**COMP90049 Knowledge Technologies Project 2**

|  |
| --- |
| **Twitter Trolls and the Tweeters who Love them** |

**1. Introduction**

To some extent, the birth of the Internet has inexplicably changed the way we humans communicate. This can be observed with the ever so blossoming spectrum of different communication platforms. First came email, blogs, forums, then social media, followed closely by instant messaging, as well as the more recent uprising of live stream chatting [CITATION NEEDED].

However, akin to the sprouting of the methods of communicating, new types of communicators have evolved along, with some of them erring to the negative side. One such are termed ‘Internet Trolls’, which describes an individual posting comments or some other form of online content, with the intention of attacking, offending, disrupting, or to cause trouble within the community [https://www.lifewire.com/types-of-internet-trolls-3485894].

This project focuses on ‘trolls’ originating from one of the more popular social media platforms, Twitter. With the help of Machine Learning methodologies, this project attempts to analyse and justify whether tweet data is useful in identifying ‘troll’ tweets by Twitter users associated with the Internet Research Agency [specification].

**2. Hypothesis**

There are, without a doubt, many reasons as to why humans make typographical errors. However, certain types of typographical errors are more common than others. It is hypothesised that:

* People make mistakes due to inaccurate typing using different input mechanisms.
* Lower proficiency in a particular language leads to a higher probability of typos.
* Words with silent pronunciation are typically misspelled more often.
* Spelling a word while reciting the wrong pronunciation in one’s head can lead to errors.

**3. Domain and Sources**

There are two datasets. The first used is entitled ‘wiki\_misspell.txt’, obtained from Wikipedia and lists 4453 common typos made by Wikipedia editors ("Lists of common misspellings", 2018).

The second is the Birkbeck spelling error corpus, a transcript of 34683 hand-written misspellings ("Birkbeck spelling error corpus ", 1946).

|  |  |
| --- | --- |
| Dataset | Features |
| Dict | 370099 words |
| wiki\_correct | 4453 corrected words |
| wiki\_misspell | 4453 typed misspellings |
| birkbeck\_correct | 34683 corrected words |
| birkbeck\_misspell | 34683 written misspellings |

Table 1: Number of words in the dataset

**3.1. Dictionary**

In this context, the dictionary represents the lexicon based on the language in test, English. It is used by the approximate matching algorithms to search for best matches.

**3.2. Wikipedia Dataset**

Two lists of identified common typographical errors made by editors on Wikipedia, one with misspelled words and the other correct words. This dataset mostly represents keyboard typing errors and assumed that all words in misspell are not the correct intended spelling.

**3.3. Birkbeck Dataset**

This dataset contains hand-written spellings from a different century. It is assumed the dictionary is more relevant to today’s vocabulary.

**3.4. Limitations**

There are several observable peculiarities with the dataset and dictionary, namely a small number of correct words do not appear in the dictionary, a number of misspelled words do appear as real words in the dictionary, absence of some proper nouns, and inconsistency between British and American spellings.

**4. Methodology**

The program uses three different approximate string-matching algorithms. An outline and short description of the different algorithms are described below.

**4.1. N-Gram Distance**

N-Gram distance leverages a different mechanism to find the distance between token and dictionary entry. An n-gram is the encompassment of the concept of the longest common subsequence (Kondrak, 2018).

The formula used to calculate the n-gram distance of a string s with set and string t with set is (Nicholson et al. 2017):

**4.2. Edit Distance**

Edit Distance is a widely used distance metric designed to determine similarity between two strings. The system uses three different variants, namely global edit distance, local edit distance, and Damerau-Levenshtein distance.

The basic idea behind global edit distance is transforming an input string into each dictionary entry based on scores associated with insert, delete, replace, and match operations (Langmead, 2018).

Local edit distance is based around the idea of finding the longest matching substring of the word.

**4.3. Neighbourhood Search**

The concept of neighbourhood search is somewhat akin to edit distance, in which the algorithm generates all variants of a string s that utilise at most k changes, be it insertions, deletions, or replacements (Nicholson et al. 2017).

