COMP90049 Report: Analysis of Spelling Correction Methods

Anonymous

1 Introduction

This report examines a diverse set of spelling correction techniques and compares their respective performances on a particular dataset from UrbanDictionary. Spelling correction can be treated as an approximate string matching problem.

2 Dataset

The data consisted of misspelled words, the correct spellings of the respective words, and a "dictionary" of correctly spelled words. These words were identified from UrbanDictionary by Saphra and Lopez (2016). The words are formed from Latin characters but are not necessarily English words.

This dataset has several deficiencies, which could produce false negatives or false positives in the results. For example, "guilford" (a "correct" entry) doesn't appear in dictionary but "guildford" (a "misspelled" entry) does. The following table collates these shortcomings:

misspelled entries in dictionary	*171*
correct entries **not** in dictionary	*108*
correct entries same as misspelled ones	*9*

Table 1: Deficiencies in the provided dataset

Further calculation revealed that 275 (out of 716) words are in some way "deficient". This is close to 38.4% of the data. Since this is such a high percentage, precautions should be taken before making any critical conclusions from this study.

3 Methodology

Three different approximate string matching approaches were used for this problem. This was done in order to get a broader scope of results and to appreciate the respective strengths and weaknesses of each technique.

3.1 Global Edit Distance

The Needleman-Wunsch Algorithm was used to calculate the Global Edit Distance (GED) between the misspelled word and a word in the given dictionary. It returns the maximum possible "score" which represents the GED between the two words. This dynamic-programming algorithm was selected in order to improve the run time complexity of the otherwise lengthy process. Regardless, this program took the longest to execute (i.e., approximately 7 hours for only 716 words).

A simple `Python` script was used to execute the algorithm, calculating the GED between each word in the given file of misspelled words and each word in the dictionary, returning a list of dictionary words with the largest GED from each respective misspelled word.

3.1.1 Parameters

After careful consideration, the parameters $(m, i, d, r) = (+1, -1, -1, -1)$ were chosen over the Levenshtein distance parameters $(m, i, d, r) = (0, -1, -1, -1)$. With these parameters, the difference between a match and a non-match character is further exaggerated. This could be useful when searching for "similar" strings.

3.2 Neighbourhood Search

3.2.1 Implementation with agrep

This search algorithm was implemented with a `bash` script which utilised the UNIX command-line utility `agrep` due to its impressive efficiency. The code used the `-#` flag in the `agrep` command in order to define the neighbourhood limit, `k`. `agrep` was executed on each misspelled word with `k`-values starting from `0` and incremented by `1`, until at least one match from the dictionary (a neighbour) was found.

All the matches for that specific misspelled word were then outputted to standard output. This allowed for each misspelled entry to have at

least one result without requiring preprocessing to determine the smallest value of `k` needed to guarantee a match.

3.3 Phonetic Matching

3.3.1 Implementation with Soundex

The phonetic matching algorithm was implemented with a `Python` program, which utilized Soundex to encode each word in the dictionary into a four-character code. This encoding is meant to be shared amongst "like-sounding" words. The program maps each encoding to the list of words that translate to it. Then, the corresponding list for each given misspelled word is returned. It is possible for an empty list to be returned. Given that all the dictionary information is cached, the overall time complexity is very efficient.

3.3.2 Truncation

It was decided that the truncation of Soundex codes to four characters should be kept in order to retain the simplicity of exact indexing. Having to account for "similar" codes introduces another approximate string matching problem where additional techniques (eg. edit distance) would be required. Even though it may give a better solution, this could introduce unnecessary complexity instead.

4 Evaluation Metrics

In the initial stages of the project, all methods returned a single answer chosen at random from the pool of appropriate solutions. This was repeated 10 times and the accuracy for each run was calculated. (see Table 2).

#	GED	`agrep`	Soundex
1	85	110	2
2	80	105	3
3	89	115	3
4	83	107	4
5	84	118	6
6	79	106	4
7	87	102	4
8	83	109	7
9	84	110	2
10	85	109	4
Average	83.9	109.1	3.9

Table 2: Accuracies of each method over 10 runs

Due to the low accuracy results (only 0.5% - 15.2% of the total words) and consistent presence of random factor(s), it was difficult to reliably compare the respective algorithms based on accuracy.

Thus, precision and recall were more appropriate evaluation metrics since they are designed to work on lists of results rather than just one. The algorithms were then amended accordingly. This reduced the unaccountable variation in results.

4.1 Precision

Precision is determined by calculating

$$\frac{|correct|}{|attempted|} \tag{1}$$

where *correct* is the correct words the method obtained and *attempted* is all of the obtained words. It is used to determine the "proportion of retrieved records that are relevant".

A technique that returns not only correct answers, but also many wrong answers is inadequate. It would be difficult to determine an actual relevant solution. Thus, determining precision is more useful in this regard.

4.2 Recall

Recall is determined by calculating

$$\frac{|correct|}{|total|} \tag{2}$$

where *correct* is the correct words the method obtained and *total* is the misspelled words. It evaluates the "proportion of relevant records actually retrieved".

Recall is a method of verifying if a list of solutions contains the right answer at all. If the precision of a procedure is low but it has a high recall, perhaps additional post processing could filter through the results and determine the most relevant solution.

5 Results

A `Python` script was written to calculate the precision and recall of each approximate string matching method.

Method	Precision	Recall
GED	0.0481378262	0.2653631285
`agrep`	0.0457670043	0.3533519553
Soundex	0.0029336621	0.5949720670

Table 3: Results for each method, approximated to 10 decimal places

6 Analysis

Regardless of which evaluation metric was used, the results imply that more than half of the solutions would be incorrect. This proves unsatisfactory for any real world applications.

6.1 GED vs `agrep`

GED has slightly better precision than `agrep`, but only by approximately 5.18%. However, the recall of `agrep` is 33.16% better than GED recall, which is a significant difference. If an algorithm that assists in ranking relevance of all results could be applied, it is possible to obtain the "correct" answer, rendering the difference negligible.

6.2 GED vs Soundex

In this study, Soundex obtained the highest recall but the lowest precision out of the three methods, with a recall value 2.24 times better than that of GED. These results seem to indicate high numbers of false positives for Soundex. This may be a consequence of truncating the Soundex codes. On the other hand, the method with the highest precision is GED. It obtained a precision approximately 16.4 times better than Soundex. Interestingly, it also acquired the lowest recall.

Choosing the right algorithm for a problem would require determining a suitable trade-off between precision and recall. In situations where false positives may be tolerated better, Soundex would be the better option. Additional techniques like using a similarity metric could be used to improve the precision of Soundex. However, if there is no reliable way of filtering through the results, GED is more likely to return a correct answer.

6.3 `agrep` vs Soundex

`agrep` is significantly more precise than Soundex (i.e., 15.6 times better). Recall value of Soundex is only 1.68 times better than `agrep`.

This infers that even though Soundex is more likely to have the "correct" answer in its list of solutions, there is a much lower chance of obtaining the actual answer compared to using `agrep`, provided the selection of final answer is randomised. Unless a sufficiently efficient algorithm that could rank the relevance of all results is available, the Soundex algorithm is likely to be least applicable in real world situations.

7 Conclusions

In conclusion,

Given that all three methods do not necessitate any user input, perhaps some modifications could be applied to make these algorithms user-assisted.