COMP90049 Project 1: What weird spellings! Are people crazy?

Anonymous

1 Introduction

Auto-correction of misspelled word is a common but useful feature that exist in browsers, editing tools and IDEs. In this report, I analyse and compare various spelling-correction methods, over a sample of most commonly misspelled words on UrbanDictionary [1].

The chosen algorithms are Global Edit Distance (GED), Local Edit Distance (LED), Neighbourhood Search (NS), N-gram Distance (Ngram) and Soundex.

The overall performance suggests that Damerau-Levenshtein method is the best for simple word-correction tasks.

2 Dataset Description

The dataset contains three text files: 716 commonly misspelled words, 716 correctly spelled words and a dictionary with 393,954 English words.

3 Evaluation Metrics

To compare results from different algorithms, I employ four metrics: **accuracy, precision, recall and time**. The reasons behind are as follows: firstly, the top priority is to find the correct word for typos, therefore, accuracy of the predicted word is an important metric to consider. Secondly, sometimes the typos are very ambiguous so that there are multiple correct answers according to the algorithms. In that case, a list of suggested words that contains the correct word is more desirable for users than a single predicted word. Hence, precision and recall are also important metrics. Thirdly, on-line correction, i.e. correct as we type, is widely applied nowadays in search engines and word processors. In on-line auto-corrections, having a fast algorithm with reasonable accurate suggestions is necessary.

[1] http://urbandictionary.com

4 Methodology and Algorithms

4.1 General Algorithms

Algorithm 1 Word Correction

```
1: function ALGO(misspell_lst, dict_lst)
2:     for each misspell word do
3:         for each dict word do
4:             calculate value
5:         Prediction ← word/list with max/min value
6:     Eval(Prediction, Correct)
7:
8: function EVAL(Prediction, Correct)
9:     return accuracy, precision, recall
```

The overall idea is to use some algorithm to calculate some value between each misspelled word and words in the dictionary, and then choose the best one(s) as the predicted word or the predicted list. Subsequently, the suggested word(s) are evaluated by Eval function to compare the performance among algorithms. The algorithm is illustrated in pseudo-code algorithm 1.

4.1.1 Global Edit Distance (GED)

For vanilla GED method, I use the standard Levenshtein distance: $(m, i, d, r) = (0, 1, 1, 1)$ (Levenshtein, 1966; Needleman and Wunsch, 1970). Additionally, an extended version has also been employed: Damerau-Levenshtein method (DL-GED)(Damerau, 1964). DL-GED method counts transpositions as a single edit. For example, $levenshtein(apple, apepl) = 2$ while $DL - GED(apple, apepl) = 1$. The justification here is that often typos are merely alphabets in wrong orders due to fast typing behaviour. Consequently, DL-GED should be more desirable than vanilla GED method.

4.1.2 Local Edit Distance (LED)

Smith-Waterman algorithm with parameters $(m, i, d, r) = (1, -1, -1, -1)$ is used when

applying LED (Smith and Waterman, 1981). However, since the algorithm is better for substring match when comparing two strings of different length, I test it empirically for completeness rather than focusing much on the performances.

4.1.3 N-gram Distance (Ngram)

N-gram uses the same idea of GED but in this scenario, it is empirically faster than GED with slight cost of memory. By storing the N-gram lists in the memory, we avoid matrix updating for (each misspelled word, each dictionary word) pair, as we would have to perform in GED method. I mainly test Ngram with n=2 even though I design the algorithm with the flexibility of N-gram parameter. (Kondrak, 2005)

Algorithm 2 N-gram Distance

```
function N-GRAM(n=2)
    for each dictionary word do
        generate n-gram list, store in a dict
    for each misspelled word do
        calculate n-gram distance
```

4.2 Majority Vote

For the above algorithms, theoretically there is no single algorithm that dominates others. It is possible that some algorithms perform better than others on a particular sub-dataset. Hence, I aggregate the scores from the above algorithms (except for LED), including Soundex to vote for the best choices.

However, although the notion here is to improve the accuracy and precision, the cost of majority vote is time.

4.3 Phonetics: Soundex

Phenotics is another important feature in English language (Zobel and Dart, 1996). In terms of typo correction, phonetics adds on another useful tool to the general algorithms. The idea is to capture typos that have similar sound with the correct words. For example, both 'adidas' and 'addidas' have soundex of A332.

However, using phenotics is also problematic. For instance, 'adn' has soundex of A350 while 'and' has soundex of A530. Consequently searching for 'adn' using pure soundex will be incorrect regardless of whether we combine phenotic methods with distance methods. Therefore, I only use soundex as a supporting method.

4.4 Pre-processed Dictionary: Neighbourhood Search (NS)

Instead of directly applying Neighbourhood Search (NS), I borrow the idea of NS to pre-process the dictionary dataset. Conventional NS requires us to generate all variants that utilise at most k changes. For $k \geq 2$ and three potential changes (insertion/deletion/replacements), the possible combinations for a given misspelled word are massive. Besides, most of the generated words are redundant because they will not appear in a dictionary.

Therefore, the approach I take, utilising the idea of NS, is to truncate the size of the dictionary for a given misspelled word before we apply any algorithms. For example, 'adn' has length of 3, and truncated dictionary contains words that have length from 1 to 5. Choice of ± 2 is a relatively reasonable assumption to have the balance between the speed and effectiveness of truncation. This method will significantly improve the efficiency by avoiding redundant distance calculations for dictionary words that are almost impossible to be the correct answer.