However, since the dictionary is known, we can, given a parameter, search the dictionary via running the Needleman-Wunsch algorithm with Levenshtein Distance parameters, and returning words that match the score.

**5. Results**

The performance of the system was measured using two generic methods.

**5.1. All matches**

The first, takes into account all the possible matches returned by an algorithm, measured by precision and recall.

This would provide less bias against algorithms that produce many possible matches which have a higher chance of containing the correct correction.

**5.2. Best match**

The second method, given a set of matches, considers the first match as the ‘best match’. This can be used to calculate accuracy, for comparison with algorithms returning only one possible match.

However, it would seem this method is less useful, as is subject to random ordering of results. It is also plausible for an algorithm to suggest all the words in the dictionary, leading to an optimum accuracy.

As a tie-breaker for similar or equivalent scores, the algorithm which requires less execution time would be the better choice, though this would be implementation-dependent.

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Accuracy | Precision | Recall |
| NS (1) | 51.47% | 31.85% | 71.28% |
| NG (2) | 55.47% | 68.06% | 81.50% |
| LED | 31.26% | 2.27% | 62.50% |
| GED | 54.88% | 26.04% | 79.05% |
| DLD | 59.96% | 33.42% | 85.58% |

\* Accuracy is based on picking the first match in the list of matches

Table 2: Overall Comparison

**5.3. Comparing Datasets**

The N-Gram algorithm has also been run on the Birkbeck dataset, with N = 2. Results are shown below, comparing both datasets. The algorithm performs much more poorly on written misspellings.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Accuracy | Precision | Recall |
| Wikipedia | 55.47% | 68.06% | 81.50% |
| Birkbeck | 19.17% | 7.43% | 37.16% |

Table 2: Dataset Comparison

Graph 1: Overall Comparison

Graph 2: Accuracy Comparison

Graph 3: Precision Comparison

Graph 4: Recall Comparison

**6. Discussion**

The results show that, in terms of precision and recall, Damerau-Levenshtein outperforms the rest. Accuracy is rather similar, with the exception of local edit distance.

**6.1. Neighbourhood Search**

|  |  |
| --- | --- |
| Misspelled | Correct |
| 'appereance' | 'appearance' |
| 'catapillers' | 'caterpillars' |
| 'dissapears' | 'disappears' |
| 'exculsivly' | 'exclusively' |
| 'governmnet' | 'government' |
| 'harrasment' | 'harassment' |
| 'moleclues' | 'molecules' |
| 'parituclar' | 'particular' |
| 'resssurecting' | 'resurrecting' |
| 'startegic' | 'strategic' |
| 'unfortunatley' | 'unfortunately' |

Table 4: Sample of misspelled words not detected by neighbourhood of 1

While at first glance, even though the misspelled words look almost identical to the correct word, they went undetected due to being just out of the boundary.

Increasing the neighbourhood to higher values may detect the error, but also produces many more results, hence lowering the precision.

Neighbourhood search has the advantage that it is not affected by the fact that some misspelled tokens are present in the dictionary, given that a neighbourhood of > 0 is specified.

This method has one major pitfall, in which it might not return any matches, especially if the dictionary is small, or if words are exceptionally longer than the rest.

**6.1. N-Grams**

N-grams however, is more consistent, as it does not rely on having a word present in the dictionary. It is less sensitive to how strings are ordered, as matches can be anywhere, but is more sensitive to longer substrings.

In terms of typographical errors, n-grams vaguely demonstrates that people do think out loud while typing on the keyboard as each n-gram somewhat mimics the sound produced while ‘breaking-down’ words into smaller units of pronunciation.

**6.3. Edit Distance**

Local edit distance performs the most poorly here, mainly due to the fact that the dictionary is relatively large, as well as having a relatively small alphabet size of 26. The bigger the number of words, the more likely that longer substrings are to match a particular string. This causes the algorithm to return a much larger hence the low precision score.

