COMP90049 Project 1 Report: Word Blending in Twitter

Anonymised

1 Introduction

Blending words are words such as *brunch*(*breakfast* + *lunch*), *spork*(*fork* + *spoon*) and *Brexit*(*Britain* + *exit*) that are formed by combining a prefix of one word with a suffix of another. The aim of this project is using approximate matching methods to interpret the lexical words from the twitter candidate files. The initial method will rely on n-gram / jaro-winkler implementation, and, through knowledge gained from analysis of these implementations, an optimized method will replace them.

2 Dataset

The data we used to predict lexical blending is based on the Twitter[9]. Tweets messages posted to Twitter tend to be of a rather informal register; Twitter is therefore an excellent source of data to search for new blends because it is expected to contain many expressions that would be unlikely to occur in more formal registers.[5] Also, we have some true blend words data[8] that will help us to evaluate the precision of our result.[7]

3 Evaluation Metrics

To evaluate the performance of our system, the following terms will be used to evaluate each system:

1. Precision: It is the fraction of relevant instances among the retrieved instances.
   Here is the equation that used in the final precision evaluation:
   $$\frac{numberofreturnedrelevantresults}{numberofreturnedresults} = \frac{\mathrm{tp}}{\mathrm{tp}+\mathrm{fp}}$$
2. Recall: It is the fraction of relevant instances that have been retrieved over the total amount of relevant instances.
   Here is the equation that used in the final recall evaluation:
   $$\frac{numberofreturnedrelevantresults}{totalnumberofrelevantresults} = \frac{\mathrm{tp}}{\mathrm{tp}+\mathrm{fn}}$$

4 Heuristics

Before using different kinds of methods to find out the lexical words from the tweets, we first restrict the blends identified according to several heuristics related to the candidate source words.[6]

Here are some restrictions to the candidate source words:

1. Sequence of repeated characters. In Twitter, words are commonly lengthed, often for emphasis, as in the following example: “aaaawww”, “afternoooon”, “agreeeeeeddddrt”, etc. Such lengthened form cannot be blending words. To avoid wasting time comparing the distance or similarity between this kind of words and the dictionary words, we require that all candidate words couldn’t have a sequence of more than two repeated characters.

2. Short length of candidate words. In our previous knowledge about the lexical word, it most likely combining a prefix of one word with a suffix of another. So that means short length of candidate words in tweets may not be the ideal words. We then restrict the length of the candidate words must be more than 4.

5 Methodology

We search for each candidate word 'a', and search each of the dictionary word that has the same prefix(length of prefix >= 2)recorded as 'b'. And then we use n-gram method to compare whether a and 'b' are similar. After that, we search the dictionary word that has the same suffix, recorded as 'c', and using the same method to compare whether 'a' and 'c' are similar. If both 'a' and 'b', 'a' and 'c' are similar, we just added them to the final result.

5.1 N-Gram Similarity

After using our knowledge to clean our candidate words, the next step is to find a useful method to compare the candidate words with dictionary words. The initial method is using N-Gram string similarity. We choose 2-gram and 3-gram to analysis the candidate words. And the ngram library from Graham Poulter(2017) [1].We use the following function to test our method:

$$NGram.compare(w1, w2, N = n)$$

As we don't know the ideal value of the similarity that could predict one of the words would be the blend of another, so we listed several values (0.3, 0.4, 0.5, 0.6) and compared them separately.

6 Result

Using n-gram similarity functionthe results are showed in the following table:

Table 1: Results of N-Gram analysis the candidate file

Method	Similarity Value	Precision	Recall
2-Gram	0.30	1.28%	10.30%
2-Gram	0.40	1.30%	11.1%
2-Gram	0.50	1.35%	11.40%
2-Gram	0.60	1.10%	9.80%
3-Gram	0.30	1.39%	11.30%
3-Gram	0.40	1.23%	10.60%
3-Gram	0.50	1%	10.10%
3-Gram	0.60	0%	0%

7 Analysis

From the above results shown in the table and returned potential blend words, we could gain some useful knowledge that helps us to optimise our system:

1. For 2-Gram method, when choosing 0.5 as the similarity value, and for 3-Gram method, when choosing 0.3 as the similarity value, we could get the highest precision and recall result comparing to other similarity values using in the same method. So when following the equation:

   $$NGram.compare(w1, w2, N = 2) > 0.5$$

   or

   $$NGram.compare(w1, w2, N = 3) > 0.3$$

   w1(candidate) will more likely be considered as the lexical word.

