POS TAGGER

STEPS to Solve POS Tagger Task:

1. Both train and test datasets are in the form of a txt file with columns ['sent_id', 'text', 'word', 'normalized_word', 'POS_tag', 'head_word_index', 'dependency_relation'], So I have iterated over a txt file and convert it into a data frame for easy usage during the whole assignment.
2. I first calculated the frequency of each word (original word not normalized word) in the dataset and stored it. This will be used at the time of calculation of emission probability for word-tag pair.
3. I have stored all unique POS tags in a training dataset and their corresponding frequency. This will be used at the time of calculation of emission probability, and transition probability. There are a total of **47** unique POS tags in the training dataset and the same tags are also present in the test dataset.
4. I have also calculated the frequency of each of the 47 POS tags to appear at the start of a sentence, this will be required in the Viterbi algorithm for the prediction of the POS tag of 1st token as there is no other transition probability so we have to take the tag having maximum start probability. If any tag never appears at the start of a sentence then I consider its frequency as 0.
5. Now, we have to calculate the frequency of each (tag, tag) pair that is present in the training dataset. This is required at the time of calculation of transition probability from 1 step to another step in the Viterbi algorithm.
6. Also, similarly, we have also found the frequency of each (word, tag) pair in the training dataset. This is required at the time of calculation of the emission probability of a word given its POS tag.
7. Tag start probability is found for each tag by dividing the tag start frequency of the corresponding tag by the total No. of sentences (2000) as it shows the probability of each tag to come at the start of a sentence.
8. Transition Probability is found for the pair of Tags (t1, t2) using the frequency of the (t1,t2) pair which we have already found and divided by the frequency of tag (t1).

   P(t2 | t1) = Count(t1 , t2) / Count (t1)

   I have created a transition matrix of tags so that accessing the transition probability at the time of the Viterbi algorithm will be easy and I can directly access values from this matrix.
9. Similarly, the emission probability is found for the (w, t) pair using the frequency of the corresponding (w,t) pair and dividing it by the frequency of the tag (t).

   P(w |t) = Count(t, w) / Count (t)

10. Now, from the pseudocode given for the Viterbi algorithm, I have defined the function to write the code of the Viterbi algorithm using a dynamic programming approach where I first initialized the scores table to store the score and the tags table to store the corresponding optimal tags at each step. The for word 0, score, and tags table is initialized using the tag start probability and emission probability. In the iteration step, we are iterating on the remaining words in the sentence, finding the best Tag to come to a specific state of step from the particular step from the previous step and we have used values that are already stored in the scores table so that repeated computation is avoided. Then, a path with the highest probability is found, and using backtracking we have to find the corresponding POS tags sequence with maximum probability as predicted by the Viterbi algorithm.
11. I have iterated on train data and found the optimal POS tag sequence for each sentence in training data as predicted by the Viterbi algorithm and stored them in a data frame so as to output in the mentioned format in a .tsv file.
12. Now, by storing the true POS tag and POS tag predicted by Viterbi for each token in the sentence in training data, I have found evaluation metrics like Precision, Accuracy, Recall, and F1-Score. The value of these metrics for training data are as follows:

```
Precision Score: 0.9807964698191327
Recall Score: 0.9807964698191327
F-score Score: 0.9807964698191327
Accuracy Score: 0.9807964698191327
```

    The **98.07%** accuracy on training data is a good showing that Viterbi was correctly able to capture the relationship between the words and the dataset is also well balanced so as to get good probability values and good evaluation metrics. High precision and recall values indicate that the model is both accurate and comprehensive in its POS tagging, with very few false positives or false negatives and the model is not biased towards predicting certain classes more accurately at the expense of others.
13. Now, at the time of testing, there are some tokens in the test data that are not present in the train data so we don't have the emission probability values for those tokens. So, we have done Add-1-Smoothing so that the probability for those tokens will be non-zero and we can use them in sentences. In the test dataset, out of 2184 tokens, 1724 are present in the training dataset also while 460 tokens are not present in the training dataset and these are completely unseen tokens. Add-1-smoothing is applied to all the tokens in the test dataset and not only to unseen tokens as mentioned in the assignment, so in total Add-1-smoothing is done for all **2184** tokens present in the test dataset.
14. As all tags in the test dataset are also present in the training dataset, so we do not need to calculate any transition probability for a pair of tags again.

15. At the time of using the Viterbi algorithm for the test, the only change is that we are using emission probabilities obtained by Add-1-smoothing while tag start probability and transition probability are used that we have calculated for a train set.
16. Similar to the train set, I ran Viterbi on the test set, stored output in a format that is to be submitted, created 2 lists of True Tags and Predicted Tags for each word, and used these 2 lists to calculate the evaluation metrics for the test set. The results of the evaluation metrics on the test set are as follows:

```
Precision Score: 0.7106227106227107
Recall Score: 0.7106227106227107
F-score Score: 0.7106227106227107
Accuracy Score: 0.7106227106227107
```

The **71.2%** of test accuracy is good but it is lower as compared to the 98% of train accuracy. Some of the key reasons possible for this low accuracy are as follows:

A. The model may be overfitting to train data, reducing its ability to generalize to new unseen test data. This could be due to the test set containing more diverse or complex sentences, or grammatical structures not well-represented in the training set.
B. One of the main things is that, out of 2184 tokens in test data, 460 tokens are not present in train data, so we use Add-1-smoothing for all tokens for train data. But these will not help much for the unseen tokens and they don't gain any significant probability weightage causing less effectiveness of these tokens in the Viterbi task and thereby reducing their contribution in order to get the best POS sequence. If some of these tokens' POS tags are trivial, as we don't have sufficient probability value for them, there is a chance that these tokens are ignored at the time of testing and other tokens that appear in train data and relatively high probability values are considered.
C. One other reason is that train data has 2000 sentences consisting of a total of more than 36000 tokens, but for better results on test data, we want the train data should be larger than this, i,e. Is more than 10,000 sentences and 1,00,000 tokens so it is better for us to calculate the probability also there will be very less no. of tokens in test data that are not present in train data helping the model to get better results.
D. Instead of Add-1-Smoothing, if we use any other technique that gives probability based on certain kinds of feature similarity of unseen tokens with seen tokens in the dataset using some kind of embedding extraction of words, we will be able to get better results on the test data.