**Evaluate the accuracy of unigram part-of-speech tagging on the given test files.**

*Train on small-train.xml and test on text_1.xml. What is the accuracy?*
**Answer:**Similarity = 13088 / 13945 = 93.85442811043384 %

*Train on small-train.xml and test on text_2.xml. What is the accuracy?*
**Answer:** Similarity = 7743 / 9181 = 84.33721816795556 %

*Can you explain the discrepancy in accuracy between them?*
**Answer:** There is discrepancy in the accuracy between them because of the following reason:

- test2.xml has more unseen words in it.
- test2.xml has context which is not similar.

*Examine the output of the tagger and see if any patterns emerge regarding*
*where the tagger tends to make more errors.*

**Answer:** After seeing the output of tagger we see a pattern that the new tagged data is directly proportional to the context of the trained data.
If we send out a text which is not similar to the trained data or which does not belong the training context the accuracy will be reduced.

*What conclusions can you draw? What do you think might improve the tagger's performance?*

**Answer:** To improve the performance what we can do is, we can train our data using a very large corpus.
Also instead of the unigram tagging we can use the bigram or trigram tagging.
The tagging can further be improved using smoothing algorithms such as Good turing or k smoothing.

*Modify POSTag.java to tag sentences using a bigram tagger, i.e., so that the probability of a tag depends on the previous tag as well. You will need to add a Hashmap to store Tag-Tag probabilities, using plus-one (or plus-k) smoothing.*[**DONE**]
*Evaluate your bigram tagger on both test files. How did performance change?*

**Answer**:
**In terms of Accuracy:**
After changing the tagger with the bi-gram implementation, I observed that
the accuracy of the taggers changed:
The new accuracy for text_1: Similarity = 13469 / 13945 = 96.5865901756902 %
The new accuracy for text_2: Similarity = 7965 / 9181 = 86.75525541879969 %

**In terms of Functional Complexity:**
The new bi-gram tagger is taking more time than what unigram was taking.

Thus we can conclude that the bigram tagger has higher complexity than that of unigram.

*Optional enhancements (at least one required for pairs or CS-585 students):*
*Implemented Good Turing Smoothing* [**DONE**]

*Evaluate your chosen enhancement(s) in the same way you evaluated the base system.*
*Is the accuracy improved?*

*Does your chosen enhancement reduce any of the types of errors that you previously noted for the base tagger? If so, which and why?*
I have implemented a good turing smoothing algorithm.
It increases the accuracy:
The new accuracy after good turing smoothing for test1.xml:
**Similarity = 13462 / 13945 = 96.53639297239154 %**
The new accuracy after applying good turing smoothing for test2.xml:
**Similarity = 8054 / 9181 = 87.72464873107505 %**

For this case the accuracy of test_2 gets increased, it happens because we test_2 has more unseen words which occur just once. Since its probability is now taken into account hence the overall Similarity for test_2 has increased.

*How do you think you might improve the tagger even more?*

**Answer:** Under situations we can also have 4-Gram or 5-gram tagging but it will deliberately increase the code complexity and will take very long time to run.
Thus we do not go more than 3-Gram.
Also we can apply more than 1 training corpus so that our system learns through different context.
One of the examples is facebook translate app where it asks for the rating of a given translation.
When a person rates a translation, then for a given context tagging is done and a similar context will be tagged further based on the overall rating.