COMP90049 Project 1: Spelling Correction

1. Introduction

As is universally acknowledged, approximate string searching is extensively applied in many fields. Such as predicting words when people are typing on no matter computers of mobile phones. One of these is to be used for spelling correction. This report aims to develop a spelling correction system which is able to correct the misspelled words from the dataset, then provide the prediction of possible correction according to the dictionary, and finally compare the prediction with the correspondingly correct words as for the judgement of performance. In addition, this report discusses these popular methodologies as well as analyses about the evaluation using accuracy, precision and recall parameters as metric.
python

2. Data Set

Three files are included in this data set, which are *misspell.txt, dictionary.txt*, and *correct.txt*. These data are part of UrbanDictionary [1] headwords. The data in the *misspell.txt* is automatically identified misspelled (Saphra and Lopez, 2016). It comprises of 716 misspelled words, which are formatted as string by lines. In order to get each clean string, this project use rstrip() to get rid of the return symbol located at the end of each string. The data in the *correct.txt* is as the same sequence as the misspelled data. As for the *dictionary.txt*, it is a sorted list of more than 40K words. Additionally, *distance-stdout.txt* and *ratio-stdout.txt* are the files that store the output details of two methodologies

3. Approximate String Searching Methodology

3.1 Global Edit Distance

Global Edit Distance is known as the minimum cost of any sequence of edit operations that it will take in order to transform one string to another. (Hasan et al, 2015). The measurement includes m (for match), i (for insert), d (for delete), and r (for replace). For each measurement, a score is defined according to the importance of its performance.

Levenshtein is one of the well-known approaches that implements the Global Edit Distance, came out by Levenshtein (Levenshtein, 1966). As in this project, two Levenshtein methods are used for the spelling correction of the headwords.

3.2 Standard Levenshtein Distance

Detailed in this method, the parameters for each measurement are as follows:

$$m = 0; i = 1; d = 1; r = 1.$$

In order to implement the method, and as a convenient advantage of Python language is, this project is able to import a package called Levenshtein. After reading the given three files, for each headword in *misspell*, calculate the standard Levenshtein Distance with each headword in *dictionary*, using the Levenshtein. distance(). The headwords that have the top ten minimum distances are the predicted words for this particular misspelled headword.

This process clearly explains the reason for parameter deciding. That is, for each string, the best measurement is matching two characters, while it will get equally penalty for insertion, deletion and replacement. Therefore, the headwords with minimum score are always the strings with the most matching characters.

3.3 Improved Levenshtein Distance

Similar with the standard one, this method is implemented with the help of Levenshtein. ratio(). Different from the standard one, this method changes the score of parameters as:

$$m = 0; i = 1; d = 1; r = 2.$$

[1] http://urbandictionary.com

Here, this report changes the score for replacement into 2. Rather than insertion and deletion, the measurement of replacement will receive a double more penalty. This means that replacement is more welcomed. Therefore, it is more likely to match two equal length strings, rather than insert or delete its characters.

Since the most common mistake of misspelling is the typo. People may miss-press the adjacent key while using the keyboard to input. That is to say, most misspelled words are have the equal length with the correct one. Therefore, this changing is able to improve the accuracy of the spelling correction.

4. Evaluation Metric and Improvement

The effectiveness of the approximate string searching approaches is determined by evaluation metrics. The three basic evaluation metrics are accuracy, precision, and recall.

This report only implements two evaluation metrics, which are precision and recall.

Precision	15.09%
Recall	35.33%

Table: Result of Standard Levenshtein Distance

As for the improved method:

Precision	16.29%
Recall	21.22%

Table: Result of Improved Levenshtein Distance

The precision of the method is improved.

Example:

The following screenshot shows the different output of the same misspelled word using the standard and improved Levenshtein Distance.

1) Standard Levenshtein Distance:

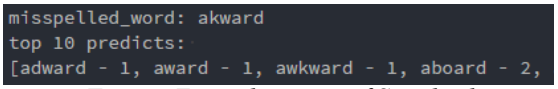

Figure: Example output of Standard Levenshtein Distance

While implementing the standard one, there are three best matches for the misspelled word '**akward**', which are 'adward', 'award', and 'awkward'. Furthermore, the corresponding correct headword in *correct.txt* is '**awkward**'. Although the correct 'awkward' do appear in the predicted word list, it is placed at the end of list. It can be easily ignored because people always care about the first prediction especially when they only want one predicted word.

Also, as it shows, since there are three predictions and only one correct word, the accuracy of the prediction is 1/3.

2) Improved Levenshtein Distance:

The figure below is the same misspelled headword with the standard one.

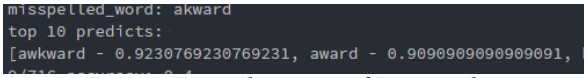

Figure: Example output of Improved Levenshtein Distance

After implementing the improved method, the only one predicted word that provided for the same headword '**akward**' is – '**awkward**', which is exactly the same as the correct one in the *correct.txt*. So, the accuracy of the prediction is 1.

As we can see, misspelled word 'akward' is verly similar to the correct 'awkward'. The typo like this is very commonly seen in daily life.

5. Conclusion

This report implements two different approximate string searching methods to correct the misspelled words of UrbanDictionary. And the improved one is more likely to suit for this situation.

The disadvantage of these two methods is that they are not very suitable for when comparing two strings of very different lengths. Such as matching a word with a sentence.

Actually, I do implement the Local Edit Distance methodology, but it is not very suitable for this case. Because the fact is that the misspelled words are mostly typos. They only have slight difference with the correct one. However, I still added the code file in to the zip, as a proof of my work.

6. Bibliography

http://urbandictionary.com

Naomi Saphra and Adam Lopez (2016) Evaluating Informal-Domain Word Representations with UrbanDictionary. In Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP, Berlin, Germany. pp. 94–98.

Levenshtein, V. I. (1966, February). Binary codes capable of correcting deletions, insertions, and reversals. In Soviet physics doklady (Vol. 10, No. 8, pp. 707-710).

Syeda Shabnam Hasan, Fareal Ahmed and Rosina Surovi Khan, 2015. Approximate String Matching Algorithms: A Brief Survey and Comparison. International Journal of Computer Applications (0975 – 8887) Volume 120 – No.8