Additionally, if we further assume that people are likely to type the first character correctly, then we could further limit the dictionary to same first character only. The improved algorithm could reach empirical speed, yet this assumption is not justified and should be employed with care.

Algorithm 3 NS processing

```
function NS(n=2)
    for each misspelled word do
        len ← calculate word length
        truncated dict ← len +/− n
        other algorithm calculations
```

4.5 Python packages

I implement all of the algorithms in python, but I also compare my code with packages for speed purposes. References are summarized in the following table:

Method	Python Package (pkg)
GED	Jellyfish (Turk, 2016)
LED	Swalign (Breese, 2017)
N-gram	N-gram (Poulter, 2017)
Soundex	Jellyfish (Turk, 2016)

5 Results

Method	Time(s)	Time with NS
GED	390	117
DL-GED	431	197
LED	43155	14385
Ngram	767	513
maj. vote	1648	791
with soundex	1515	829

Table 1: Time consumed to run full data set

Method	Accuracy	Precision	Recall
GED	0.1464	0.0457	0.3533
DL-GED	0.1606	0.0543	0.4212
LED	0.0712	0.0238	0.1852
Ngram	0.1396	0.1008	0.2123
maj. vote	0.1593	0.0535	0.4119
with Sdex	0.1704	0.1129	0.2291

Table 2: Evaluation without NS filtering

Method	Accuracy	Precision	Recall
GED	0.1403	0.0423	0.3512
DL-GED	0.1523	0.0523	0.4019
LED	0.0694	0.0225	0.1752
Ngram	0.1516	0.1057	0.2234
maj. vote	0.1466	0.0441	0.3549
with Sdex	0.1745	0.1129	0.2291

Table 3: Evaluation with NS filtering

Speed is not directly comparable since some of the packages use cython, which is much faster by default.

5.1 GEDs outperform others

Overall, GED-type methods outperform other algorithms in this paper, suggesting that global edit distance is better for typo corrections. As I suggested before, DL-GED method captures the transposition characteristics, and therefore is better than naive GED method.

LED method is not suitable for this task theoretically, and the performance justifies the suggestion.

N-gram method achieves the close-enough accuracy with a surprisingly fast speed (since the others use cython). However, N-gram has it limitation when typo and the suggested word has particular shape. For example, GED can correct 'amazong' with 'amazing' while N-gram suggests 'amazon' because those words have very close N-grams.

5.2 Majority vote is capped by the best algorithm

The performance of majority vote is close to that of GEDs. I suspect that this is because the proposed algorithm in this paper uses a simple structure of aggregating the results. I argue that a better structure, e.g. use algorithm results as features of neural network could possibly improve the performance.

5.3 Those naive approaches are not effective when typos exist in the dictionary

The problem exists in all of the algorithms described in this paper. If the typo itself is a correct word in the dictionary, then the best match is the word itself. For instance, 'oaky' is a typical typo of 'okay', but it is also a correct word. Therefore, when we search in the dictionary, the best match for 'oaky' is 'oaky' itself regardless of which naive algorithm employed.

5.4 NS filtering gains significant efficiency without losing effectiveness

Table 1 suggests that NS-filtering save at least half of the time. Meanwhile, table 2 and 3 suggests that the performance after NS-filtering remains solid.

6 Discussions

From the results we can clearly observe that exhaustive search is not helpful. A natural way to improve typo-correction methods is to consider semantic analysis, i.e. word groups or sentences. The reason behind is that words are themselves less meaningful without context. Therefore, when the algorithms scan words, they could consider "**semantics**" to "guess" what the user is trying to type. As a result, the algorithms become both quantitative and qualitative.

In addition to semantic analysis, **syntactic** analysis can also help. English has its own grammars that individuals generally follow. We could further improve our "guess" by adding grammar component into algorithms.

Both semantic and syntactic analysis can be achieved using **word2vec**-type models (Mikolov et al., 2013).

Eventually, those improvements are to limit the search space down to a list of words that

are reasonable under certain context and fit the conventional grammar. The ultimate goal is to produce both effective and efficient algorithms, rather than naive exhaustive search.

7 Conclusion

I implement 5 misspelled-correction methods. The results indicate that Damerau-Levenshtein method with Neighbourhood-Search performs the best. However, the performance should be greatly improved with semantic and syntactic analysis since context is important in typo-correction.

References

Breese, M. (2017). swalign. https://github.com/mbreese/swalign.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176.

Kondrak, G. (2005). N-gram similarity and distance. In *International symposium on string processing and information retrieval*, pages 115–126. Springer.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Needleman, S. B. and Wunsch, C. 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):443–453.

Poulter, G. (2017). ngram. https://github.com/gpoulter/python-ngram.

Saphra, N. and Lopez, A. (2016). Evaluating informal-domain word representations with urbandictionary. *Proceedings of the 1st Workshop on Evaluating Vector-Space Represenations for NLP, Berlin, Germany*, pages 94–98.

Smith, T. and Waterman, M. (1981). [a]identification of common molecular subsequences. [o] j. *Molecular Biology*, 147:195–197.

Turk, J. (2016). jellyfish. https://github.com/jamesturk/jellyfish.

Zobel, J. and Dart, P. (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*, pages 166–173. ACM.