2. The precision results are too low because the system return about 1500 words from the candidate files and only 15 of them are true blends. When we further looking at the returning words, it will easily find out some restrictions we missing in our initial program implementation.

   In "candidate.txt" file, there have many words that were misspelled words like "afternnon", "againn", "already", etc. These words have the same prefix of the right spelled words from the dictionary file and these words also have high similarities so it will always make our system return this misspelled words to the results, which make lots of useless words influence our precision result.

   Furthermore, when we calculate the prefix of two words, we just restrict that these two words will be granted if they have more than two same characters from the beginning. That will make some words like "barioth" and "baraybee", although they have same "prefix": "ba", but that "prefix" definitely will not be accepted by our system.

   Not only our prefix function needs to change, but also for the suffix function. Some words have the suffix of "ing", "es", etc. These kinds of suffix couldn't be accepted as well.

8 Optimized methods

According to our analysis of the previous system result, the next step is using our useful knowledge to improve the current system.

We use a very useful method from the nltk library from Steven Bird, Ewan Klein, and Edward Loper (2009)[2] called stem(), which will help us detect the true prefix in every word.

And for the suffix, we defined a list called suffixes, which containing "s", "ed", "ing", "ed" and "ed". That will make sure that some candidate word which has these kinds of suffix will be removed from the result.

Moreover, for some misspelled words in the candidate files, we use Levenshtein/global edit distance package[3] following the equation:

$$editdistance.eval(t, d) > 1$$

That will make sure some misspelled words removed from the candidate files.

Finally, from the previous test the similarity values of the N-Gram, we choose 0.50 when using 2-Gram and 0.30 when using 3-Gram.

8.1 Optimized Result

After adding these optimized methods, the optimized results shown below:

Table 2: Results of optimized N-Gram analysis

Method	Similarity Value	Precision	Recall
2-Gram	0.50	3.24%	11.60%
3-Gram	0.30	3.00%	12.90%

8.2 Evaluation Result

From Table 2, the precision result has improved a lot. But the Recall result just remained the same. That means the system successfully remove some useless words from the result using the knowledge we obtained.

The reason why the system has a low recall value has several reasons. First of all, when opening "blends.txt" file, we easily find out that many lengths of the true blend words have less than 5. So our restriction condition removes several true blend words

from the candidate file. Furthermore, N-Gram also has some drawbacks like it doesn't deal with long-distance dependencies.[4] So the efficiency of this method is not very good. In future implementation, we may use other similarity methods to get the most efficient way to get as much as the blend words.

References

[1] Graham Poulter 2017.
Available at: https://github.com/gpoulter/python-ngram.

[2] Steven Bird 2009.
Available at: http://www.nltk.org/.

[3] Hiroyuki Tanaka 2017.
Available at: github.com/aflc/editdistance.

[4] N-grams.
Available at:https://fenix.tecnico.ulisboa.pt/downloadFile/845043405445783/sebentaLN-semana4-Ngramas.pdf.

[5] Paul Cook. Using social media to find english lexical blends. In *Proc. of EURALEX*, pages 846–854, 2012.

[6] Paul Cook and Suzanne Stevenson. Automatically identifying the source words of lexical blends in english. *Computational Linguistics*, 36(1):129–149, 2010.

[7] Kollol Das and Shaona Ghosh. Neuramanteau: A neural network ensemble model for lexical blends. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 576–583, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.

[8] Aliya Deri and Kevin Knight. How to make a frenemy: Multitape fsts for portmanteau generation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 206–210, 2015.

[9] Jacob Eisenstein, Brendan O'Connor, Noah A. Smith, and Eric P. Xing. A latent variable model for geographic lexical variation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1277–1287, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.