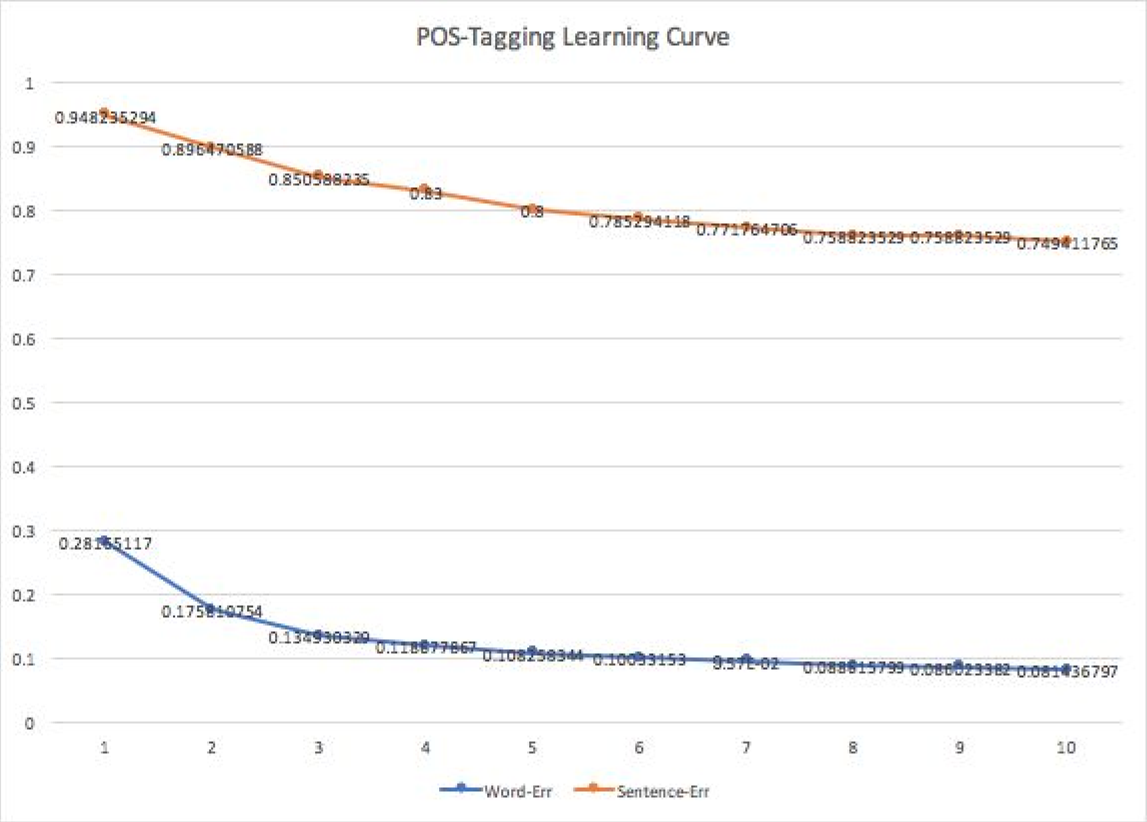
Alexander Soong
Professor Yu
ECS189G HW2
February 12, 2018

POS Tagging

- Task 1
  - For plotting the learning curve of the bigram viterbi algorithm, I decided to train the bigram hmm on the corpus with increments of 1000 lines at at time. Machine learning algorithms need lots of training for the “learning” aspect to work. So, I thought the visualization of learning curve with a growing data set would show the correlation between more training data and a lower testing error.



- Task 2
  - My approach to improving the bigram hmm algorithm was to implement a trigram hmm algorithm. This involved referencing the TA, Austin’s, bigram viterbi algorithm on Piazza and the available code for training the bigram hmm.
  - In training the trigram hmm, instead of just keeping track of the current and previous tag, I added another variable called *prev2tag* that kept track of the second preceding tag in the sequence. To do this, I initialized both *prevtag* and *prev2tag* to *INIT_STATE*. To populate model, I had a transitions dictionary: a

dictionary of dictionary of integers to keep track of how many times the two consecutive tags showed up in the training data. The emissions dictionary remained the same.

