Named Entity Recognition assignment:

I used the same implementation that I used for POS tagging assignment. The changes that I made to that are as follows,

- I used Laplace smoothing (add-1 smoothing) instead of Kneser-ney smoothing for the transition probabilities. Tried to remove this smoothing, but one sentence forced me to add it. Since I made this change I removed the unwanted counts required for the Kneser-ney smoothing.
- Removed the smoothing part for the start probabilities.
- Added an <end> tag instead of using '.' as a delimiter. Since '.' (Period) cannot be used as sentence completion indicator in this dataset, as it is used in many other scenarios.
- Added rules and conditions in all parts of the code to support the edge cases.
- Since the product of lot of probabilities of the words in the long sentences affect the capacity of Python handling very small numbers, there is library available in Python known as Decimal, I used it for managing the small numbers, so I didn't opt to logarithmic calculations.

The following is the rest of the methodology that has been taken from the previous assignment for further clarifications,

From the given training set, I extracted the data horizontally into a 2D list, taking new lines as end tags. Then formed a dictionary within a dictionary that has words with their tags and their counts. Word frequency, tag frequency, start tag frequency, sentence count and the bigram count were computed. Later found out the words with low frequency (say 1) and updated those words with the word 'UNK' in the dictionary, the initial list, and their respective word frequencies.

After extracting all the required counts from the training data, the next step is to generate the required probability estimates for the model. The emission probability was calculated for every word, as it is usually defined (The count of the word given the tag over the frequency of the tag), computed the transition probabilities between the tags (The count of the tags occurring together over the count of the first tag). To avoid zero probabilities, Laplace smoothing (add 1 smoothing) was used for it. Start probabilities (probability that a tag being a start tag) are also computed. The start probability is the count of each tag being a start of a sentence over the number of sentences. Smoothing is not applied over the emission probabilities and the start probabilities.

The Viterbi function was implemented as per the standard algorithm. I used a dictionary with two keys for Viterbi probabilities at each word with respect to its tags, where the first and second key denote the tag and the word position in the sentence respectively. A Similar notation was used for the back pointer as well. When a sentence has words that are not in our dictionary, they are changed to 'UNK' at the beginning of the function itself. When a sentence is passed to the function as a list of words, it returns a list of predicted tags for the corresponding words in it.

The given test set is extracted into a list similar to the training data and is passed to the Viterbi function sentence by sentence, and the tags are obtained accordingly and appended to a list. The test-set data and the obtained tags are printed in a text file in the required format.