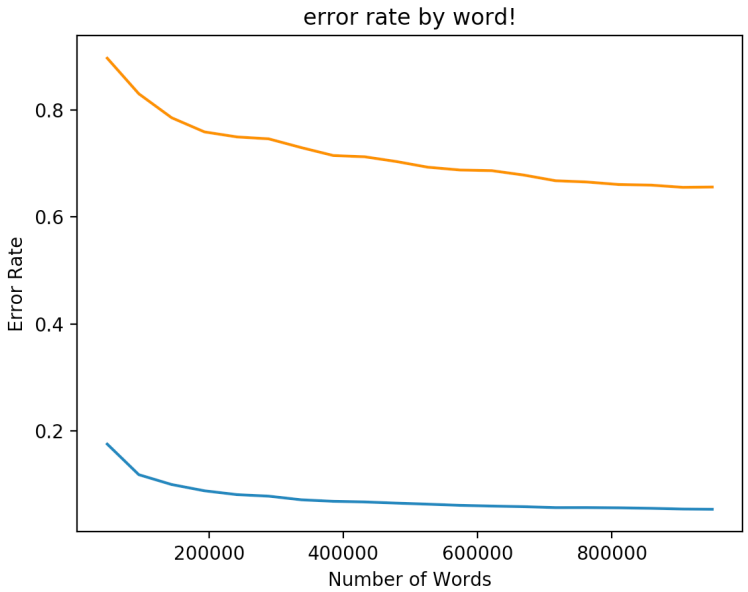
Shraddha Agrawal
S.I.- 913001594
ECS 189 HW 2

HW-2 REPORT

1. Learning curve for bigram HMM using different size corpus.



In the above graph the blue line represents the error rate by words while the yellow line represents the error rate by line in the corpus.

My training set increases by 2000 lines per corpus. For a data set of 100000 words the error rate of words is around 0.2% while for a data set of 800000 words the error rate of words is approximately 0.1%, which is less than the initial value. As the number of words in the corpus increase the error rate decreases.

But on a closer observation, I see that after training on certain amount of data the error rate is more or less constant. So even though getting more POS tagged data will increase accurateness of our model it will not affect it by great percentage. The accurateness of our model will come with better natural language processing algorithm rather than increasing the POS tagged data.

2.

I approached problem 2 by developing Viterbi for trigram with back-off as bigram Viterbi model with add on smoothing in train_hmm.py.

Firstly, I modified the train_HMM.py file by initializing two previous states: prevprevtag and prevtag and calculating the trigram transition probability of a tag that is dependent on two of its previous tags. Whereas, Emission probability of a word depends only on its own tag and is independent of neighboring words and tags. Furthermore, I added smoothing by initializing emissions[tag][token] to 1 whenever it sees a OOV_WORD. By adding 1 we assume that each bigram/trigram we encounter has count 1, thus making sure that no tag-token pair has zero probability.

Then I implemented the Viterbi trigram algorithm using the pseudo algorithm that was TA went over in discussion. In the algorithm, pi[(u, v, w)] represents a cell of dynamic programming table and keeps track of maximum probability of a tag sequence ending in tags u, v at position k. While bp= {} keeps track of backpointers. After evaluating the backpointers it prints the max probability tags in reverse order to get the correct order of maximum probable tags of a sentence.
But on only applying Viterbi trigram, the above algorithm failed at several lines and I got higher error rate around 8%. Therefore, in order to mend that I developed a backoff to bigram Viterbi. Whenever Viterbi trigram fails the sentence is run on a Viterbi bigram.

This new model, decreased my error rate significantly to 5.43%.

```
KeyboardInterrupt
[campus-050-038:ECS 189G HW 2 shraddhaagrawal$ ./train_hmm3.py ptb.2-21.tgs ptb.2-21.txt > my.hmm
[campus-050-038:ECS 189G HW 2 shraddhaagrawal$ python ./viterbi3.py my.hmm < ptb.22.txt > my.out
[campus-050-038:ECS 189G HW 2 shraddhaagrawal$ ./tag_acc.pl ptb.22.tgs my.out
 error rate by word:      0.0543410524216666 (2180 errors out of 40117)
 error rate by sentence:  0.661176470588235 (1124 errors out of 1700)
```

3.

Bulgarian
On running baseline model and my model on btb.* files, my error word rate and sentence error rates are less. One main reason is that Bulgarian and English are both Indo- European languages. Therefore, a model that runs well on English language will give significantly good results for Bulgarian language. Further, on observing Btb.txt we see that words are separated thus having less ambiguity.

Baseline model performance of Bulgarian

```
[Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./train_hmm.py btb.train.tgs btb.train.txt > btb.hmm
[Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./viterbi.pl btb.hmm < btb.test.txt > btb.out
[Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./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)
```

My Viterbi Trigram with backoff performance of Bulgarian

```
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./train_hmm3.py btb.train.tgs btb.train.txt > btb.hmm
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ python ./viterbi3.py btb.hmm < btb.test.txt > btb.out
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./tag_acc.pl btb.test.tgs btb.out
error rate by word:      0.0990899898887765 (588 errors out of 5934)
error rate by sentence:  0.668341708542714 (266 errors out of 398)
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$
```

Japanese

On running baseline model and my model on jvj.* files, my error word rate is less but sentence error rates are worse. Sentence error is worse because in Japanese word segmentation is difficult in a sentence so Viterbi gives higher error results. Some of the other key differences between Japanese and English is the the fact that Japanese is composed of three main scripts: Kanji, Hiragana and katakana, which increases the vocab in Japanese language significantly and creates more ambiguity. Whereas considering trigram transition probability instead of bigram transition probability gives a less word error in Japanese.

Baseline model performance of Japanese

```
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./train_hmm.py jv.train.tgs jv.train.txt > jv.hmm
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./viterbi.pl jv.hmm < jv.test.txt > jv.out
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./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)
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$
```

My Viterbi model for Japanese

```
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./train_hmm3.py jv.train.tgs jv.train.txt > jv.hmm
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ python ./viterbi3.py jv.hmm < jv.test.txt > jv.out
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$ ./tag_acc.pl jv.test.tgs jv.out
error rate by word:      0.0621607424268955 (355 errors out of 5711)
error rate by sentence:  0.169252468265162 (120 errors out of 709)
Shraddhas-MacBook-Pro:ECS 189G HW 2 shraddhaagrawal$
```

Addition Notes:

Files submitting:

- final.out => my tags of ptb.23.txt
- Trigram_Viterbi.py => my model of trigram Viterbi with bigram back-off
- Train_hmm3.py => modified train_hmm.py that considers trigram transition probabilities and has add on smoothing to account for OOV_WORD

Files for plotting a learning curve

- learningCurve.py
- graph.sh

I discussed code with:

Kavitha dhanukodi ,
Iyesha Puri,
Srivarshini Ananta,
Yash Bhartia,
Aakaash Kapoor,
Sakhsham Bhalla.

Online reference:

1) https://stathwang.github.io/part-of-speech-tagging-with-trigram-hidden-markov-models-and-the-viterbi-algorithm.html
2) TA discussion slides from Viterbi trigram algorithm and bigramViterbi.py.