  - To better understand how a trigram hmm works, I watched a YouTube video series taught by Arnaldo Pedro Figueira Figueira, who has a online class for NLP on coursera. He went over the basics of hmm, how a trigram works, and the recursive definition of the viterbi and backpointer algorithms. The first step was paring the hmm output file "myTrigram.hmm". For the transitions I stored the previous two tags as a tuple, and the current term as a pair to that tuple. Together, this was the key that I used to store into the dictionary *trans*. The *emit* dictionary stayed relatively the same. To implement viterbi algorithm, instead of having just a product between the set of tags with itself, I used a product of the set of tags with it self, and with itself again; effectively having a triple for loop.The resulting list of combinations of three tags *u, v, w* allowed me to keep track of the current, preceding and second preceding term. My backpointers stored the previous tag, and the second previous tag as a tuple. This made it easier for me, when I was iterating backwards searching for the preceding tag.
  - My trigram hmm with viterbi scored an accuracy of 16% word error and 64% sentence error. Compared to the bigram hmm with viterbi, the word error is higher, but sentence error is lower. I believe this highlights the fact that using a trigram hmm, produces results for the current tag by looking more heavily at preceding tags. So while estimating the word given the current tag may have a lower accuracy with the trigram hmm, given information about preceding tags allowed it to perform better.
  - Results:

    ```
    Alexander:hw2 asoong$ ./tag_acc.pl ptb.22.tgs trigram.out
    error rate by word:      0.164992397238079 (6619 errors out of 40117)
    error rate by sentence:  0.649411764705882 (1104 errors out of 1700)
    ```

- Task 3
  - Japanese Results:

    ```
    Alexander:hw2 asoong$ ./tag_acc.pl jv.test.tgs myjap.out
    error rate by word:      0.530905270530555 (3032 errors out of 5711)
    error rate by sentence:  0.997179125528914 (707 errors out of 709)
    ```

  - Bulgarian Results:

    ```
    Alexander:hw2 asoong$ ./tag_acc.pl btb.test.tgs mybulg.out
    error rate by word:      0.273003033367037 (1620 errors out of 5934)
    error rate by sentence:  1 (398 errors out of 398)
    ```

  - While both the Bulgarian and Japanese sentence error rates were almost 100% the word error rates were 27% and 54% respectively. After inquiring why the big difference in word error rates between the three languages I arrived at my conclusion. There are many sentences in the Japanese corpus that are only one word or two words long. In Bulgarian corpus there are as well; however, most of the sentences are closer to the length of the English sentences. Why does this matter? Because I am using a trigram hmm, if a sentence is only one or two words long, they would both share the *INIT_STATE* tag. This would result in them not being found in the *trans* dictionary and therefore generate a wrong solution. A possible way to improve on the performance of these models is to use a backoff approach similar to what we had done for spell checker in homework 1. If the trigram hmm had not found an optimal tag sequence, we could use a bigram hmm to find it.

- Collaboration:
    - Youtube series - https://www.youtube.com/channel/UCzMIECXd856E028HnSYExOQ
    - Consulted student Richard Gao regarding finding the best backpointer
    - Consulted student Henry Le regarding how to store the prevtag and prev2tag in the trigram hmm

Trigram hmm trainer:
My_trigram_hmm.py

English trigram viterbi:
Trigram-viterbi.py

Hmm sequence output for english for ptb-22.txt:
My.out

hmm sequence output for japanese for btb.test.txt:
myjap.out

trigram viterbi for bulgarian for jv.test.txt:
bulg_viterbi.py

hmm sequence output for bulgarian:
mybulg.out

Tag sequence for ptb.23.txt to test against gold:
final23.out