**Assignment #3 Write-Up**

Mimi Benkoussa

CS325

*\* Complete code/Python file is included at the end of this write-up, as well as the complete list of all outputs including maxmatch outputs for each of the three lexicons, WER for each lexicon, and the average WER for each lexicon.*

1. Maxmatch on hashtags with various lexicons

For this portion of the assignment, I used the maxmatch algorithm from the Speech & Lang. Processing textbook pg. 71. The three lexicons were originally in different formats (list objects and text files), so I had to first make them consistent by normalizing the text and stripping/splitting the text in a text file into a list. For the testWithAnswers.txt, I used re.sub on commas to eliminate the original hashtag and preserve only the correct segmented solution, so that comparison was made easier for the latter portions of the lab.

In terms of the maxmatch outputs, I was surprised at how accurate all of the lexicons were. When maxmatch failed, one common failure was when the algorithm saw an ‘s’ after a word and made it plural when it wasn’t supposed to be.

Ex: NLTK lexicon made ‘computerscience’ ‘computers c i enc e’ rather than computer science because it used plural for ‘computers.’ Another example of this was with ‘governmentshutdown’, which the Linux and Google lexicons made as ‘governments hut down’ by using plural for ‘governments.’

Conversly, another failure/error was not accounting for plurals when they needed to be accounted for.

Ex: NLTK lexicon made ‘ilovemyfans’ ‘i love my fan s’ rather than pluralizing the word ‘fan.’ It didn’t account for the plural of the word in its dictionary.

Finally, another error involves the regular vs. past tense of verbs – the algorithm would make verbs in the past tense if the word after them allowed for it.

Ex: Linux and Google lexicons made ‘visualizedata’ ‘visualized at a’ and ‘visualized ata’, respectively. Both of these lexicons use the past tense of the verb visualize.

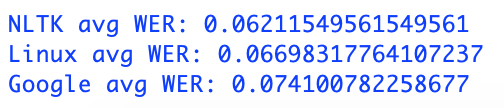
2. Objective evaluation of maxmatch using WER

The minimum edit distance algorithm was also based off of the algorithm from the Speech & Lang. Processing textbook (pg. 76). Then, the WER function simply divided the minimum edit distance by the length of the correct segmentation string. My print\_all function is what tied the functions all together and printed the respective maxmatch outputs, as well as the WERs for each hashtag for each lexicon and the average WER for each lexicon. The solutions will be below, and here are the respective table and graph of the outputs:

Table:

|  |  |  |  |
| --- | --- | --- | --- |
| Hashtag | NLTK WER | Linux WER | Google WER |
| 1 | 0 | 0.3124 | 0.0625 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0.28571429 |
| 4 | 0 | 0 | 0 |
| 5 | 0 | 0.15789474 | 0.15789474 |
| 6 | 0.07142857 | 0 | 0.21428571 |
| 7 | 0.36363636 | 0.36363636 | 0 |
| 8 | 0.1875 | 0.25 | 0 |
| 9 | 0 | 0 | 0.0625 |
| 10 | 0.1 | 0 | 0 |
| 11 | 0.11111111 | 0 | 0 |
| 12 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 |
| 14 | 0.09090909 | 0.18181818 | 0 |
| 15 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0.09090909 |
| 17 | 0 | 0.21428571 | 0.14285714 |
| 18 | 0.04166667 | 0 | 0.08333333 |
| 19 | 0.11111111 | 0.11111111 | 0.11111111 |
| 20 | 0 | 0 | 0.16666667 |
| 21 | 0.38461538 | 0 | 0 |
| 22 | 0 | 0.08333333 | 0 |
| 23 | 0 | 0 | 0.36363636 |
| 24 | 0 | 0 | 0.11111111 |
| 25 | 0.09090909 | 0 | 0 |

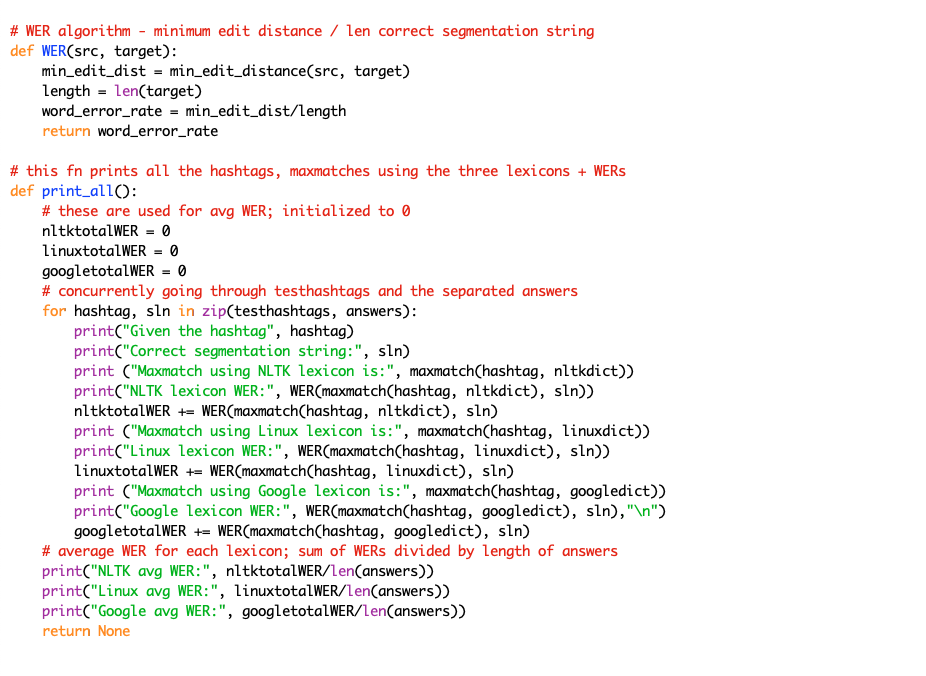
Average WER across all inputs for each lexicon (these were calculated by the function):



Graph:

Complete Python Code:





Complete outpuf of print\_all():



