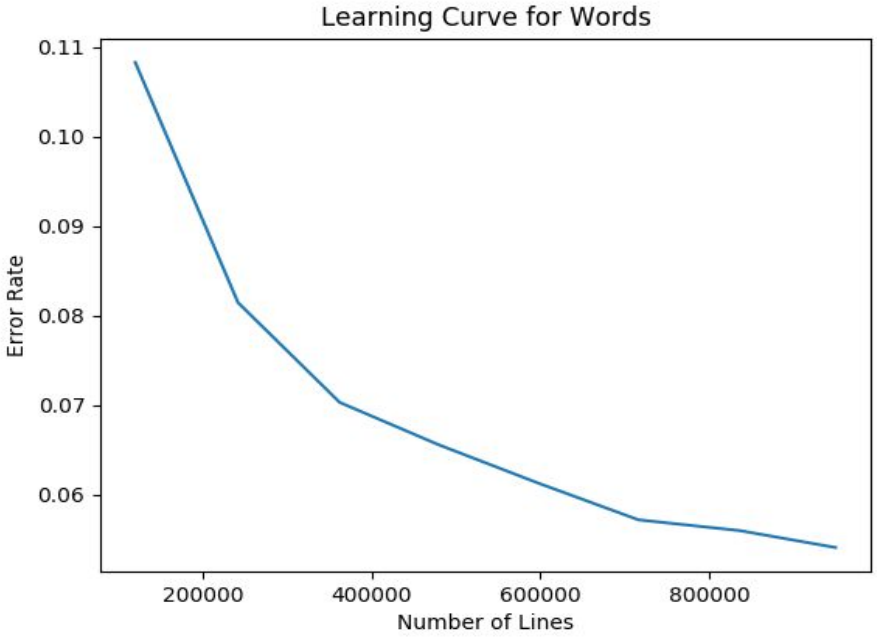
Srivarshini Ananta

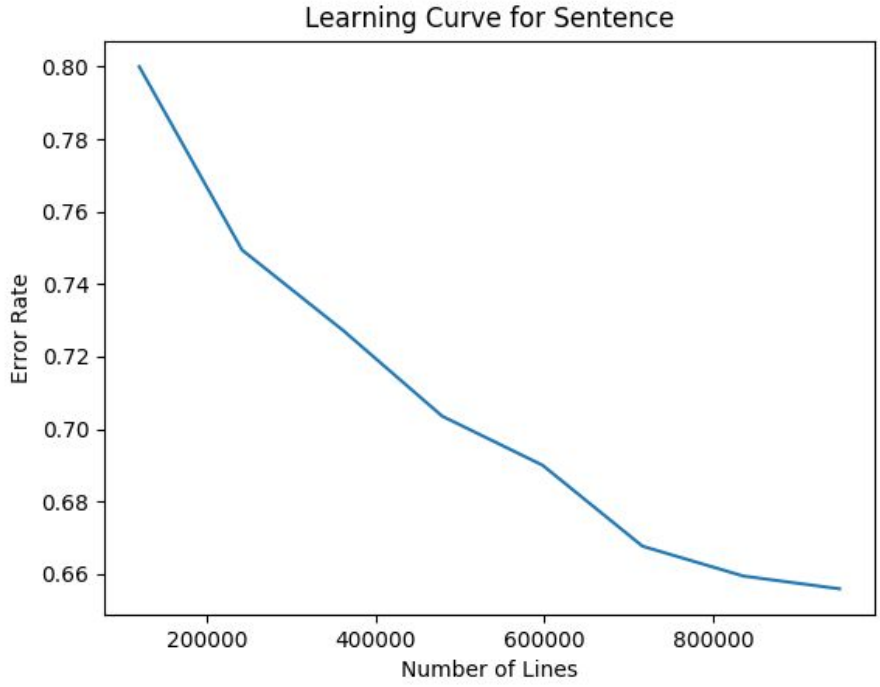
ECS 189G

February 12, 2018

Homework 2: POS Tagging

1.





The two figures above show the relationship between the error rate in correlation with the number of sentences and the number of words present in the text files we are evaluating given the bigram HMM provided to us. Based on these models, we can see that when we are provided with more training data the error rate decreases. Therefore, getting more POS-tagged data would make our system more accurate by reducing error rates because we are provided with a larger corpora associating certain words with POS-tags. One possible benefit would be getting more POS-tag possibilities for words which were previously labeled as an out-of-vocabulary word when the training data was smaller.