Based off the Needleman Wunsch algorithm, both Levenshtein distance and Damerau-Levenshtein distance perform well on the dataset. Damerau-Levenshtein is an improvement over Levenshtein, whereby it also allows for transposition of two adjacent characters (Setiadi, 2018).

This would consider misspellings that could be corrected with at most one edit operation, hence the higher overall score compared to Levenshtein.

Mispelled: acquiantence Correct: acquaintance

|  |  |
| --- | --- |
| Algorithm | Matches Returned |
| Levenshtein | acquaintance,  acquiescence,  acquiesence,  acquittance |
| Damerau-Levenshtein | acquaintance |

Mispelled: dimensional Correct: dimensional

|  |  |
| --- | --- |
| Algorithm | Matches Returned |
| Levenshtein | digestional,  dimensional |
| Damerau-Levenshtein | dimensional |

Mispelled: directly Correct: directly

|  |  |
| --- | --- |
| Algorithm | Matches Returned |
| Levenshtein | dejectly,  directly,  erectly,  oriently,  priestly |
| Damerau-Levenshtein | directly |

Tables 5: Sample of words Damerau-Levenshtein is able to correct, but not Levenshtein

**7. Improvements**

There are many ways to improve on the methodology used in this project. One such is incorporating phonetic algorithms, such as Editex, Soundex, or Metaphone to consider typographical errors arising from words which sound similar, especially with words pronounced with silent characters.

Incorporation of an ensemble method such as bagging (Hauskrecht, 2018), which would, given all the results from the different algorithms, choose the best match based on majority vote. This is akin to a Random Forest classifier, which has the potential to be improved via boosting.

Another way would be to implement a ‘keyboard-distance’ algorithm that would determine the relative distance of each key to every other key, and base word distances on this. This might improve the typographical suggestions but may be tied to a specific keyboard layout such as QWERTY.

Implementation of a neural network model which would be able to, via supervised learning, continuously update its knowledge about typographical errors with each new corpus processed.

**8. Conclusions**

In all, while this analysis may be far from optimal, it suggests that there are many different approaches and methods to determine the types and causes of typographical errors.

However, the algorithms chosen might not be close enough to be relevant in justifying the hypotheses stated initially. More advanced methods are required to further test the hypotheses, strengthening the consensus that spelling correction is still considered a knowledge task, even with the many advancements in AI and machine learning,

**References**

Linvill, Darren and Patrick Warren (2018) Troll factories: The Internet Research Agency and state-sponsored agenda building (working paper). Clemson University.

Roeder, Oliver (2018) Why we’re sharing 3 million Russian troll tweets. In FiveThirtyEight. 31 Jul 2018, https://fivethirtyeight.com/features/why-were-sharing-3million-russian-troll-tweets/.

Kondrak, G. (2018). N-gram similarity and distance. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.9369&rep=rep1&type=pdf

Zobel, Justin and Philip Dart. (1996). Phonetic String Matching: Lessons from Information Retrieval. In Proceedings of the Eighteenth International ACM SIGIR Conference on Research and Development in Information Retrieval. Z¨urich, Switzerland. pp. 166–173.

Langmead, B. (2018). Dynamic programming and edit distance [Ebook] (p. 10). Johns Hopkins Whiting School of Engineering: Department of Computer Science. Retrieved from http://www.cs.jhu.edu/~langmea/resources/lecture\_notes/dp\_and\_edit\_dist.pdf

Hauskrecht, M. (2018). CS 2750 Machine Learning Lecture 23: Ensemble methods. Bagging and Boosting [Ebook]. University of Pittsburgh. Retrieved from https://people.cs.pitt.edu/~milos/courses/cs2750-Spring04/lectures/class23.pdf

Setiadi, I. (2018). Damerau-Levenshtein Algorithm and Bayes Theorem for Spell Checker Optimization. Retrieved from http://file:///C:/Users/Xenzaki/Downloads/Damerau-LevenshteinAlgorithmandBayesTheoremforSpellCheckerOptimization.pdf