PeerMark® by Turnitin

Project 1 reviews

Print Options:

☑ Include Questions & Answers ☑ Include Comments ☑ Include All Reviews ☑ Include File Info

Print

HAONAN LI'S PEERMARK REVIEW OF ANONYMOUS'S PAPER (100% COMPLETED)

ASSIGNED QUESTIONS

1. Briefly summarise what the author has done

   The author use three global edit distance scheme to finish a spelling correction task and evaluate the performance of them.

2. Indicate what you think the author has done well, and why

   There are many tables in the report, which is a good method to show the experiment results.

3. Indicate what you think could have been improved, and why

   First, I do not think use one method with three difference parameters is enough for the task, the author even use the algorithm library which makes the workload less. Maybe all experiments can be finished in 1 hour.
   The author use so-called “Top N” approach to get the predicted words, but what if there are more than N most near items? Randomly choose N items from them? Or other strategy.
   Moreover, there is not a demonstrate of the performance of three global edit distance schemes’ difference. What kind of words could be matched by DLD but not matched by LD and NLD? Why DLD could get higher score? No demonstrate. The explanation is simple and unpersuasive. The author display Table 4 but also without an systematic explanation, why Top 5 results in a lower precision?
   For the last part of the report, if something is not a conclusion, do not talk about it in this part. Maybe add a section of future work is helpful.
   As for writing, a. There are some grammatical errors; b. The second reference can be replaced by a footnote; c. Academics used to put the method section before the dataset and evaluation sections.

COMMENTS LIST

No comments added

SUBMITTED FILE INFO

file name	KT_-_Project1.pdf
file size	152.82K

"KT ASSIGNMENT 1 PROJECT REPORT" BY ANONYMOUS

COMP90049 Project 1 Report

Anonymous

1. Introduction

This report analyses the performance of basic global edit distances algorithms applied to a given dataset of misspelled words (Saphra and Lopez, 2016) and a dictionary to test and evaluate their effectiveness.

The aim of this task is to analyze the different outcomes we get, understand their underlying behavior and provide a thorough analysis based on our observations.

2. Dataset

The dataset used for this project is a collection of 716 misspelled words taken from UrbanDictionary[1] with its corresponding pair of intended correct words, for evaluation purposes.

Another dataset of approximately 400.000 words has been provided as our dictionary and our implementation will focus in comparing the misspelled tokens against it to try to successfully predict their corresponding correct word.

As per the project description, these resources are considered real-world data, so our implementation will definitely be challenged by the different situations that could come across when analyzing these data.

3. Evaluation Metrics

As our system was built with the purpose to capture the closest words (in terms of similarity) for every misspelled word, we chose to stablish a "Top *N*" approach (e.g. Top 3 potential candidates). So, under these circumstances we did not find it suitable an Accuracy metric.

Instead, we are prioritizing Precision and Recall as our main indicators of effectiveness. For Precision, we've calculated the number of correct matches among out attempted responses (Top *N*). This will widen our matching possibilities (increase Recall) but as our *N* increases will shorten our Precision considerably. We realize that having this scheme of Top *N* candidates for every word directly impacts the precision but that's something that will be analyzed in the following sections.

This approach gives us the possibility to detect more matches and it's very helpful in situations where we don't have any easy way to choose a preferred word amongst tied candidates.

Another metric that we also include -for evaluation purposes- in our analysis is the time it took to accomplish its goal (Runtime). According to the application scenario, in some cases it may be necessary to evaluate also the time it takes it to do the predictions.

4. Global Edit Distance

In order to compare and evaluate the approximation between the misspelled words and our dictionary we have calculated and analyzed the performance and effectiveness of different variants of the Levenshtein distances (Levenshtein, 1966).

Each of the words from the first dataset (Misspell.txt) was compared against the dictionary using dynamic programming (Wagner-Fischer algorithm) as part of our implementation. The obtained results were relatively efficient, but the final conclusion would vary depending on the intended use and the selected criteria.

As it was expected, comparing the misspelled words against a dictionary produced as a result a large number of potential candidates for every word. According to what we stated before, we've consolidated a Top 3 selection. As there is no easy way to break ties among equally good candidates (without further context at least), we've decided that the final selected Top *N* candidates would be as good as any, so, our implementation considers the first best scores calculated, and only they will be replaced by other better scores, if any, while analyzing the rest of the dictionary.

