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

/@@@@@@@@@@@@@@@@@@/ – reword this Here, a list of words entries that are ?correct? We can break our input into words substrings that we wish to match, and compare each of them against the entries in the dictionary A word item in the input which doesn?t appear in the dictionary is misspelled A word item in the input which does appear in the dictionary might be correctly spelled or misspelled (probably slightly beyond the scope of this subject) /@@@@@@@@@@@@@@@@@@/

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

"a number of headwords taken from UrbanDictionary that have been automatically identified as being misspelled (Saphra and Lopez, 2016)."

all the words are "english" and are latin characters. however they are not necessarily English which is why we use the provided dictionary as reference.

explain where the data set is from. explain how it is not "clean". give some numbers related to this. number of misspelled entries that do appear in the dictionary: 171 number of correct entries that don?t appear in the dictionary: 108 - eg. "guilford" (from correct.txt) doesn't appear in dictionary but "guildford" (from misspell.txt) does. number of correct spellings are the same as the misspelling': 9 number of deficient data in dataset: 275 (out of 716) – 38.4% of data is ?dirty?.

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. This dynamic-programming algorithm was chosen in order to improve the run time complexity of the otherwise lengthy process. It returns the maximum possible "score" which represents the GED between the two words. This score is determined by the chosen parameters.

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 Methods

4.1 Precision

- explain what it is and what it's good for (with some references)

4.2 Recall

- explain what it is and what it's good for (with some references)

4.3 Results

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 1).

Text of the subsection with citations such as

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

Table 1: Results for each method, approximated to 10 decimal places

(formal) citation styles if you prefer.

4.3.1 GED vs Neighbourhood Search

4.3.2 GED vs Soundex

4.3.3 Neighbourhood Search vs Soundex

5 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.

Soundex - use better code values?

[1] Footnote text