Computational Detection of English Lexical Blends

Anonymous

1 Introduction

Twitter – a social microblogging site – has millions of users registered on it, and the numbers keep getting added. Consequently, there comes a large amount of data, mostly in the form of text, images, numbers and much more. Using language informally on social media has become a common trend these days, thereby giving birth to many new terms.

This report is particularly concerned with the common phenomenon of blending words, or more specifically, *lexical blending*. Lexical Blending refers to the coinage of new terms by combining parts of other words, and such terms are called *lexical blends*. For isntance, *spork* is a blend word formed by combining *spoon* and *fork*. In this paper, we are going to discuss method(s) to detect/identify occurences of lexical blends among a pre-processed list of tokens that has been extracted from Twitter data set.

It is important to note that this paper gives a method which might be effective in detecting blend words; but it doesn't by any means solve the general problem. We are going to dicuss about the shortcomings and how algorithms can be improved, in the latter part of the report.

2 Dataset

Dataset	Frequency
dict	370,059
tweets	62,345
wordforms	31,763
candidates	16,925
Blends(along with a pair of source words)	183

The table shows the datasets that have been provided to us. The tweets have been preprocessed to give unique tokens(wordforms) and candidates are obtained by filtering the novel words out from the wordforms. We further filter the candidates to make a list of words which could potentially be blends. A reference set of dictionary words has been provided for detecting the blend words, and lastly, a list of true blend words is given which would be used as a test set to validate our results.

3 Relevant Literature

Not much has been published in this direction though, but a paper named "Using Social Media to find English Lexical Blends" provided me an approach towards my research. It is written by Paul Cook, who's a research professor at the Department of Computer Science in the University of Melbourne. It was published in 2011, and was preceded by another landmark paper – "Automatically Identifying the Source Words of Lexical Blends in English" - which was also written by Paul Cook, along with his colleague, Suzanne Stevenson. Both of these research papers give us an overview of how Prof. Cook has defined some effective methods in not only predicting the blends(which are not part of Dictionary, but are neologisms), but also proposed a statistical model for inferring the component words of lexical blends.

Following the same principles, I shall go ahead with my research.

4 Approximate Matching Techniques

Here, I have used two well known edit distance methods, namely Jaro-Winkler Distance and Levenshtein Distance. Jaro-Winkler method measures the similarity between two strings, and represents it as a real number between 0 and 1. This method is said to be best suited for short strings such as person names, and thus here I shall be using it as my prime algorithmic function for prefix matching.

Another method that we shall be using will be the Levenshtein Distance, which by definition is the minimum number of single-character edits(I,D,R) used to change one string into another. The lesser the distance, the more similar the two strings are said to be. We'll be using these two string matching techniques to carry out our process, and later infer the results.

5 Methodology

For any Machine Learning task, we always have a training set, upon which we perform our operations and a test set, with which we validate our results obtained from the training set. Here, we will be defining a method to detect occurences of word blends from a list of candidate words, obtained by pre-processing the tweets. As mentioned earlier, the candidate.txt file already had the filtered words from the wordforms file which were not included in the dictionary.

Further cleansing was performed on the candidates, by removing overemphasized[1] words like "okayyyy" and "yummmm". Using regex patterns, more filtering was done on the words to get rid of such terms which couldn't potentially be blends. Finally, we reduced our candidates list to approximately 12,000 words, amongst which we were to identify the blends.

Another list of strings just containing the true blend words is created (without the 2 source words), and stored in a list named *target.*

A training set was created by randomly selecting 20 terms that were common in candidates and true blends.

```
-----------------------------------------------------
train_set = random.sample(target,20)
-------------------------------------
```

Now, from the training set of blends, we figure out the length of the prefix being used (for each blend) from the first source word. Consequently, we fix a general prefix length[2] for our algorithm, which comes out to be 3. The rest of the word length ($l - 3$) is then considered to be the suffix from the second source word. So, we have two lists, one of which has the prefixes and the other has the suffixes of the words from training set.

For every set of prefix and corresponding suffix, we filter out a list of words from the dictionary with respect to respective blend.

We then further filter out words in that list by restricting them with threshold values of Jaro – Winkler[3] similarity and Levenshtein Distance.

A run of experiments was performed on a few true blends to get average threshold values :

Method	Threshold
Jaro - Winkler	>= 0.6
Levenshtein	<= 5

[1] Quite common in language used on social media, following an informal style.

[2] A general prefix length is set by taking out the mean of 20 prefix lengths from the training set.

[3] Effective in prefix matching.

```
-----------------------------------------------------
if
(round(distance.get_jaro_distance(train_
set[0], list[i], 1)) >= 0.7 and
stringdist.levenshtein(train_set[0],
list[i])) <= 4
-----------------------------------------------------
```

This filtered list is our final list, and if it contains the two source words for our blend, then the blend is said to be predicted correctly, else the word isn't identified as a blend.

6 Evaluation

Here, we give you an analysis of our result. From our training set, 5 words out of 20 were predicted to be true blends. According to our results on this training set, we received an accuracy of 25%.

7 Conclusion

Working on the training set, we conclude that this approach might be useful when predicting new lexical blends from a list of words, and would therefore, using string matching techniques, give a combination of source words as well. And then as defined by Prof Cook, we can use statistical methods to infer the best pair of source words. But this method would be effective only for a subset of words, where the blend word is solely formed by combining the prefix and suffix of two source words respectively. There are plethora of blends which violate this phenomenon, for example *entreporneur(entrepreneur + porn).* Also, keeping the threshold values of Jaro-Winkler Similarity and Levenshtein Distance fixed is not a good idea for every string. Rather, depending on the length of each candidate word, we should have an algorithm that sets the respective threshold values.

Lastly, a phonetic string match algorithm would be effective if included, as a blend is likely to sound the same like either of the two source words. Hence, our methods might predict the blends for a restricted set of candidates only. Perhaps, further research on this could provide with more effective algorithms that would inculcate the shortcomings in this procedure.

References

[1] Cook, P., & Stevenson, S. (2010). "*Automatically Identifying the Source Words of Lexical Blends in english"*. (Computational Linguistics: Volume 36, Number 1)

[2] Cook, P. (2011) "*Using Social Media to find English Lexical Blends"*.

[3] Luoz. Hoyang (2018).
https: // github.com/louzhouyang/python-string-similarity#jaro-winkler

[4] Ratte, Jean-Bernard (2016)
https: // pypi.org/project/pyjarowinkler/

[5] Mohit, Mayank (2017)
http: //itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227

[6] Jacob Eisenstein, Brendan O'Connor, Noah A. Smith and Eric P. Xing. 2010. 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

[7] Deri, A. and Knight, K. (2015) 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*

[8] Das, K. and Ghosh, S. (2017) Neuromanteau: A Neural Network Ensemble Model for Lexical Blends. In *Proceedings of the 8th International Joint Conference on Natural Language Processing,* pages 576-583