2. Baseline Model for Development Data

```
error rate by word:      0.0540917815389984 (2170 errors out of 40117)
error rate by sentence:  0.655882352941176 (1115 errors out of 1700)
```

My Model for Development Data

```
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ ./train_hmm.py ptb.2-21.tgs ptb.2-21.txt > my.hmm
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ python viterbi.py my.hmm < ptb.22.txt > my.out
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ ./tag_acc.pl ptb.22.tgs my.out
error rate by word:      0.0524715208016552 (2105 errors out of 40117)
error rate by sentence:  0.652352941176471 (1109 errors out of 1700)
```

*** Note:** output for testing with ptb.23.txt is called **new.out** in my directory

My model for the development data performed at a slightly better rate than that of the baseline model. My initial approach simply just used a viterbi trigram and everytime my model was not able to assign POS tags to a particular sentence it would just print a blank line. With my initial approach I was simply getting an 8% error rate and therefore to get a lower error rate I decided I needed to find a way to assign tags to sentences I was printing

blank lines for. Therefore, I came across the idea that in some cases certain trigram transitions do not exist but bigram transitions might, similar to the backoff model in the first homework. Therefore, I replaced printing a blank line with a custom bigram model for which I additionally sent in data for from train_hmm.py along with the trigram transition and emission values. I believe I also improved my error rates with the additional smoothing in train_hmm.py where I had an extra addition of 1 for emissions[tag][token] and also gave more of a probability for out-of-vocabulary words by appending emissions[tag][OOV_WORD] through each (tag, token) in pairs iteration. My improvement in sentence error rate is not as significant as my improvement in word error rate and I believe this is because the errors in the output file were more spread out therefore affecting more sentences.

3. Baseline Model for Bulgarian

```
Srivarshinis-MacBook-Air:baseline_test charanananta$ ./train_hmm.py btb.train.tgs btb.train.txt > btb.hmm
Srivarshinis-MacBook-Air:baseline_test charanananta$ ./viterbi.pl btb.hmm < btb.test.txt > btb.out
Srivarshinis-MacBook-Air:baseline_test charanananta$ ./tag_acc.pl btb.test.tgs btb.out
error rate by word:      0.115942028985507 (688 errors out of 5934)
error rate by sentence:  0.751256281407035 (299 errors out of 398)
```

Baseline Model for Japanese

```
Srivarshinis-MacBook-Air:baseline_test charanananta$ ./train_hmm.py jv.train.tgs jv.train.txt > jv.hmm
Srivarshinis-MacBook-Air:baseline_test charanananta$ ./viterbi.pl jv.hmm < jv.test.txt > jv.out
Srivarshinis-MacBook-Air:baseline_test charanananta$ ./tag_acc.pl jv.test.tgs jv.out
error rate by word:      0.0628611451584661 (359 errors out of 5711)
error rate by sentence:  0.136812411847673 (97 errors out of 709)
```

My Model for Bulgarian

```
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ ./train_hmm.py btb.train.tgs btb.train.txt > btb.hmm
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ python viterbi.py btb.hmm < btb.test.txt > btb.out
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ ./tag_acc.pl btb.test.tgs btb.out
error rate by word:      0.104819683181665 (622 errors out of 5934)
error rate by sentence:  0.688442211055276 (274 errors out of 398)
```

My Model for Japanese

```
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ ./train_hmm.py jv.train.tgs jv.train.txt > jv.hmm
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ python viterbi.py jv.hmm < jv.test.txt > jv.out
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ ./tag.acc.pl jv.test.tgs jv.out
-bash: ./tag.acc.pl: No such file or directory
Srivarshinis-MacBook-Air:189gpt2_final_smooth charanananta$ ./tag_acc.pl jv.test.tgs jv.out
error rate by word:      0.0614603396953248 (351 errors out of 5711)
error rate by sentence:  0.169252468265162 (120 errors out of 709)
```

My model for Bulgarian performed at almost the same rate as the baseline model with a very slight improvement in both the error rate by word and error rate by sentence. The improvements were similar to how my model performed with English, which makes sense considering how they both are subject-verb-object languages which my viterbi.py takes into account in the bigram backoff by assigning any empty tag appended into the taglist as a plural noun 'NN' and also for any empty backpointers. My model for Japanese also performed quite similarly to the baseline model with a very slight improvement in the error rate by word but slightly worse result in the error rate by sentence. The slightly worse error rate by sentence was likely caused by the fact that errors were more spread out in the output file and therefore affected more sentences, similar to what happened when testing with the development data. I believe the reason that there was more significant progress for Bulgarian through my model than Japanese is because my viterbi.py was built to look at sentences which have higher probabilities of having words with plural nouns and sentences where words are separated by spaces and have a clear order. However, this is not the case with Japanese because the sentence structure is quite different and more liberal. My trigram with backoff algorithm in viterbi.py is unable to take note of the unique Japanese sentence structure despite looking at the training data and providing exceptions similar to the baseline model. This might also be due to the fact that there are different dialects in the Japanese language so the training data was not sufficient. I believe the slight improvement in word rate was due to the smoothing I performed in train_hmm.py where I had an extra addition of 1 for

emissions[tag][token] and also gave more of a probability for out-of-vocabulary words by appending emissions[tag][OOV_WORD] through each (tag, token) in pairs iteration.

Resources:

Viterbi

https://stathwang.github.io/part-of-speech-tagging-with-trigram-hidden-markov-models-and-the-viterbi-algorithm.html

English vs. Japanese

http://www.japaneseprofessor.com/lessons/beginning/structure-japanese-sentence/

Discussed ideas with Saksham Bhalla, Kavitha Dhanukodi, Shraddha Agarwal, Yash Bhartia, Aakash Kapoor and Iyesha Puri.