**MP 5: HMM POS Tagging**

Bum Jun Kim (bkim102) Samuel Hung (shung5)

**Introduction**

In this assignment, we are implementing part-of-speech tagging using Hidden Markov Modeling. We are given a set of training data in the form of words & tags and are asked to create a model that can accurately tag a set of tagless words. There are twelve possible tags for each word. We will write two approaches for this: Baseline and Viterbi.

**Baseline**

In our baseline algorithm, we are treating each word as independent. We disregard information pertaining to the history of the tags that came before. The only thing that matters in this implementation are the words given and the number of times it has been given a particular tag.

Essentially some variant of a bag-of-words model.

To do this, we used a dictionary that takes the given word as a key and stores a 12-tuple. Each dimension or index of the value in the tuple corresponds to the type of tag used. The values themselves correspond to the number of times the given tag has been given to the word.

When tagging words for our development model, we will inevitably run into the situation where we have never encountered a specific word before. For this situation, we just assume the word to be a noun, due to the high likelihood a word is a noun in general. This actually worked out quite well.

Our results are below:



*Brown*

*MASC*



*Both*

This sort of naive method works when we are given a vast set of words to train with because we are creating a case for every word we encounter. Assuming a word is a noun if we haven’t seen it before works sparingly. It’s like guessing a few questions on a multiple choice exam versus guessing on the entire exam. In the case where we don’t have so many words to work with, we want to use Viterbi, which is just a smarter approach to begin with.

**Viterbi**

This is where things get a bit more complex. Instead of treating each word as independent, we are looking at the sequence of the tags, and the probability of the word/tag pair to determine the possibility of the tag being correct. For the start of the sentence, we also store the probability of a tag appearing in front of the sentence.

Sentence by sentence, for the first word in the sentence, for each possible tag, we added the log of the probability of a tag appearing first in the sentence, as well as the log of the probability of the word having that tag. For the first word in the sentence we calculate the initial tag probability by:

log(P(tag)) = log(P(tag|start of sentence)) + log(P(tag|word))

For each possible tag, we choose only the top *n* tags with the highest possibility (*n* defined as a parameter *top\_x\_idx* of the Viterbi function) and store the tag sequence and probability for the sequence as a tuple in an array.

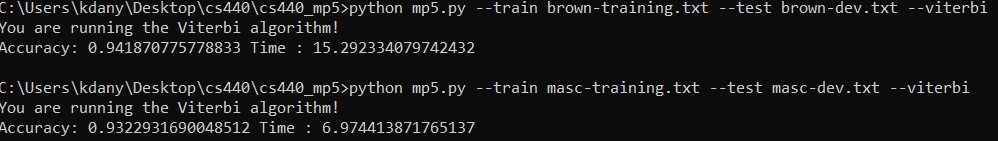
We then move on to the next word. For each of the *n* previous tags, we calculate the current tag probability for each combination, for each stored possible previous tags.

This probability P(*current tag\_sequence*) is calculated by:

log(P(*prev\_tag*)) + log( P(*current\_tag* | *prev\_tag*)) + log(P(*current\_tag* | *current\_word*))

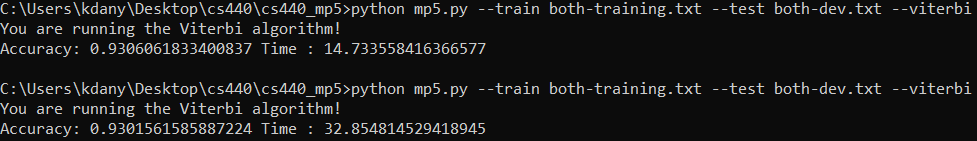
Then until the end of the sentence, we apply the same formula for each word. From all top-*n* possible tag sequence, we choose the sequence with the highest probability, then return the word-tag pair with the highest probability.

Our results for Viterbi are below:



Top: *Brown set*

Bottom: *MASC set*

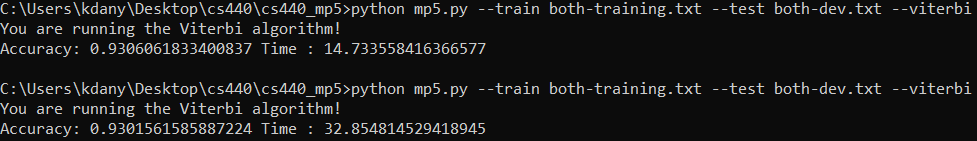


*Both set*

**Extra Credit**

To increase runtime, instead of storing all of the possible combinations that is possible for the word sequence, we chose only the top-*n* elements that hold the highest probability.As mentioned above, the user is able to specify the *top\_x* variable, which takes into account the top *x* tags for each word, probability-wise.

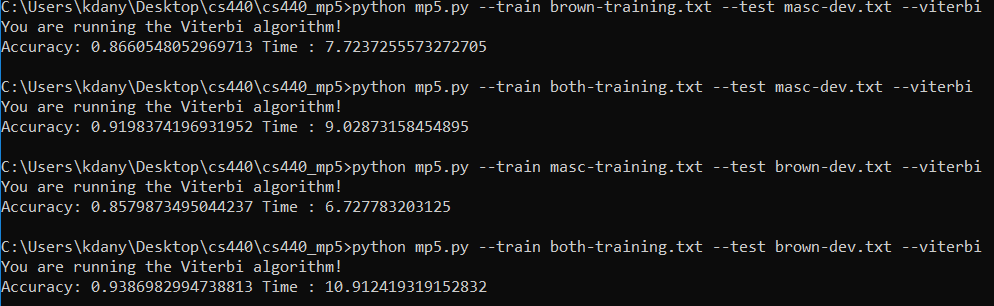
There is a trade-off for choosing a higher *x* value -- choosing a higher value for *x* will result in higher accuracy but the runtime of the computation will increase linearly with regard to *x*.



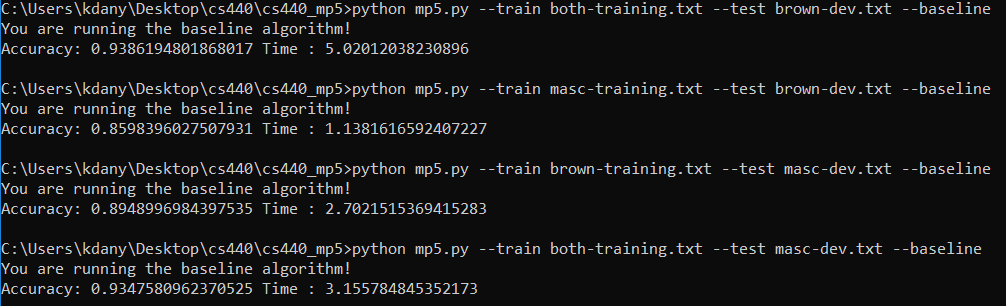
*Top*: Viterbi algorithm selecting only the top **5** tag sequences to store.

*Bottom*: Viterbi algorithm selecting the top **11** tag sequences to store.

As seen above, surprisingly when we stored only 5 tag sequences, it worked better than the storing 11 tag sequences.



*Viterbi mixed training & development*

*Baseline mixed training & development*

The Viterbi algorithm was expected to work better on mixed training sets, but the baseline worked better by at least 3%. The cause might be that the viterbi algorithm is narrowing the factors too precisely to the given dataset, that it does not do too well on other, unfamiliar datasets.