COMP90049 Project 1: Finding Misspelled Location Names

San Kho Lin (829463) sanl1@student.unimelb.edu.au

1 Introduction

In this project, I am going to experiment approximate string matching techniques learnt in COMP90049. The objective of the project is to identify misspelled location names in tweets collection from Twitter. At first, the project objective seems trivial but indeed it is analogous to finding needle in a hay stack. The rest of the report will elaborate on why searching misspelled location in tweets collection is a tough problem to solve.

2 Observing dataset

To begin with, there are two dataset; *a gazetteer* and *a tweets collection*. *A gazetteer* comprise of location names in US and one location per line. Each location name spells in a single character, a word or multi-words. It also contains non-alphabetic and numeric characters. The location names usually start with capital letter follow by lower case. This gazetteer is the main dictionary and it has total of 1.3M locations — 1,294,704 entries to be exact. *A tweets collection* comprise of a tweet per line in format of:

`user_id` (tab) `tweet_id` (tab) `tweet_text` (tab) `time_stamp` (newline)

The `tweet_text` column contains Twitter user writing in normal natural language sentence. The `tweet_text` also contains non-alphabetic and numeric characters. This tweets collection is the main text corpus to search whether a twitter user has entered a misspelled location in the tweet. Tweets collection has total of 400K tweets — 384,323 entries to be exact.

3 Preprocessing

In my initial observation to *a gazetteer* and *tweets collection* dataset, I hypothesize that the amount of data that has to process will be enormous in such that every tweet entry comparing against every location entry. This is equal to $384,323 \times 1,294,704$ amounts of iterations. Furthermore, if I take in consideration of approximate string matching algorithm computation and, make assumption that each iterations takes 1 second then it will take approximately 15,778 years to complete the task. Therefore I work on strategies to preprocess the raw data in justifiable running time frame. The following subsections details on how I make treatment to dataset.

3.1 Treatment to gazetteer

As this is my main dictionary to determine a given string is a location, it is the key knowledge dataset. Besides, the gazetteer dataset contains an entry with shorter length and usually a keyword. These keywords are not part of speech or sentence. Each entry has characteristic of complete information entity on what it is meant of. Based on this analysis, I apply the following operations on the gazetteer dataset.

(i) Remove non-alphabetic characters except space character

(ii) Remove location name with less than 3 characters

(iii) Covert all characters to lower case

(iv) Sorted in natural ascending order

Justification for removing location name with less than 3 characters is that it is hardly able to make spelling mistake with fewer characters. For instance, location name 'Y', 'Ii', 'Ko' and so on. Table 2 illustrate the gazetteer preprocessing treatment.

3.2 Treatment to tweet collection

For tweet corpus, I partition the tweet collection into 1% of total size. This results in $\approx$ 3843 tweet entries per file and total of 100 individual files. The key strategy is that later I can process these files simultaneously. Unlike gazetteer dataset, tweets are written by Twitter user in part of speech manner or natural sentence. With this hypothesis, I apply the following operation to tweets dataset.

(i) Remove non-alphabetic characters except space character

(ii) Covert all characters to lower case

(iii) Normalise the tweet text for common English stop words such as articles and prepositions

Table 3 illustrate the treatment to a tweet.

3.3 Tokens and multi-word locations

Tokenization always apply to tweet text. On the contrary, the gazetteer dataset is never tokenize. The rationale behind no tokenization for location name is that it may loose information if I break down location name into individual words. For example, if I tokenize 'San Francisco Pass' into space delimited token then it ends up 'San', 'Francisco' and 'Pass'. In this case 'Francisco' may be still keyword for a location name but 'San' and 'Pass' become more general words. In nutshell, there are two approaches in handling token:

(a) if single word location matching, tweet text is tokenized into space delimited chunks — Table 5

(b) otherwise multi-words location matching (default setting) and tweet text is tokenized into the same length as location name — each chunk increment at each tweet words Table 4.

4 Methods and their effectiveness

After preprocessing, string matching is performed by using 5 different methods of approximate string matching process. In the following subsections, I will discuss about how I apply these methods and their effectiveness to this project scope.

