Identifying Lexical Blends in Social Media Text

Anonymous

1 Introduction

A lot of work has been done in the field of natural text processing over the years. Most of this work has been around tackling problems such as approximate string matching and spelling correction. Very little work has been done in the lexical blending space. This is mainly because of the complexity of the task and also the availability of data sets and gold standards to evaluate the efficiency of the same.

A lexical blend in the English language is formed by the combination of a prefix and suffix taken from two different source words respectively (Cook and Stevenson, 2010). This report aims to build a system that recognises if a given word is a lexical blend. This can also be looked at like a classification problem of deciding if a given word is a blend or not. This report also analyses the performance of Jaro Winkler distance applied to implement such a Lexical Blends identification system with the aid of Levenshtein distance to narrow down candidates.

Most of these blend words are not present in any dictionary. With their increasing usage especially in social media, identifying these blend words will help many natural language processing tasks. In this project, the Twitter dataset (Eisenstein et al., 2010) set is analysed to identify the blend words. Twitter is a microblogging website where users are allowed to post their opinion. Incidentally, these posts contain massive amounts of blend words. For comparison purposes, a sample blend words dataset coined by Deri and Knight (2015); Das and Ghosh (2017) and Cook and Stevenson (2010) is taken.

2 Methodology

In this implementation, in the first stage the candidates are filtered using strategies that legitimate words in English would follow. After this, the Jaro Winkler distance is calculated with each candidate from the dictionary file that has the same prefix as the candidate. And the same value is calculated for the suffix of the candidate as well. A threshold value is inferred from the blends file upon computing the distances of each source word with the corresponding blend. This distance is compared to decide if a word is a blend or not. An overview of this is given in Figure 1.

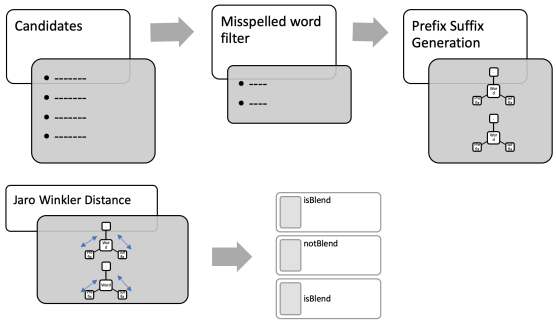

Figure 1: Blends Identification system overview

2.1 Shortlisting Candidates

The candidates of blend words are those that are not present in any dictionary. This helps us in shortlisting the candidates. We further try to narrow down the candidate list as much as possible. This would not only reduce the time to process but also improve the accuracy of the system by reducing the amount of false positives. A number of heuristics are used for this.

2.1.1 Orthographic Features

The twitter data set has content that is mostly written informally which is subject to grammatical errors and spelling mistakes. Very often users would intentionally spell words incorrectly to stress on expression by using repeated letters. This can be used as a heuristic to shortlist blend words. The main intuition behind this is that these words would not fall under the blend words category. Hence, we can filter these out to cut down the candidate list. A similar heuristic

has been used by Kaufmann and Kalita (2010) in their attempt to normalize Twitter data and also by Cook and Stevenson (2010) for a similar task. For example, words like 'zzooommmmm' with three or more consecutive repeating characters or 'aahhahhaha' where a single letter appears more than four times is also pruned out of the candidate list. This can be easily done with the help of regular expressions.

2.1.2 Names Corpus Filtering

This is an additional heuristic that helps us further narrow down the true candidates for the blend words. This arises from the intuition that a person name can never be a blend word. For this purpose the Names Corpus from nltk,[1] was used (Loper and Bird, 2002). This corpus consists of eight thousand male and female names contructed by Kantrowitz (1994). With the help of this about four hundred candidates were identified as names and were pruned out of the candidates list.

