi_table and initialize the values as 0 at the beginning. We would add a start_node and stop_node in the pi_table for the transition from 0 to Start and n+1 to Stop respectively. For the forward moving algorithm, we initialize a temp_result table to store the pi values of each position and iterate through each state and their values using the following formula. The value is calculated using the value of the previous layer, multiplied by the transition probability to the current state, multiplied by the emission probability of the current state. After acquiring the max value for each state at position n, we would then transit to the state at n+1 position, “STOP”. This would hence give us the optimum path for the forward algorithm.

The backward algorithm transits from “STOP” all the way to “START”. This includes iterating through the states from STOP to all the states at position n and acquiring the argmax from it. Thereafter, we would iterate through the values of each column to attain the optimum path from position n-1 to 0.

Following the calculation of the forward and backward portions of the algorithm, we will then be able to observe the most optimum path overall, concluding the Viterbi algorithm.

Our results are as follow:

EN

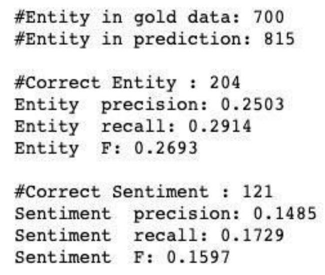


```
#Entity in gold data: 13179
#Entity in prediction: 14282

#Correct Entity : 10633
Entity  precision: 0.7445
Entity  recall: 0.8068
Entity  F: 0.7744

#Correct Sentiment : 9795
Sentiment  precision: 0.6858
Sentiment  recall: 0.7432
Sentiment  F: 0.7134
```

CN

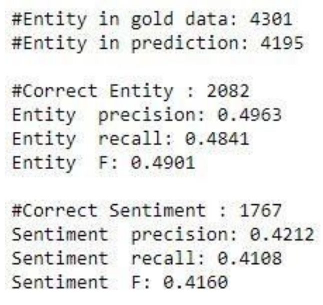


```
#Entity in gold data: 700
#Entity in prediction: 815

#Correct Entity : 204
Entity  precision: 0.2503
Entity  recall: 0.2914
Entity  F: 0.2693

#Correct Sentiment : 121
Sentiment  precision: 0.1485
Sentiment  recall: 0.1729
Sentiment  F: 0.1597
```

SG



```
#Entity in gold data: 4301
#Entity in prediction: 4195

#Correct Entity : 2082
Entity  precision: 0.4963
Entity  recall: 0.4841
Entity  F: 0.4901

#Correct Sentiment : 1767
Sentiment  precision: 0.4212
Sentiment  recall: 0.4108
Sentiment  F: 0.4160
```

Part 4:
Instead of the normal Viterbi algorithm, the question is asking to find the 3rd best output sequence. The usual output from the Viterbi algorithm is the best output sequence, which is

of the maximum optimum. Hence, via dynamic programming, we would list out the top 3 sequences, and henceforth attain the 3rd best sequence.

We will initialize an empty table, pi_table, with the state as the column value and word as the row value.

From layer 0 to layer 1, it starts off with the transition from the “START” node to the next state. Since there is only one transition possibility from layer 0 to layer 1, layer 1 will only intake in values of a 1D array. To obtain the value of each node, we will multiply the value from start_node (initialized as 1), the transition parameter from “START” to node layer 1 and emission parameter. The result is then appended to the table.

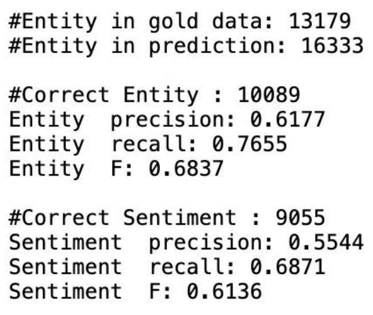
From layer 1 to 2, the input value would be a 1D array. To obtain the value for each node, we will multiply the 1D input value with the transition parameter from each node in layer 1 and the emission parameter for the node. The result would then output a 3D array, saving the top 3 values.

From layer 3 to n-1, we would input in the values as a 3D array from the previous layer. Each node will follow the same formula, multiplying the value of the previous node with the transition parameter from layer k to k +1, and emission parameter from each of the 3 input values in the 3D array. The output value would be a 3D array as well, saving the top 3 values.

From layer n-1 to n, the input value would also be a 3D array from layer n-1. The result calculated would be the multiplication of the 3D input array and the transition parameter from layer n-1 to n. The output value would also be a 3D array.

Backtracking, we would move from the “STOP” node to the nodes in layer n-1. With the 3D array output from the forward propagation, we would take the 3rd value and save it in a list optimum_path. We would iterate through each node, to check for the match of the value in optimum_path. Once found, the node would be saved in optimum_path as well. This backward propagation continues from layer n-1 to the 3rd layer, inputting and outputting 3D arrays each. From the 3rd layer to the 2nd layer, the input would be a 3D array while the output is a 1D array. Finally for layer 2 to layer 1, both would be a 1D array.

Hence by following the 3rd value in the array, we would be able to obtain the 3rd best node for each path and generate the 3rd best output sequence.



```
#Entity in gold data: 13179
#Entity in prediction: 16333

#Correct Entity : 10089
Entity  precision: 0.6177
Entity  recall: 0.7655
Entity  F: 0.6837

#Correct Sentiment : 9055
Sentiment  precision: 0.5544
Sentiment  recall: 0.6871
Sentiment  F: 0.6136
```

<u>Part 5:</u>

For the design challenge, our initial idea was inspired by the homework, to use the previous 2 states to calculate the transition and emission parameters, and hopefully improve the result. The algorithm is as follows:

1. To initialize 2 ‘START” states at position -1 and 0
2. Moving forward recursively from 1 to n, we would find the max value given the multiplication of the previous layer’s value, the transition parameter from the 2 states before that and the emission parameter. Every node is to be iterated.
3. From layer n to layer n+1 or “STOP” state, we would calculate the maximum value of the multiplication of values from layer n and the transition parameter from each node to the “STOP” node.
4. We would then backtrack to attain the argmax value from “STOP” to layer n, followed by n-1 all the way to the first layer.