4.1 Edit Distance, N-Gram Distance

Global Edit Distance(GED) and N-Gram Distance(NGM) are calculated on tweet text in single word token or multi-words token matching to each location entry. This two methods are sensitive to length of the string. For instance GED(`m=1 and i,d,r=-1`), the longer the string, the higher score and highest is the best. Vice versa for NGM and Levenshtein distance[LVS⇒GED(`m=0 and i,d,r=1`)], the shorter the string, the lower score and lowest is the best. Table 1 illustrate this behaviour. In this case, both candidates are valid for approximate string matching. However, GED will be C2 and, NGM and LVS will be C1 for the *best match*.

4.2 Soundex

On the other hand, Soundex is not sensitive to string length. In Soundex, the initial character has major influence on matching and it is not working well with single word location[1] matching.

Edit Distance, N-Gram Distance and Soundex methods work more effectively with string length of similar size — Table 4 vs Table 5 — and require tokenization. One advantage with tokenizing is that it is relatively easy to figure out term matches. However, there are more iteration require to process each tokens.

4.3 Local Edit Distance

Local Edit Distance(LED) does not require tokens. It is looking for the best local substring match. Therefore, it can be used to compare two strings of vastly different lengths [Smith and Waterman, 1996]. The disadvantage of LED is that it requires back tracking to know which region has matched the query term. However, the fact that it does not require tokens, it processes much faster.

4.4 Neighbourhood Search

For Neighbourhood Search, I use Agrep tool. It works similar to LED in such that it identifies strings that contain a substring which is matching to a query [Wu and Manber, 1992]. There are two exceptions compare to the other methods:

(i) Inverse searching — I instead query each location keyword in tweet corpus.

(ii) Tweet corpus is not normalised.

Justification for none normalise tweet corpus is that Agrep is expecting a File as corpus for searching and it is looking for group of subsequence matches anyway. Only the case-insensitive search option '`agrep -i`' is crucial. The disadvantages of using Agrep are that it lacks of matches score and I have to predetermine how much error I would want to allow.

[1]Location name is never tokenize

4.5 High-pass and low-pass filter

Since the project objective is searching misspelled location, I may not want to have best matches. Rather, 1 less from best matches. One approach is to filter the best matches output. For instance, `GED(m=1 and r,d,i=-1)`, I can apply low pass filter as higher score are better matches and chances of misspelled is low. The same goes for N-Gram Distance and I can apply a high pass filter. However, these algorithms are parameters dependant and it is hard to make a judgement call for the threshold value.

5 Evaluation

For this report, I manage to process and analyse 5% of tweets corpus ($0.05 \times 384,323 \approx 19,216$) for each methods.

5.1 Precision

In gazetteer, *San Francisco* appears as a location name. In Figure 1, I observe the approximate matching of misspelled *San Francisco* for each methods. I justify *San Fran* as false positive because it is rather informal abbreviation. In this case, the precision breakdown as follow:

Methods	Precision
Global Edit Distance	1/2
Local Edit Distance	1/3
N-Gram Distance	0
Soundex	0
Neighbourhood Search	0

5.2 Limitation

However, the evaluation is still limited. The precision is still subjective because it is based on the best **1** match of each methods output. For given query token, it may match multiple locations and score can be tied. At this point, I have not learnt yet for how to deal with score tie breaker between matching locations.

6 Conclusions

Searching the exact match is straightforward. All 5 approximate string matching methods do reasonably well for finding exact matches. However, finding misspelled locations in the tweets is not straightforward. This conclude that approximate matching a specific string in big data is challenging. It becomes more subjective and no ***one*** perfect method to solve the problem. This enlighten me insight of how I need to prepare and deal with real world data analysis.

7 References

Needleman, Saul B. and Wunsch, Christian D. [1970]. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". Journal of Molecular Biology 48 (3): 44353. doi 10.1016/0022-2836(70)90057-4

(Originally in Russian, published in English as:) Levenshtein, Vladimir I. [1966]. "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady 10 (8): 707710.