2.1.3 Misspelled word filtering Using Edit Distance

Another heuristic is that misspelled words cannot be word blends. For this reason, a spell checker [2] is used to identify misspelled candidate. This uses the Levenshtein Distance algorithm to find permutations within an edit distance of two from the original word (Okuda et al., 1976). For this implementation the edit distance was set to one. This would help us further eliminate misspelled words from candidate word blends. This strategy helped us narrow down the candidate list which was around eleven thousand after the previous filtering techniques to a value of just five thousand. This is a huge reduction and helped achieve better accuracy.

2.2 Jaro Winkler Distance computation

Jaro Winkler distance is a very efficient string similarity matching measurement (Wang et al., 2017). In this measurement, more preference is given to the prefix matches. It gives us a normalized value between zero and one. In case of blend words, comparison is made easily with these normalized values. The characteristic of blend word is such that prefix will be matched with a dictionary word. Hence, giving more importance to this nature of a blend word may help us in identifying blends. This implementation is not biased towards suffix matches.

In this implementation, the Jaro winkler distance is used to identify potential blends. This is done by approximating a threshold value from the blend words data set. The average value of the Jaro winkler distance between the blend word and the prefixes in blend words data set is taken as the prefix threshold. Similarly, the suffix average value is also computed and the suffix threshold is obtained. Table 1 shows the average distances obtained with the blends file considered. It can be observed that both the values are due to the presence of many matching characters present in the blend word.

2.3 Normalized edit distance

Normalized Levenshtein distance was also used in the same manner Jaro winkler is used in testing this system. Similar results were observed in both cases with minute variations in accuracy whereas the F1 score saw no change. This report focuses on the Jaro Winkler implementation.

String in blends	Avg. Jaro Winkler Distance
Prefix	0.80
Suffix	0.74

Table 1: Average distance between Blend words dataset and prefix/ suffix

Now, for each candidate word, the prefix and suffix are generated of increasing length iteratively. Then, the Jaro Winkler distance is computed between each prefix and the candidate word followed by suffix and candidate word. If this is more than the threshold values, this word is marked as a blend word.

3 Evaluation

For evaluation of this system, the F1 score and Accuracy are the best metric to show the performance. This is mainly because this is a binary classification problem where there are fewer positive cases. The table below has the consolidated report of the metrics used to evaluate this system (see Table 2). The confusion matrix in 2 depicts the number of items in each case that is identified by the implementation against the actual values.

3.1 Accuracy

Accuracy is defined as the percentage of total words that are classified correctly. This is the

[1] Natural Language Took Kit in Python

[2] pyspellchecker library in python was used for this

	precision	recall	F1	supp
Yes Blend	0.99	0.72	0.84	16537
Not Blend	0.02	0.58	0.04	147
accuracy			0.72	16684
macro avg	0.51	0.65	0.44	16684
weighted avg	0.99	0.72	0.83	16684

Table 2: Classification Report of the proposed system

ratio of correctly identified blend words along with correctly identified non blend words to the total words attempted.

$$\frac{TruePositive + TrueNegative}{AllAttempted}$$

. This system has showed an Accuracy of 72%

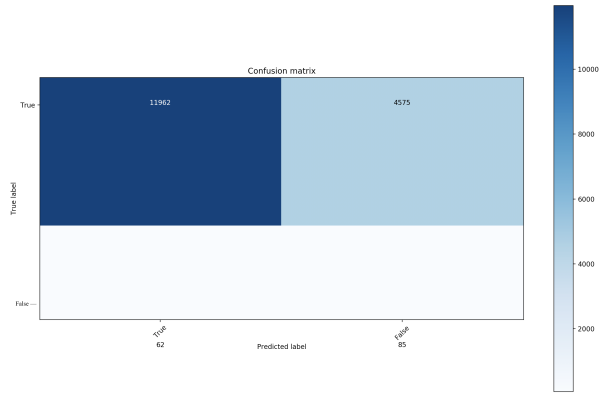

Figure 2: Confusion matrix of Blends Identification

3.2 F1 Measure

In evaluation of binary classification the F1 measure is very popular. The F1 score lies between zero and one. The system is said to be best when it achieves an F1 score of 1 and is said to be bad one if it is nearing zero. The F1 score is the most appropriate way of evaluating the system as it does not give much importance to the true negative classes which in our case are the words correctly classified as non blend words. The proposed blends identification system has F1 score of 0.0354. The F1 measure is the harmonic mean of Precision and Recall which are explained in the subsequent sections.

$$\frac{2 * Precision * Recall}{(Precision + Recall)}$$

We can also see from the data (see Table 1) that for correctly classifying a word as a blend the F1 score is 0.84 as a result of high precision and recall. Whereas, when identifying a word correctly as a non blend, F1 score of just 0.04 is obtained.

