

1. Evaluate the accuracy of bigram part-of-speech tagging on the given files.

Training File	Testing File	Reported Accuracy
Literature.sgm	Detective.sgm	64.445%
Media.sgm	Detective.sgm	66.527%
Social.sgm	Detective.sgm	61.899%
X-train	Y-Test	76.389%

I modified the Evaluation/Evaluator.java file to output the misidentified word-tag combinations. The output was given in the following format: **word <tag> - testWord (testTag)**. From this output, I analyzed the X-train/Y-Test combination since it was the smallest and easiest to test. There was no real pattern in the tags being misidentified. The following TAG(# Mismatched) is from the Y-Test evaluation: VVB(1); POS(1); VBZ(1); AV0(2); NP0(2); VVI(1); VVD(1); TO0(1); AJ0(1); VVG(1); PNQ(1); CJS(1); VVD-VVN(1). No tag was misidentified more than twice with 13 different tags being misidentified. This gave no real clues on where to proceed; however, there was a pattern to be found in the type of incorrectly labeled tags. The tags in Y-Test were misidentified as: PUN(4); AT0(3); NN1(3); SENT(2); PRP(1); NP0(1); AJ0(1). The 13 tags that were misidentified were only tagged in 7 different groups. The most commonly misidentified tags were identified as punctuation, articles, and singular nouns.

I have found, as explained below, that an enhancement to the smoothing and evaluation would generate a noticeable increase in tagging accuracy.

2. Choose (at least) one of the following enhancements to the bigram POS tagger and implement it.

After analyzing the HmmModel/ModelProbabilities.java and Evaluation/Evaluator.java, I had found a few key areas that could be modified to increase the accuracy of the tagger. These two files will be the focus of my enhancements to the tagger.

HmmModel/ModelProbabilities.java. After reviewing the code in this section, I have noticed that Plus-One Smoothing was being used by this tagger. The first step I have decided to take is to implement Add-Delta smoothing using Lidstone's Law.

Lidstone's Law is defined as:

$$P_{AddDelta}(w_1 w_2 \dots w_N) = \frac{C(w_1 w_2 \dots w_N) + \Delta}{N + \Delta V}$$

Where N is the number of occurrences in the corpus, V is the number of distinct occurrences, and δ is less than one. I chose to use a δ value of .01, well below the Jeffery-Perks value of .5 in order to increase the effectiveness of the tagger.

After completing the assignment, I had begun an implementation of the Good-Turing Smoothing algorithm, but due to time limitation caused by other class projects, only the computation of the frequency hash map was completed.

Evaluation/Evaluator.java. While analyzing the output of the misidentified word-tag combinations that I added, I noticed a key fault. The evaluator did not correctly identify tags if the pre-tagged word had a hyphenated acceptable tag list. To correct this problem, I split the tag at the dash (“-”) and compared the computed tag to each of the tags in the acceptable tag list. I noticed this after implementing Lidstone’s Law and achieving 97.222% accuracy when the only misidentified tags were correctly identified. After this fix, the X-train/Y-Test combination yielded a perfect 100.000% accuracy rating.

3. Evaluate your chosen enhancement in the same way you evaluated the base system.

After the enhancements to Evaluation/Evaluator.java and HmmModel/ModelProbabilities.java, the following accuracies were reached:

Training File	Testing File	Reported Accuracy
Literature.sgm	Detective.sgm	85.074%
Media.sgm	Detective.sgm	84.927%
Social.sgm	Detective.sgm	83.579%
X-train	Y-Test	100.000%

With the additions/corrections made to the base POS Tagger, I have achieved an increase of accuracy above 20%. The X-train/Y-Test combination yielded a 23.611% increase in accuracy. Literature.sgm/Detective.sgm yielded an increase of 20.629%. Media.sgm/Detective.sgm yielded an increase of 18.400%. And Social.sgm/Detective.sgm yielded an increase of 21.680%. Disregarding the X-train/Y-Test combination, the modifications to the tagger have resulted in a 20.236% increase in accuracy.

The average reported accuracy I received from the updated HMM POS Tagger (disregarding the X-train/Y-Test combination) was 84.527%. While this falls short of using the most common POS for each word (91% accuracy) and the OAK system at NYU (97% accuracy), it does show drastic improvement over the tagger that was originally given by increasing accuracy by an average of over 20%.

From looking at a random sample of the misidentified output, I have noticed a few reoccurring themes. The tagger still has difficulties correctly identifying nouns. It appears that most of the misidentifications were within the same category (i.e. a plural noun is identified as a singular noun). I think the best fix for this would to be to strip words down to analyze word endings. The tagger was not able to identify “ as punctuation. This is probably due to the limited

corpus with unseen words. Smoothing and a bigger corpus would improve accuracy here. Parentheses “(” and “)” were only identified as punctuation and not as the finer classification of left and right parentheses. A common standard across pre-tagged data and a larger corpus would improve accuracy in this example. Finally, I noticed a problem in the identification of hyphenated words. This is most likely due to the limited corpus and unseen words. To correct this problem would probably require breaking the hyphenated words into their individual components and analyzing each individual word and then combining the results to help identify the original hyphenated word.

It also appears that the `Tokenize/MiddleTier.java` file could use some modification as well. Using the tagged corpus, it appears that not just the tagged words are being stored in the hash tables. The tagger is giving occasional bad tags such as “for ,” “ample,” and “n=48,” all of which seem to indicate a problem with the tokenizer. Modifying `Tokenize/MiddleTier.java` to filter based on a predefined set of POS tags would improve the accuracy of the system more than any modification to the smoothing algorithm or change to the Viterbi algorithm would.