# NLP ASSIGNMENT 1

Om Khare
112003066
Div 1, B4

**Dataset Used:**

1. Penn Treebank Corpus: The model is trained on the treebank dataset by nltk, which follows the Penn Treebank tagset for labeling.
2. Conll2000 Corpus: This is another dataset that uses Penn Treebank tagset for labelling. The code uses the first 100 sentences of this corpus for testing (test_data).

**List of variables used in the Viterbi algorithm:**

1. tags: A list of all tags from the training data.
2. words: A list of all words from the training data.
3. tag_bigrams: A list of consecutive tag pairs (bigrams) from the training data.
4. tag_freq: The frequency of tags.
5. tag_bigram_freq: The frequency of tag bigrams.
6. word_tag_freq: The frequency of word-tag pairs.

**Functions:**

1. transition_probability(t2, t1): [ $P( t_i | t_i\text{-}1 )$ ]
    - This function calculates the probability of a tag t2 following a tag t1.
    - It returns the ratio of the frequency of the bigram (t1, t2) to the frequency of the tag t1.

2. wordtag_probability(word, tag): [ $P( w_i | t_i )$ ]
    - This function calculates the probability of a word being associated with a particular tag.
    - It uses a smoothing factor k to handle cases where a word-tag pair is not present in the training data.
    - The probability is calculated as the ratio of the frequency of the word-tag pair to the frequency of the tag, adjusted with the smoothing factor.

### Viterbi Algorithm:

The Viterbi algorithm uses a dynamic programming algorithm approach for sequence labeling. The algorithm finds the most probable sequence of tags for a given sentence.

Steps:

1. Initialization: For each tag, the algorithm initializes the probability of the tag being the start tag of the sentence. This is done based on the lexical probability of the first word of the sentence and the initial probability of the tag.
2. Recursion: For each subsequent word in the sentence, the algorithm calculates the maximum probability of reaching that word with a particular tag, considering all possible previous tags. It also keeps track of the tag that resulted in this maximum probability (backpointer).
3. Termination: Once the end of the sentence is reached, the algorithm identifies the tag with the highest probability for the last word. It then backtracks using the backpointers to find the most probable sequence of tags for the entire sentence.

### Testing the Viterbi Algorithm:

1. A test sentence is extracted from the test_data.
2. The Viterbi algorithm is applied to this sentence to predict the sequence of tags.
3. The predicted sequence of tags is printed in the output file.

## Limitations

While coding and implementing the Viterbi algorithm, several limitations can be observed:

1. The Viterbi algorithm relies on observed probabilities from the training data. If a word-tag pair or a tag bigram hasn't been seen in the training data, it will have a probability of zero. This can lead to zero probabilities for entire sequences, making the algorithm unable to tag certain sentences. In the provided code, this is addressed using a smoothing factor k, but choosing an appropriate value for k can be challenging.
2. The Viterbi algorithm has a time complexity of O(T * N^2), where T is the length of the sentence and N is the number of unique tags. For long sentences or tag sets with many unique tags, the algorithm can be computationally expensive.
3. The algorithm uses dynamic programming and requires storing probabilities and backpointers for each word-tag pair in the sentence. This can lead to high memory usage, especially for long sentences.
4. The Viterbi algorithm assumes that the probability of a word being tagged with a particular tag only depends on the previous tag. This is a simplification, as in reality, the context might span more than just one previous word or tag.

5. The algorithm's performance is heavily dependent on the training data. If the training data is not representative of the sentences the algorithm will encounter in real-world scenarios, the performance can be suboptimal.
6. Words that were not present in the training data but appear in the test data can be problematic. The provided code uses smoothing to handle such cases, but this is a heuristic, and there's no guarantee it will always work well.
7. The Viterbi algorithm focuses on the syntactic structure of sentences (i.e., the sequence of tags) but doesn't consider semantic information. This means it might not always make the most semantically appropriate tagging decisions.