3.2.1 Precision

Precision is the ratio of items correctly classified as true out of the total items that are classified as true. This is an important metric in our case. The overall precision of the system is 0.0182. This is low because there were many words that were wrongly classified as a blend word. It can be observed from Table 1 that among all the labels that were classified as blends, almost all of the true blend words that were present in the candidate set were identified correctly. When it comes to the non blend words, the value of 0.02 tell us that the system is very bad when it comes to predicting non blend words.

3.2.2 Recall

Recall is defined as the number of items classified as positive among the true positive values. In this context, it is the ratio of words that are classified as blends among the true blend words. The overall recall of the system is 0.578. This is because in the shortlisting stage, many words that were true blends got eliminated. This led to a decrease in recall. However, if we implement the same method without the shortlisting of candidates and only by maintaining the threshold distance parameter, the recall observed was almost 0.99. Hence, recall although an important metric, it cannot depict the true performance of the system. From Table 2 we can see above average performance in terms of recall in both identifying blends and non blends.

4 Improvements

There are many areas of improvement in this system. An F1 score of 0.035 is not a suitable score for a functioning word blends identifier. However, given the complexity of this task, even a system with an F1 score of around 0.5 can be regarded as a good one. The shortcomings of this method and their improvement techniques are:

1. In this implementation, there are flaws in the filtering of data which gives rise to reduced recall. When the misspelled words are eliminated, this very often removes blend words that seem like a wrongly spelled word. This is because when analysing misspelled words using

an edit distance of one, this results in pruning of words like 'funemployment' and 'bromance' from the candidate list. Many true blend words that resemble a misspelled word of a unit edit distance are removed. At this cost, the precision increases, the recall lowers. However, the F1 measure increased two folds from 0.017 to 0.035.

2. The blend words are extracted based on a threshold value that is observed in a small data set. This is not an efficient way of extracting as this might result in dropping of words that are true blends but did not match up to the threshold value. This threshold has to be fine tuned to achieve a desirable accuracy.

3. Another area of improvement is by considering the context of the blend word. The position of a word in a sentence could give us more knowledge and capturing these patterns in our implementation will improve the accuracy of classification. Contextual features can also be designed to take into account the previous word. This heuristic is implemented in the work done by Cook and Stevenson (2010) where a regular expression that consisted of frequent terms that occurred before a blend word was used.

5 Conclusions

Overall, this system achieves an accuracy of 72% and an F1 score of 0.035 in identifying blend words. It uses the Jaro winkler Similarity distance and also the edit distance to achieve this result. This report also establishes that identifying lexical blends is a complex task that cannot be achieved by only considering approximate string matches. Additional heuristics and context have to be considered to improve accuracy.

References

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.

M. Kantrowitz. *Name Corpus: List of Male, Female, and Pet names*. NLTK, 1994.

M. Kaufmann and J. Kalita. Syntactic normalization of Twitter messages. In *International conference on natural language processing, Kharagpur, India*, 2010.

E. Loper and S. Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1118108.1118117. URL `https://doi.org/10.3115/1118108.1118117`.

T. Okuda, E. Tanaka, and T. Kasai. A method for the correction of garbled words based on the levenshtein metric. *IEEE Transactions on Computers*, C-25(2):172–178, Feb 1976. doi: 10.1109/TC.1976.5009232.

Y. Wang, J. Qin, and W. Wang. Efficient approximate entity matching using jaro-winkler distance. *Lecture Notes in Computer Science*, pages 231–239, 2017. doi: 10.1007/978-3-319-68783-$4_1$6.