This heuristic file included three stages. The first stage is processing the incoming training files and constructing the HMM. The second stage is to use the Viterbi algorithm to predict the probability of the words in the test file. The third stage is some optimization techniques that are applied to enhance both performance and accuracy.

Stage 1: Making the HMM Tagger

During the training phase, I constructed the initial probabilities (76x1) matrix, transition probability (76x76) matrix and emission probability (76x len(unique_word)) matrix. These matrices set the base for the following Viterbi Algorithm. The number 76 comes from all the possible POS tags that are available. If it is the ambiguous tag, only one tag will be saved (ie. xxx-yyy is the same as yyy-xxx and will be save once). The first step is to break the whole paragraph into sentences. The code not only checks if the sentence ends with just (. ? !), but it also checks with quotation marks. For example, the start word and the end word of the following sentences are different: ***"Hi, James."***, ***I am good.*** , ***"Good morning.*** and ***Today is sunny."*** . After breaking the whole word paragraph into each sentence. The initial probability matrix will be constructed from the probability of accuracy of each tag based on all the words of the starting sentences. The transition probability will be constructed from the probability of the second word (idx = 1) from each sentence and saved the accuracy of P(tag_t | tag_t-1). The emission probability will be constructed from the probability of each tag of occurrence per unique word.

Stage 2: Viterbi Algorithm

For each timestamp t, there are several steps that will be done in the Viterbi Algorithm. S represents each possible tag, and e represents each word that is needed to predict the tag.
1. calculate the base case t=0, P(S0 | e0) = P(S0) * P(e0 | S0).
2. for each time stamp t and all previous tag type St-1, calculate P(St | e0 ^ … ^ et) = MAX (P(St-1 | e0 ^ … ^ et-1) * P(St | St-1) * P(et | St) , St-1)

The probability will be normalized with its given previous tag after each time stamp's calculation. This algorithm not only learns from the training occurrence per word but also can learn from the trend of the sentence to predict the most possible probability of the tag for that word.

Stage 3: Optimization

There are two optimization methods that are applied. The first one is for efficiency. Unlike the previous labs, I use NumPy 2darray as the data structure to store the data. The Numpy library is very useful in this lab. Since after each time stamp of the Viterbi process (after predicting each word) and after the HMM construction, the probability needs to be normalized by the sum of each row as well as the argmax functionality in the Viterbi Algorithm. This will be time complexity efficient if we use the built-in function from the Numpy library instead of using for loop for summation. The second one is for accuracy. Because there are not all the cases when the test word will be observed, in the case that the test word is not in the emission table, preprocessed it in the HMM stage by adding one extra line in the emission table to include that is not shown word. The probability that I put in for that row is the overall appearance of the tags in the training files. For example, if there are only three tags and 10 words in the training file, if there appear TAG-1 2 words, TAG-2 3 words, and TAG-3 5 words. The probability of the given new and not observed word will be 20% as TAG-1, 30% as TAG-2, and 50% as TAG-3. This probability will be used, instead of using all tags having the same probability. This is due to the reason that some of the tags are more often seen in a sentence than others not a common ones. When the training set is large, the occurrence will tend to be a more accurate prediction.