Part 2

This goal of part 2 is solely using the emission parameter to predict the states of each word. After loading the training set, we calculate the number that a state appears (CountY) and the number of a specific state and word appear on the same line (CountY\_X), then obtain the emission probability by calculating CountY\_X/CountY.

Next, we tweak the training set a little bit by adding a word ‘#UNK#’ for every state. Then we apply the same algorithm as mentioned above. There are some minor adjustment to do afterwards. First, every entry in the CountY column should be subtracted by 0.5. k is 0.5 in this case, but we actually added 1 because of the word ‘#UNK#’ added for each state, forming column CountY+k. Second, after calculating the emission probabilty using CountY\_X/CountY+k, every row with word ‘#UNK# should be divided by half because their CountY\_X is 1 which should be 0.5.

At last, we just iterate every word in the test set, changing every word that is not in the training set to ‘#UNK#’ and then find the state that has the highest emission probability and compare it to the answer to see how the model performs.

The result is as follow:

EN:

Precision:0.6208779304068986

Recall:0.6315232722143864

F1:0.6261553588987218

CN:

Precision:0.06893382352941177

Recall:0.2674750356633381

F1:0.10961707103186204

SG:

Precision:0.2788090195589884

Recall:0.5057055699920913

F1:0.3594458943987151

Part 3

This goal of part 3 is to use Viterbi Algorithm to predict the states of each word. Therefore, we need to calculate the transition probability first. The training set is loaded, and every blank line is filled with ‘Nil Nil’. The first ‘Nil’ is the word while the second ‘Nil’ is the state. The ‘Nil’ state here will serve as both the START state and the STOP state. ‘Nil Nil’ ensures that the emission probability is 1 for word ‘Nil’ and 0 for all other words so it won’t affect likelihood.

The we iterate through the training set to see how many times that a state appears (CountYi) and how many times it transfer to a certain state (CountYi\_j), then use CountYi/CountYi\_j to calculate the transition probability.

Next is the using the Viterbi Algorithm to predict the states. In order to deal with the underflow of value, we divided the test set by sentence. In other words, we use the Viterbi Algorithm to predict one sentence at once instead the entire dataset. With less word at one shot, the likelihood is a higher value. For every sentence in the test set, we add a line ‘Nil Nil’ to both the start and the end of the sentence so that it is compatible to the algorithm.

For every word in the input, the algorithm first select the emission probability of the word for every state, which is a column vector, then for every possible state, select the transition probability to this state, which is another column vector, multiply with the emission probability that is corresponding to the state, then multiply with the likelihood of previous layer, which is another column vector. Select the highest value for each result and use as the likelihood for the next layer’s calculation. At last, back track to find every state we have chosen and convert that to the final output. The whole process is just simply following the Viterbi Algorithm, nothing else.

At last, we compare the result to the answer and the result is as follow:

EN:

Precision:0.758757753123089

Recall:0.7656470380818053

F1:0.7621868281339126

CN:  
Precision:0.1626267078007933

Recall:0.2631954350927247

F1:0.20103514028874966

SG:

Precision:0.2551662174303684

Recall:0.38504123827816067

F1:0.30693024721934525