The following distances were tested, compared and analyzed:

- Levenshtein Distance (LD)
- Normalised Levenshtein Distance (NLD)
- Damerau-Levenshtein Distance (DLD) (Damerau, 1964)

[1] http://urbandictionary.com

4.1. Results

We have used the Java String Similarity Library from Debatty (2014), written in java, to address this issue. Our implementation consisted in generating the necessary lines of codes to go through the provided datasets and calculating the evaluation metrics based on the most suitable candidates for every token. The results obtained are presented below:

Method	Precision	Recall	Accuracy	Runtime (sec)
LD	0.089	0.27	0.150	85.9
NLD	0.097	0.292	0.158	84.59
DLD	0.101	0.303	0.17	567.3

Table 1: Approximated Efficiency (Top 3)

Additionally, for evaluation purposes, we have increased our prediction selection to 5 (for every misspelled word). The results are shown in Table 2:

Method	Precision	Recall	Accuracy	Runtime (sec)
LD	0.066	0.328	0.149	92.76
NLD	0.068	0.34	0.154	92.6
DLD	0.075	0.377	0.169	581.57

Table 2: Approximated Efficiency (Top 5)

As we stated before, we are not considering Accuracy a very suitable metric in this scenario, where we have multiple predictions for each misspelled word. Our implementation selects the best scores, regardless of the applied methodology, as it goes through the dictionary and only replace any selected prediction when it finds a better score than what it already has filtered. This approach has its downside that will be discuss in the Analysis section of this paper. Nevertheless, we've made efforts to include the Accuracy in this evaluation to explore its behavior but should not be taken more than just a reference.

4.2. Analysis

According to the given results in Table 1 and 2, we see that NLD slightly outperforms LD, and DLD does the same with NLD (at a cost of a considerable greater processing time). This shows us that the improvements that NLD attempts to provide by considering the lengths of the tokens, and DLD by considering character transpositions (besides of just measuring the number of edits required for strings similarities) are somehow beneficial for our goal of getting as many matches as we can.

However, the improvement could be arguable as it may not be considered significantly enough to move the balance towards one preferred option.

With the spirit to capture as many matches as we can we had widened our predictions selection from 3 to 5 and quickly notice the effect it had in our results, especially with the overall correct responses (Recall).

Method	Recall (Top 3)	Recall (Top 5)
LD	0.27	**0.328**
NLD	0.292	**0.34**
DLD	0.303	**0.377**

Table 3: Approximated Efficiency, Recall Comparison Highlighted

If we take into account our best measurement, DLD, we can clearly see that it escalated from having a 30% of successful matches (217 matches from 716) to a 37% (270/716), 7% more by just adding 2 more predictions for every misspelled token. As a counterpart, this action doesn't help consolidating a very precise system:

Method	Precision (Top 3)	Precision (Top 5)
LD	0.089	**0.066**
NLD	0.097	**0.068**
DLD	0.101	**0.075**

Table 4: Approximated Efficiency Precision Comparison Highlighted

An important observation we should take into account is that there are many cases where all of the candidates (whether it'd be from our Top 3 or even Top 5) have the same distances from the misspelled word. No word overcomes the others, so under this scenario it's impossible to properly select a unique correct prediction (for Accuracy purposes). The only scenario where we may find real value in the Accuracy metric is if amongst our Top *N* predictions there is any one word -and only one- that surpasses with its score to every other candidate word, but as was pointed out right from the beginning, these datasets resemble real-world raw data, and doesn't have to be necessary clean to be processed.

In many cases, there is even the problem that our whole selection doesn't contain the intended correct word. Instead, our Top *N* Array gets populated with other equally distanced words:

Misspelled	Levenshtein Top 5 Predictions	Correct?
yoru	york	✗
	yore	✗
	yor	✗
	yaru	✗
	toru	✗
wimmin	dimmit	✗
	dimming	✗
	dimin	✗
	diamin	✗
	cummin	✗

Table 5: Examples of equally distanced incorrect words

5. Conclusions

After several tests we could clearly define that amongst the different Levenshtein distances that was compared and analyzed in this report, the Damerau-Levenshtein consolidated as the one that got the best matching results, at the expense of higher processing costs (it took almost 7 times more time to analyze our datasets). Disregarding the performance obtained during the execution and/or the relative comparison made between the selected methods, the application of these methods overall didn't prove to be highly precise, especially when we are considering only 3 predictions. Instead of applying these methods on their own, another possibility would be to make a hybrid implementation that combines other methods (such as N-Gram) to the final candidates in order to test if this could improve our matching ratio. The caveats of these implementations are that there is often a cost involved or a sacrifice to be made in order to improve some key aspect of the task. Whether this metrics would satisfy or not the expectations would depend highly on the deployment scenario.

References

Damerau, F.J. (1964) "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, vol. 7, no. 3, pp. 171-176.

Debatty, T. (2014, April 17). *java-string-similarity*. Retrieved from https://github.com/tdebatty/java-string-similarity

Levenshtein, V. I. (1966) "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, pp. 707-710.

Saphra, N. and Lopez, A. (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.