POS tagging assignment report:

I started my solution approach with the suggested baseline approach which gave me an accuracy of 93% and the advanced approach (i.e., the Viterbi decoder) is implemented as follows.

From the given BERP training set, I extracted the data horizontally into a 2D list, skipping the blank lines. 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 the word frequency.

I did not use a start-tag to denote the start of the sentence, instead used the dot symbol as a delimiter, which ends the previous sentence and thereby giving a start to the next sentence. 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, smoothing algorithms have to be applied over the transition probabilities. Initially, Laplace smoothing (add 1 smoothing) was used for it. Start probabilities (probability that a tag being a start tag) are also smoothed similarly. 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.

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. I assessed my system using K fold (where **K=10**) and obtained an average accuracy of 94.57% with a variance of 0.56. I preferred Kneser-ney smoothing the transition probabilities over Laplace smoothing since it gave a better average accuracy of **95.69%**. To do that we need the counts for each tag type being a first tag type/ a second tag type in a bigram and also the total number of bigram types. Lambda was set to 0.5 if the frequency of the bigram is 1 or else it is set to 0.75. As the obtained accuracy is better than the baseline approach, this method of implementation is a reasonable one.

Finally, the whole training data is used to compute the counts, probabilities and to train the model. The given test set is extracted into a list 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. I believe that this model will have the shortest runtime and with competing accuracy supported by the promising and authentic results for any unseen data, provided it satisfies my model's settings.