Smith, Temple F. and Waterman, Michael S. [1981]. "Identification of Common Molecular Subsequences. Journal of Molecular Biology 147:195197. doi:10.1016/0022-2836(81)90087-5

Kondrak, Grzegorz [2005]. "N-Gram Similarity and Distance". In Proceedings of the 12th international conference on String Processing and Information Retrieval (SPIRE'05), pp.115-126, Buenos Aires, Argentina

Zobel, Justin and Dart, Philip [1996]. "Phonetic String Matching: Lessons from Information Retrieval". In Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR'96), pp. 166-172, New York, USA.

Wu, S. and Manber, U. [1992]. Fast text searching allowing errors. Communications of the ACM, 35(10):83-91.

8 Appendix

Global Edit Distance

(1) @trapped it's really good!!! It is made by a brewery in San Fransisco... It has a hint of watermelon but it is not over powering!!

(2) @adinaholistics Heard you cleaned up the beach in San Fran. Nice Work!

Local Edit Distance

(1) @trapped it's really good!!! It is made by a brewery in San Fransisco... It has a hint of watermelon but it is not over powering!!

(2) @adinaholistics Heard you cleaned up the beach in San Fran. Nice Work!

(3) @sammytjames Totally agree with your comment re: " San Fran " :)

Soundex
none

N-Gram Distance (bi-Gram)
none

Neighbourhood Search (Agrep)
none

Figure 1: Misspelled San Francisco

C	Tweet token	Location	GED	NGM	LVS
1	san fransisco	san francisco	11	4	1
2	san fransisco has	san francisco pass	12	10	3

Table 1: Best Match Subjective to String Length

Raw	Normalise
Sanford-Springvale Chamber of Commerce	sanford springvale chamber of commerce
519 / Wrigley Volunteer Fire Department	519 wrigley volunteer fire department
"Rebecca of the Well" Fountain	rebecca of the well fountain
A and K	a and k

Table 2: Preprocessing of Gazetteer Locations

Raw	Normalise
@trapped it's really good!!! It is made by a brewery in San Fransisco... It has a hint of watermelon but it is not over powering!!	trapped it's really good made brewery san fransisco has hint watermelon over powering

Table 3: Preprocessing of a tweet

Multi-word tweet tokens	Location	GED	NGM	SDX
`trapped it's reall`	`san francisco pass`	-12	31	1
`it's really good m`	`san francisco pass`	-12	33	1
`really good made b`	`san francisco pass`	-13	33	0
`good made brewery`	`san francisco pass`	-15	33	0
`made brewery san f`	`san francisco pass`	-10	25	2
`brewery san fransi`	`san francisco pass`	-8	20	0
`san fransisco has`	`san francisco pass`	12	10	4
`fransisco has hint`	`san francisco pass`	0	16	0
`has hint watermelo`	`san francisco pass`	-13	31	0
`hint watermelon ov`	`san francisco pass`	-11	31	1
`watermelon over po`	`san francisco pass`	-11	28	0
`over powering`	`san francisco pass`	-15	25	0
`powering`	`san francisco pass`	-14	23	0

Table 4: Processing multi-word tweet tokens

Single word tweet tokens	Location	GED	NGM	SDX
`trapped`	`san francisco pass`	-12	20	1
`it's`	`san francisco pass`	-14	19	0
`really`	`san francisco pass`	-15	21	0
`good`	`san francisco pass`	-16	19	0
`made`	`san francisco pass`	-16	19	0
`brewery`	`san francisco pass`	-16	22	0
`san`	`san francisco pass`	-12	14	2
`fransisco`	`san francisco pass`	-2	12	0
`has`	`san francisco pass`	-14	16	0
`hint`	`san francisco pass`	-16	19	1
`watermelon`	`san francisco pass`	-12	25	0
`over`	`san francisco pass`	-16	19	0
`powering`	`san francisco pass`	-14	23	0

Table 5: Processing single tweet tokens