Assignment 1 - Spell Correction

Reza Barzegar
barzegar@uwindsor.ca
University of Windsor
Windsor, Ontario, Canada

Hamed Waezi
waezi@uwindsor.ca
University of Windsor
Windsor, Ontario, Canada

Abstract

This assignment is about solving the misspelled correction problem using Levenshtein algorithms[2]. Given a dictionary and a corpus of misspelled words alongside the corresponding correct spelling, calculate the minimum editing distance (MED) of each misspelled token against the tokens of the dictionary and check if the correct spelling exists in the K least distant words which are retrieved from the dictionary. The output of this assignment is the average occurrence of the correct word in the set of the top K least distant words.

Keywords: Levenshtein, minimum edit distance, spell correction, natural language processing

1 Introduction

As the assignment indicated, we used the Wordnet dictionary and the Birkbeck spelling error corpus in this project. The original Birkbeck corpus includes many different rules for the extraction of the required data; such as the differences between American and British English spellings. Handling this matter was out of the context of this assignment. Therefore, we use the version of Roger Mitton's [3] Birkbeck which includes the correct spelling of each token alongside it.
We used the Levenshtein algorithm to measure the distance between two tokens.

2 Problem solving

In the following sentences, we explain our approach to solving this problem. First, we calculate the minimum edit distance between each misspelled token and every word in the dictionary. Then, we keep the 10 most similar (least distant) dictionary words for each misspelled word. In the final step, we calculate success at 1, 5, 10 for the correct spelling of the misspelled token. In other words, we search the set of most similar words to find the existence of the correct form of the misspelled token. We used Python and Golang programming languages to implement the solution. In python nltk library is used for the Wordnet corpus. Also, by writing Wordnet's words into a file, we used the same dictionary in Golang.
As per the request of the assignment, we reported results for average success at k=1, 5, 10 using the pytrec eval terrier. Assume n is the *length of dictionary*, m is the *maximum length of a token*. The time complexity of the algorithm for each misspelled token is $O(nm^2)$. Considering the time complexity, it is critical reducing the run-time by utilizing different methods such as parallelism. In table 2 you can see the impact of parallelism on this task.

	Python	Go
Success at 1	0.27	26%
Success at 5	0.44	44%
Success at 10	0.50	49%
runtime	14.3 H	18min

Table 1. Table of Results in Python & Golang.

	runtime for 6 instances using python
Sequential	80 seconds
Parallel	17 seconds

Table 2. Table of Comparison between sequential and parallel

2.1 Experiments

Both source codes are available in a single repository on GitHub [1]. The python version of the program includes a *utils* package, data directory, and *main.py* and *main.ipynb*. the *utils* package consists of the Levenshtein algorithm and some functions that create the set of most similar words and success at K calculator. For the Golang program, there are two files in the src-go directory, consisting of utils and main. In order to enhance the speed of the calculations we employed the concurrency features of Golang, known as Goroutines. A Goroutine is a lightweight thread managed by the Golang program. We reported the results in table 1.

3 Discussion

This method is not applicable in spell correction programs due to its time complexity. Furthermore, this method for spell correction misses many misspelled words because it does not consider the context.

References

[1] assignment1 https://github.com/rezaBarzgar/Assignment1 COMP-8730.
[2] Levenshtein https://en.wikipedia.org/wiki/Levenshtein$_{distance}$.
[3] Roger Mitton https://www.dcs.bbk.ac.uk/ roger/.

Conference'17, July 2017, Washington, DC, USA
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn