Jeremy Udis
December 4th, 2016

Viterbi Algorithm Writeup

Purpose

The main purpose of this assignment is to match tags with words presented in a sentence. In other words, given a word's context or position in a sentence, can one predict the likelihood a given tag belongs to that word? Viterbi's algorithm is our attempt to solving this problem.

Procedure

Transition Probabilities

To generate the transition probabilities, I first had to parse the penntree.tag file into a list of tuples in the form of (word,tag). In addition, I also kept track of a count of all the tags; I used a dictionary to store the count of each tag. From there, I traverse through the list of tuples, checking if the tag at index i-1 and the tag at index i is in the dictionary of dictionaries I use to build the transition probabilities. If it's not, I add the keys to the dictionary and set the value equal to one. If the keys are in the dictionary, I simply increment the value at that key, creating a counter of how many times the consecutive keys appear in the list. Then, I traverse through the dictionary with the keys mapped to their count, reassigning their values to the count divided by the count of the tag at the i-1 index. It would like this in pseudocode: dictionary[tag1][tag2] = dictionary[tag1][tag2] / tag_count[tag1].

Emission Probabilities

The process to parse the file was also needed to generate the emission probabilities, the only addition being I created a dictionary that stores the count of each (word,tag) pair. I used the count of each word tag pair to create another dictionary of dictionaries with the keys being d[word][tag], and values of each corresponding key is the count of the word tag pair divided by the tag count. Here's the general idea: dictionary[word][tag] = word_tag_count[(word,tag)] / tag_count[tag].

Viterbi Algorithm

The algorithm relies on both the transition and emission probabilities, which are both represented as dictionary of dictionaries. However, the final probabilities that viterbi generates are stored as a list of dictionaries. In terms of the back pointer, the list structure allows us to know the previous state by simply getting the index one position back from the current state. The algorithm begins by initializing probabilities at the 0th index(first observation) for each state. Because I set my start probabilities to 0 except for the start tag(SSSS), the start tag is the only non-zero value to begin. Then, the next job is to loop through the rest of the observations, looking at all combinations of states(i.e 47^2 combinations), and find the maximum transition probability. Then, our next job is to incorporate the emission probability to get our max probability; this is achieved by multiplying the max transition probability by the emission probability at the current state and current observation. The next task to find the max probability as well its corresponding state. We append this state to our list of steps of states states, then we traverse backwards through our list of dictionaries, inserting the state which has the highest probability to come before the current state(i.e a previous state), until we reach

the beginning of the list. At that point, our list of steps of states can be printed(note that this list is built from back to front).

Data

The data was derived from the penntree file given. The file consisted of many word tag pairs. Each word tag pair shared its own line, and the word and tag were separated by tabs. I parsed the penntree.tag file into a list of tuples in the form of (word,tag). In addition, I also kept track of a count of all the tags as well as a count of each (word,tag) pair; I used a dictionary to store this information. This allowed me to have a fast program as the count of everything needed to run viterbi was cached.

Result

For testing viterbi, I used the sentences given in the writeup. In each case, my program gave the correct output. One interesting thing of note was the variance of tags outputted to the terminal. Of the 47 possible tags, there were roughly 12 unique tags across the three sentences provided. I'm not sure if this can be attributed to structure of the sentences, but perhaps many sentences share common tags.