COMP90049 Report

Anonymous

1 Introduction

This report examines a diverse set of spelling correction techniques and compares their respective performances on a particular dataset from UrbanDictionary

Explain that this is an approximate string matching problem. – maybe put this in methodology?

- include stats of misspelled words in general - include history of spell checkers / spelling correction

aims to apply multiple spelling correction methods on a given data set. The methods are then analysed and compared based on their respective performances.

2 Dataset

The data provided consisted of misspelled words, the correct spellings of the respective words, and a "dictionary" of correctly spelt words. These were identified from UrbanDictionary by Saphra and Lopez (2016). The words were constructed from Latin characters but were 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 illustrates these shortcomings:

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

Table 1: Deficiencies in the provided dataset

Further calculation disclosed 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 number, some precaution should be taken before making any critical conclusions from this study.

3 Methodology

Three different approaches were chosen to attempt to solve the problem. This was done in order to get a broader scope of results and to attempt to incorporate the advantages from each technique into a singular better solution for this problem. /@@@@@/

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 score is determined by the chosen parameters. This dynamic-programming algorithm was chosen in order to improve the run time complexity of the otherwise lengthy process. Regardless, this program took the longest to execute, around 7 hours for just 715 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 consideration, the Levenshtein distance with parameters of $(m, i, d, r) = (+1, -1, -1, -1)$ was chosen to be applied in this implementation. With this parameters, strings which are more "similar" have a higher GED score.

3.1.2 Comparison with Local Edit Distance

Local Edit Distance (LED) is a ... used for more appropriate for substring matching, but due to

the nature of this problem, it will not be very effective in getting good solutions.

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 implementation of the phonetic matching algorithm was a `Python` program which used 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 evaluate to that specific translation. Then, it returns the corresponding list for each given misspelled word. 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

Although the truncation of Soundex codes to four characters may not be as useful in an approximate string matching problem as compared to one that required a more exact solution, it was decided keep the truncation in order to retain the simplicity of exact indexing.

Having to account for "similar" codes will introduce another approximate string matching problem where some technique (eg. edit distance) would be required. Even though it may give a better solution, this could introduce unnecessary complexity instead of solving the actual problem.

4 Evaluation Metrics

Precision and recall seemed more appropriate evaluation metrics than accuracy since all the algorithms returned a list of solutions as opposed to a single answer.

Also calculated the accuracy for each algorithm, where the chosen word is a random answer from the list of answers.Values were like – insert table –, which is – insert percentage –, which is quite low. Because it is quite low and the random picking of answers is not very reliable, decided to return a list instead and do analysis on that. In that case, less unaccountable variation in the results.

4.1 Precision

Precision is determined by calculating

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

where *correct* is all the correct words the method derived and *attempted* is all of the derived words.

Determining the precision is useful because even though a technique may return the correct answer, that solution is useless if many other wrong answers are returned as well. It would be quite difficult to discern the actual desired solution from such a large pool.

In this study, the method with the highest precision is the Global Edit Distance. This makes sense because

On the other hand, Soundex obtained the lowest precision out of the three.

4.2 Recall

Recall is determined by calculating

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

where *correct* is all the correct words the method derived and *total* is the misspelled words.

Recall is a method way of verifying if a certain method will return the right answer at all. If the precision of a procedure is low, but it has a high recall, perhaps additional post processing could help sieve through the results and determine the most relevant one.

Amongst the three techniques covered in this report, Soundex had with the highest recall. This may be because

Interestingly, the lowest recall was acquired by Global Edit Distance.

5 Results

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

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

Table 2: Results for each method, approximated to 10 decimal places

5.1 GED vs `agrep`

GED has better precision but not that much better, only approximately 5.18% better, with respect to agrep's precision. However, the recall of `agrep` is 33.16% better (wrt to GED recall), which could significantly affect any dataset. If there exists some algorithm that could help rank the relevance of all the results, it could be quite likely to get the "correct" answer, rendering the difference in precision negligible.

5.2 GED vs Soundex

Eventhough the recall value of Soundex is 2.24 times better than GED, the precision of GED is approximately 16.4 times better than Soundex. This infers that although Soundex would most likely have the correct answer in its list of solutions, if selection of the final answer is random, there is a much lower chance of getting the actual answer than if GED was used. Due to this, unless a sufficiently effective algorithm that could help rank the relevance of all the results is available, the Soundex algorithm is most likely useless for real world application.

5.3 `agrep` vs Soundex

`agrep` is also significantly better at precision than Soundex, 15.6 times better. Soundex's recall value is only 1.68 times better than `agrep`. That would most likely make the most difference in a large dataset, maybe with some extra information that could provide some context to the problem.

6 Conclusions

Include discussion + possible improvements Talk about how all of these algorithms are quite naive and do not include any user input, which is quite important in the context of spelling correction (give an example as well as a reference). Some context could be the whole sentence typed by the user - what is grammatically correct? Also use user data to determine which words are more commonly followed by which others.

Text,[1] with footnotes at bottom of page. Text of the subsection with citations such as Note that the citation style is defined in the accompanying style file; it is similar to AAAI house style. You may use other (formal) citation styles if you prefer. Text of the (see Table 2).

Soundex - use better code values? agrep is a nice middle ground thing

[1] some Footnote text