A Blend Word Detection Algorithm Based on Approximate String Matching

Jinzhe Shan

1 Introduction

As the flourish of social media like Facebook and Twitter, languages are used with more and more creativity. Coining new words is one of typical innovations, which mixes two existing words with a new blend word such as Brexit (mixing Britain and exit) and Brunch (mixing breakfast and lunch). However, blend-word detection is a challenging problem that has been studied by many researchers.

This report aims to propose an appropriate method of blend-word detection by using approximate string matching algorithms, implement a practical system to detect blend words candidates from Twitter data and evaluate the performance of my proposed detection algorithm based on the evaluation metrics including precision and recall. It also illustrates knowledge I have gained from handling this problem

Twitter dataset (Eisenstein et al., 2010) and blend datasets (Deri and Knight, 2015; Das and Ghosh, 2017; Cook and Stevenson, 2010).

2 Dataset

The Twitter dataset is from the data set of research by Jacob Eisenstein, Brendan OConnor, Noah A. Smith, and Eric P. Xing (Eisenstein et al., 2010). The list of blend words was compiled using resources presented in publications by Deri A , Das K and Cook P (Deri and Knight, 2015; Das and Ghosh, 2017; Cook and Stevenson, 2010). Thanks to the organized work by subject coordinators and tutors in COMP90049 Knowledge Technologies at Unimleb. All dataset files and their meanings related to this project are shown as Table 1.

The main contributions of this project are to detect whether candidate words from candidates.txt are blend word or not depending on analyzing the approximate string matching with dictionary from dict.txt. Blends from blends.txt is used to evaluate the performance of my detection algorithm.

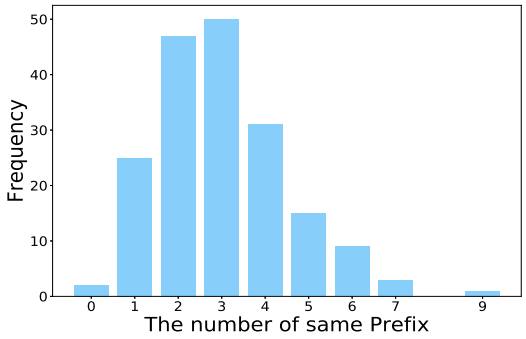

Figure 1: Prefix Count

3 Patterns Mining and Hypotheses

3.1 Mining patterns of blend words

In order to detect blend words, the first step is to observe the features and patterns of them. By direct observation and statistical analysis, three typical features of blend words are found.

3.1.1 Prefix and suffix matching

By observing blends, it is obvious to find that most blend word has the same prefix and suffix with its source words. For instance, brunch is combined by prefix br from breakfast and suffix unch from lunch and minja is combined by prefix mi from midget and suffix nja from ninja. Hence, I used Python to get statistical data onto the number of prefix and suffix letters blend words have the same with their source words, shown as Fig 1 and Fig 2. Based on the frequency figures of the number of prefix and suffix, it is noticeable that most blend words have less than six same prefix and suffix , which is a significant tool for following implementation on filter appropriate source words.

3.1.2 Global Edit Distance

Additionally, Global Edit Distance (GED) is thought of naturally when it comes to string matching. I analyze the GED between all blends and their source words combination

dataset file	meaning
blends.txt	A list of true word blends
candidates.txt	A list of 16925 tokens present in wordforms.txt but not in the dict.
dict.txt	A list of approximately 370K English entries
tweets.txt	A list of the text from 62345 tweets
wordforms.txt	This is an alphabetically-sorted list of 31763 unique tokens present within the tweets

