Project 1 Report

Brandon Cabrera Marquez

1. Introduction

For Project 1, I used Global Edit Distance to produce spelling recommendations to the Twitter user. In my report I will talk about the system's filtering methods, multi-word comparison methods, its behaviour, and why it was very difficult to get a system that could quickly and effectively find typos in a location name.

2. Discussing the Code/Methodology

Filter Methods:

Given that Global Edit Distance uses matrices to calculate the relative distance between 2 words, it became really important to filter out unnecessary data. So I created a "polishData" function in order to filter out 3 types of data that are very common in tweets:

-Hyperlinks
-Usernames
-Hashtags

Problems with Filtering

In order to speed up the process of calculating the GED score for every word in a tweet, I first intended to filter out the top 100 most used English words. However, that turned out to be useless because most of those words were part of city names. Getting rid of them within the tweets and location names would skew the GED score calculated by the computer (especially for multi-word locations). It would also ruin my ability to contextually compare them in order to see if the results were meaningful.

City Names with More Than One Word

City names vary in length. Unfortunately for Global Edit Distance, this caused processing to take painstakingly long. Even a word as short as "I" took 15-20 minutes to get through the gazetteer. Therefore, after noticing that the best single-word relevant results had a distance of -2 and above, I only allowed my code to analyse a single word with the first word of the location. If the distance between the first words was greater than or equal to -2, I would allow it to continue to compare the subsequent words. More often than not, multi-word cities failed to pass this parameter which, consequently, saved my algorithm from unnecessary processing.

To prove that it works, take the example of a tweet that mentioned "…tiger woods still…"

Tweet Word	Result
`tiger`	`{"['Singer', 'Woods']": 4, "['River', 'Woods']": 4}`
`woods`	`{"['Cooks', 'Still']": 4}`

Table 1: My system prints the leading word and returns tokenized location names followed by its GED score.

3. Evaluation: Precision, in the context of this project, is measured by (# of relevant results)/(# of attempts). The slow nature of GED only allowed me to get through 86 tweets. Within those, it parsed through 1021 words that my system deemed relevant. Out of these 1021 words, six of them were correctly spelled US locations, three were from outside the US, and four were misspelled locations.

True Positive Results: Those four were the only truly relevant words. It caught four cities which were un-capitalized and returned it capitalized. Though it's arguable whether this is an error or not, I wished to mimic the behavior of modern predictive keyboards. Take "dallas" for example. Google's keyboard, Gboard, will ask if you meant "Dallas" or if you'd like to add "dallas" to the dictionary, thus categorizing it as an error. Therefore my precision was 4/1017.

Behavior: The score is low with good reason. Without context, it is incredibly difficult to know whether certain words make up a location name. Filtering lower-cased words may increase the speed of your algorithm (and quite possibly your precision), but it misses out on finding potential errors in location names that were erroneously un-capitalized (in other

words it will have a low recall). Increasing the number of words checked can increase the recall, but it is not completely possible to accurately calculate recall in tweets. This is because we can never be completely sure of what a twitter user meant to type. So if analyzing a user's intentions is an ambiguous task to accomplish as humans, it is even more difficult for a computer to do. Twitter's 140 character restriction makes things worse because user's will often abbreviate words to stay within the limit. The tweet "…See you in Cali soon!" holds the perfect False Positive example. My system believed Cali was misspelled and suggested "Calio" or "Cali Woods", but it failed to realize that this was a correct abbreviation of California. A similar error occurs in Table 1, but this one is due to the computer's lack of pop cultural knowledge.

4. **Conclusion:** Ultimately, a machine with no knowledge of our slang and culture has no chance of refining its word filtering methods. Checking only capitalized words in order to save processing time causes the system to miss any lowercased typos. Contrastingly, checking through every word (with the exception of the three types of data I filtered), increased the chance of finding typos, but significantly reduced its ability to parse through tweets quickly. Given that my algorithm had no way of filtering common words without skewing the GED results, most of the processing time was spent on correctly spelled words that were assumed to be misspelled location names. But I preferred a more careful system over one that only checked the few capitalized words.