**Purpose**

The purpose of this assignment was to use a large sequence of sentences from the Wall Street Journal (WSJ), and to create a Hidden Markov Model to identify part-of-speech tags for each word of a given sentence. The sentences from the WSJ acted as training data, from which we could then identify the part-of-speech tags for new sentences with reasonable accuracy.

**Procedure**

* **Processing Data:** We defined a function that used the csv python library to parse the penntree.tag file. In using the csv library we were able to create individual rows (arrays) for which we could split on the tab (‘\t’) character easily. We made sure to add our start and end rows too. Then we defined another function to get all of the possible tags and their counts, and all the possible words in the WSJ data.
* **Transition Probabilities:** The data structure used for the transition probabilities was a python dictionary (hash table). We were able to forgo a nested dictionary by iterating through the processed WSJ data and finding the possible transitions from tag-to-tag first. Then we iterated through the WSJ again, counting the number of times a specific transition appeared. Now we just needed to calculate the transition probabilities themselves, P(Tagi |Tagj) = count(Tagi , Tagj ) / count(Tagj). From the getWordsandTags function we had already created a dictionary of all possible tags and their respective counts. Hence, we divided our dictionary of possible transitions by the number of Tagj in our tags dictionary. Structuring the transition probabilities in this manner created for a much faster program; the first attempt for finding transition probabilities using a nested dictionary was far less efficient.
* **Emission Probabilities:** For emission probabilities we used a nested dictionary. The reason for this is that for each word, we needed to find the emission probabilities given that word and a tag, P(word | Tag), meaning that we needed to find the emission probabilities of a word and all of the possible tags. One important detail to note though is that constructing this nested dictionary was much faster than using a nested dictionary for transition probabilities. The reason for this is that the majority of emission probabilities given a word and tag were 0. Thus, we only needed to calculate the emission probabilities of a word and two, or three, tags. We needed to be careful in constructing the nested dictionary, updating it at both levels of ‘’nestedness’’, i.e. if we wanted to update the dictionary, we first needed to update level 1, and then level 2. There is not an elegant way of doing this in python, but it did not present too much of challenge once you noticed this flaw of nested dictionaries. We then iterated through our nested dictionary and divided by the count of the Tag, much like the transition probabilities to get our emission probabilities. Before calling our emission probability finished, we manually entered the start (and end) probabilities to the dictionary, i.e. ‘<s>’ = 1 and ‘</s>’ = 1.
* **Viterbi Algorithm:** The Viterbi algorithm created the HMM given our transition and emission probabilities. For this assignment a general structure of the Viterbi algorithm was given. We added some minor changes to the algorithm: The first being modifying it to build our HMM given a dictionary of transition probabilities instead of a nested dictionary. This was not a difficult problem to fix, and the logic of the Viterbi algorithm stayed the same. The second being ridding the algorithm of the dptable function, it served no use besides formatting.

**Data**

We were given the penntree.tag data. It consisted of ~ 400,000 sentences. As mentioned in the processing point of the procedure sections, we created a function that used the python csv library to process our data into rows (arrays). The we used the getWordsandTags function to find all the tags and their counts, and the words of the data.

**Results**

The algorithm worked on all test sentences:

‘This is a sentence.’ : ‘DT, ‘VBZ,’DT’,’NN’,’.’

‘Can a can can a can?’ : 'MD', 'DT', 'NN', 'MD', 'DT', 'NN', '.'

‘This might produce a result if the system works well.’ : 'DT', 'MD', 'VB', 'DT', 'NN', 'IN', 'DT', 'NN', 'VBZ', 'RB', '.'

‘Can a can move a can?’ : 'MD', 'DT', 'MD', 'VB', 'DT', 'NN', '.'

‘Can you walk the walk and talk the talk?’ : 'MD', 'PRP', 'VBP', 'DT', 'NN', 'CC', 'VB', 'DT', 'NN', '.'

Not only did the algorithm succeed in the above tests, it also ran very quickly, which is also something of importance, though it is a secondary importance to creating an algorithm that gives the correct answer.