Table 1: The data set files

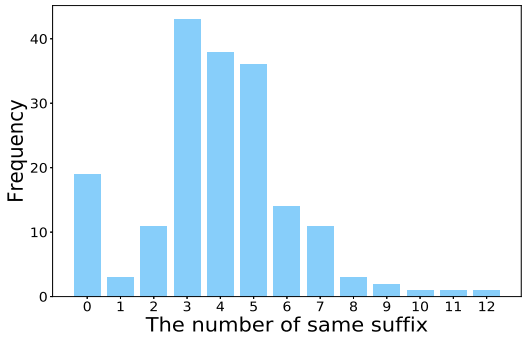

Figure 2: Suffix Count

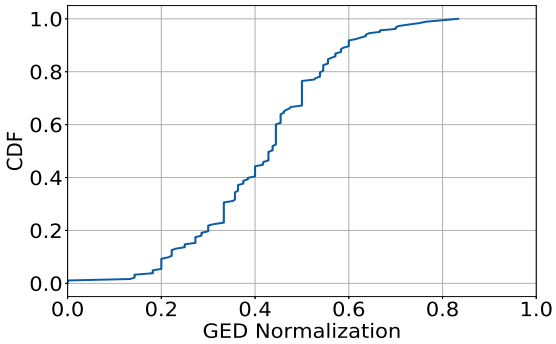

Figure 4: CDF Diagram of Global Edit Distance Normalization

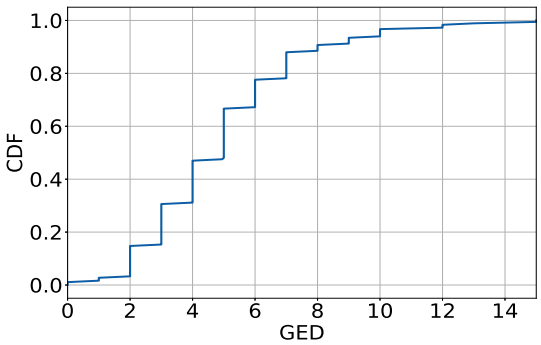

Figure 3: CDF Diagram of Global Edit Distance

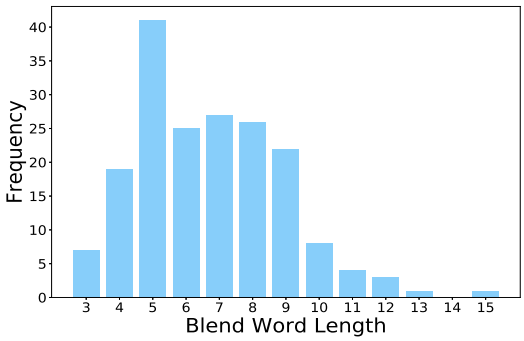

Figure 5: Count different length of blends

string (first word splices second word). In consideration of the influence on different length of string, a normalization is implemented by dividing the length of combination string as GED Normalization.

In order to get the distribution information about global edit distance and its normalization, two Cumulative Distribution Function (CDF) charts to illustrate the relationship between GED, GED Normalization and their distribution curves in Fig 3 and Fig 4. Based on CDF charts, the blend word detection parameters can be defined logically as GED Normalization less than 0.6, because such situation can meet more than 93% blend words and filter out candidates that are not blend words effectively.

3.1.3 Length of blend words

Finally, the length of blend words is a plain but reasonable feature. It is less probability for people to coin a new blend word very short or extremely long. Thus, I do some statistical analysis shown in Fig 5.

3.2 Hypotheses

According to the above analysis, there are three corresponding hypotheses of blend words as follows.

1. The longest prefix or suffix is six;
2. The Normalized GED is less than;
3. The length is larger than two but smaller than 13.

4 Methodology

4.1 Background Knowledge

The following two approximate string-matching algorithms are used in this blend word detection algorithm, which are Jaro-winker similarity and Global Edit Distance (Levenshtein distance).

4.1.1 Jaro-winker similarity

As the above analysis, the most obvious feature of blend words is they have the same prefixes and suffixes with their source words. Hence, I selected Jaro-winker similarity as a useful tool for matching prefixes / suffixes and detecting blend words. Particularly, I designed a program which is not only calculate the jaro-winker similarity with prefixes matching but with suffixes matching based on the work by Toastriven toastdriven (2012)

4.1.2 Global Edit Distance

According to the above hypotheses, Global Edit Distance is a distinguishing feature of blend words to other candidates. Therefore, this project calculates the Global Edit Distance between each candidates and their all combined strings based on splitting two source words and divides the length of combined string to get the normalization GED as the judgement indicator for blend words detection algorithm.

4.2 Blend Word Detection Algorithm

The blend word detection algorithm was proposed as shown in Algorithm 1.

Algorithm 1 The game structure

Require: *candidates.txt*;

```
for candidate in candidates do
    Get prefix words using prefix tree
    Get suffix words using suffix tree
    Calculate Jaro-Winker similarity between candidate and each prefix word
    Calculate Suffix-Jaro-Winker similarity between candidate and each suffix word
    Get top 10 most similar prefix words as first source-word list
    Get top 10 most similar suffix words as second source-word list
    Combine first source-word list and second source-word list to 100 combined strings
    for each combined string in 100 combined strings do
        Calculate the Normalization Global Edit Distance between the combined string and candidate
        if Normalization Global Edit Distance < 0.6 then
            Add the candidate to detectBlends
        end if
    end for
end for
Calculate precision, recall
Return detectBleds.txt
```

5 Implementation

5.1 Distance algorithms and Tools Used

Under consideration about the volume of dictionary and candidates, I used trie (prefix tree) for search possible words with same prefix and suffix from dictionary to improve computational efficiency in this project implementation.

