NLP - POS Tagger

Initially, a baseline system was developed which basically classified a word as a tag when it had the highest occurrence count for that word in the training data. The accuracy of such a system was a mere 44% on an average. The reason for such a low accuracy with the baseline system was because this system did not handle unknown words. The given data set was divided into training data and test data with an 80-20 random split, trained using the training data and then the test data was classified. This process was repeated multiple times and an average of the accuracies achieved in each classification step was recorded as the final accuracy of the system.

Then, the Viterbi decoder algorithm was implemented by first building the emission probability matrix and transition probability matrix. The data structure used here was a python's dictionary of dictionaries with tags being rows and words of the training data being columns in the emission matrix. The transition matrix was a NxN matrix with PoS tags. However, the data structure used for the Viterbi matrix was a dictionary of dictionaries with a tuple - the values of the inner dictionary were tuples that contained the Viterbi values of the word given tag and the PoS tag from the previous iteration that lead to optimal Viterbi value in the current iteration.

Kneser Ney smoothing was then applied to the transition probability matrix to ensure that the final Viterbi value does not converge to a zero.

Unknown words were handled by replacing the words in the test data that did not exist in the vocabulary of the training data with a pseudo-word - '<UNK>'. I then set emission probabilities such that each state has the equal probability of emitting this word and hence unknown words are tagged purely on context. This mechanism handles unknown words fairly well, where each word is classified based on the tag of the previous word. However, there could be better mechanisms used - we could have multiple pseudo-words representing different types of unknown words, for example - <firstLetterCap>, <fourDigitNumeral>, etc. We can then replace every "stop-word" in the training data with their respective pseudo-word. This mechanism would predict the tags of unknown words in a far better way than the mechanism implemented.

The current system gave an accuracy of 92% on an average with an 80-20 split training and test data.