5.2 Implementation Details

This project implementation used Python including numpy, heapq, pytrie, JaroWinkerSim and LevenshteinSim libraries.

There are some creative and significant work for reproduction as following:

1. Realized a prefix dict tree and a suffix dict tree for search prefix and suffix string quickly
2. Realized a reverse Jaro-Winker similarity to match suffix
3. Increased the parameters of Jaro-Winker similarity in order to give prefix or suffix matched word more weight.

5.3 Unsuitable Approaches Review and Handling

In the first version of my program, I have not considered about the algorithm complexity, because I test using small sample data set with 500 candidates and a dictionary including 1000 words. However, when I used it to execute the whole dataset, I did not get results over 12 hours. By analysis of time package, I calculate the first version of my blend-word detection program need more than 700 hours in my laptop. The algorithm complexity of first version was not $O(n^2)$ as I calculated because I ignored the complexity Global Edit Distance algorithm is also $O(n^2)$. Finally, the algorithm complexity of my first version was $O(n^3)$, which was not a good design for the project with large volume dataset.

5.4 Strategies for performance improvement.

1. remove candidates which are not possible as blend words
2. add some condition judgement before distance extremely short or long combined strings are dropped
3. using prefix tree to get prefix matching words rather than calculate each candidate with all words in dictionary
4. using heap structure to get top 10 similar prefix and suffix source words rather than using sort algorithm to sort all items

6 Evaluation

In this report, the following terms are used to evaluate the performance of detection algorithm. Precision indicates that fraction of correct responses among attempted responses. In this detection algorithm, precision means how many really blend words in the whole set the algorithm judges as blend words. Recall shows the proportion of correct responses among true items (blends). Particularly, recall means how many really blend words are detected by the detection algorithm. According to the result of experiment, the precision and recall are evaluated as Table 2.

Evaluation	True Detection	Total Detection	Blends	Result
Precision	122	8253		1.48%
Recall	122		151	80.79%

Table 2: Evaluation of Precision and Recall

7 Related Works

The main research paper I have studied is Automagically Inferring the Source Words of Lexical Blends from Paul Cook and Suzanne Stevenson Cook and Stevenson (2007). That paper propose four features in linguistic and cognitive aspects to define and detect blend words. Firstly, Griess work (2004) found that the second source word tends to be longer than the first, so he defined a parameter to describe this feature called Contribution and Length (len(w2) / len(w1) + len(w2)). In addition, Lehrers research (2006) had insight that the more frequency word has the more possibility as the source of blend words. Next, approximate phonetic similarity is one of important linguistic features to distinguish truly blend words. Finally, part of speech is another powerful tool to judge and narrow the range of source word pairs thereby improving the detecting precision.

8 Knowledge Gain

Through this project, there are some knowledge I have gained. At part 4 of this report, I mined three main patterns of blend words and proposed three hypotheses about blend word detection. It is easy to understand for some readers who are not familiar with detection of blend words by those charts and diagrams. In addition, I broaden horizons about this problem by reading related research papers. Except the approximate string-matching algorithm, there are many other powerful tools and features we can apply for handling the problem of blend word detection.

9 Conclusions

This report studied the problem of blend word detection and proposed a solution based on approximate string matching algorithm. By direct observation and statistical analysis, three features and hypotheses be put forward as preconditions and judgement parameters for the design of my detection algorithm. However, the evaluation result illustrated that the low precision of my proposed detection algorithm was not ideal and powerful to detect blend words. After the relevant literature of other researchers work on the field of blend word detection, it is noticeable that approximate string matching algorithm is only one of tools to handle this problem. I need to study more about other powerful algorithms and tools such as linguistic analysis and machine learning algorithms.

References

P. Cook and S. Stevenson. Automagically inferring the source words of lexical blends. In *Proceedings of the Tenth Conference of the Pacific Association for Computational Linguistics (PACLING-2007)*, pages 289–297, 2007.

P. Cook and S. Stevenson. Automatically identifying the source words of lexical blends in English. *Computational Linguistics*, 36(1): 129–149, 2010.

K. Das and S. Ghosh. Neuramanteau: A neural network ensemble model for lexical blends. In *Proceedings of the The 8th International Joint Conference on Natural Language Processing*, pages 576–583. Taipei, Taiwan, 2017.

A. Deri and K. Knight. How to make a frenemy: Multitape FSTs for portmanteau generation. In *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL*, pages 206–210. Denver, USA, 2015.

J. Eisenstein, B. O'Connor, N. A. Smith, and E. P. Xing. A latent variable model for geographic lexical variation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP 2010)*, pages 1277–1287. Cambridge, USA, 2010.

toastdriven. pylev. `https://github.com/toastdriven/pylev`, 2012.