Paraphrase Identification Using String-based, Vector-based, and Knowledge-based Similarity

Zikai Zhu
ANLY521 Final Project
Georgetown University
zz222@georgetown.edu

Abstract

This document contains methods for sentence similarity using string similarity, vector-based similarity, and knowledge-based similarity. By using sentence similarity, it identifies if the sentences are paraphrased. This paper is based on some previous work using the same dataset and similar methods. As there are more and more news on the internet, more people just copy or paraphrase the sentences from elsewhere for more benefit. In this paper, it uses sentences from news to label the paraphrasing.

1 Introduction

Paraphrase is a serious problem these days. Many people want more benefits through spending less time and efforts. That causes they steal works from other people. As there are more and more news and information on the internet today, it is also very difficult to check the news one by one to see if they are paraphrased. Natural Language Processing exists and there is a way for people to check sentence similarity to find out whether sentences are paraphrased or not.

This paper reveals three ways to calculate the sentence similarity, including string similarity, vector-based similarity, and knowledge-based similarity. By comparing the combinations of three methods, a best method will be found. Based on many research showing that simple lexical matching will fail to detect the paraphrase, this paper will also test to see which of above methods works the best.

Fernando and Stevenson(2008) gave an example that

> S1: The Iraqi Foreign Minister warned of disastrous consequences if Turkey launched an invasion of Iraq.
>
> S2: Iraq has warned that a Turkish incursion would have disastrous results.

are paraphrased but failed to be detected by lexical matching because of words like consequences and results or invasion and incursion.

The following sections are dataset, background study on this field, methods, results and discussion.

2 Dataset

The **Microsoft Research Paraphrase Corpus** dataset is used in this paper. The reason for using this one is it is free and available online. It has 5801 paraphrase sentence pairs from web news sources. It has a label column standing for whether they are paraphrased or not. 1 means paraphrased, 0 means not. For machine learning, the dataset is splitted into train set and test set, with 4076 pairs in train and 1725 pairs in test.

3 Background

There are a lot of research done on paraphrase identification using knowledge-based and vector-based sentence similarity. They think doing simple lexical matching does not work well in paraphrase detection because the use of synonyms. Mihalcea et al. (2006) describes an example that the cosine similarity with tf-idf weight will fail to detect the paraphrase while knowledge-based methods will detect it. This paper will use 5 String similarity methods, 2 vector-based methods, and 6 wordnet based methods to compare the result accuracy with it.

Islam and Inkpen (2007) uses combination of semantic and string similarity and finds a new metric which performs better that the equation from Mihalcea et al. (2006). Mihalcea et al. (2006) also gave a metric to calculate the sentence similarity through word similarity from Wordnet (e.g. (Leacock and Chodorow, 1998; Wu and Palmer, 1994; Resnik, 1995)). This paper will try to improve the

metric and also make use of logistic regression to predict the label so that the accuracy will increase in some degree.

4 Methods

Mihalcea et al. (2006) introduces a equation to calculate the sentence similarity from word similarity.

$$Sim(T_1, T_2) = \frac{1}{2}\left(\frac{\sum_{w \in T_1}((maxSim(w,T_2) * idf(w))}{\sum_{w \in T_1}(idf(w))} + \frac{\sum_{w \in T_2}((maxSim(w,T_1) * idf(w))}{\sum_{w \in T_2}(idf(w))}\right) \quad (1)$$

To explain this clearly, an algorithm is shown below.

Algorithm 1 Sentence Similarity

Result: $Sim(T_1, T_2)$
for $w1 \in T_1$ **do**
  for $w2 \in T_2$ **do**
    $tempScore = Sim(w1, w2)$
    $scoreList.append(tempScore)$
  end
  $bestScore = max(scoreList) * idf(w1)$
  Sum all the best scores
end
Get the average of Sum

The Algorithm 1 showing above explains the first part in the equation, which is to find the score of T_1 based on T_2. Since doing only this is not symmetric, which means $Sim(T_2, T_1)$ might give different score. Therefore, it is necessary to calculate the average of $Sim(T_2, T_1)$ and $Sim(T_1, T_2)$.

4.1 String Similarity

Five string similarities are used for this method, which are "NIST", "BLEU", "Word Error Rate", "Longest common substring", and "Levenshtein distance".

- **NIST**: NIST comes from the US National Institute of Standards and Technology. It is based on the BLEU metric, but with some alterations. Where BLEU simply calculates n-gram precision adding equal weight to each one, NIST also calculates how informative a particular n-gram is. That is to say when a correct n-gram is found, the rarer that n-gram is, the more weight it will be given
- **BLEU**: BLEU (bilingual evaluation understudy)'s output is always a number between 0 and 1. It simply calculates n-gram precision adding equal weight to each one.
- **Word Error Rate**: WER can be calculated as following:

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}$$

  where S is the number of substitutions,

  D is the number of deletions,

  I is the number of insertions,

  C is the number of correct words,

  N is the number of words in the reference (N=S+D+C)

- **Longest common substring**: LCS is to find the longest string (or strings) that is a substring (or are substrings) of two strings. For example, the longest common substring of the strings "ABABC", "BABCA" and "ABCBA" is string "ABC" of length 3. Other common substrings are "A", "AB", "B", "BA", "BC" and "C".
- **Levenshtein distance**: Levenshtein distance is also referred to be edit distance, a string metric for measuring the difference between two sequences. The distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other. For example, the Levenshtein distance between "kitten" and "sitting" is 3:

  kitten → sitten (substitution of "s" for "k")

  sitten → sittin (substitution of "i" for "e")

  sittin → sitting (insertion of "g" at the end)

4.2 Vector-based Similarity

- **Word2Vec Vectorizer**: This method uses a truncated version of the 300-vectors trained by Google Research on an enormous corpus of Google News to convert words to vector and then calculate the cosine similarity between vectors.
- **Tf-Idf Vectorizer**: This method uses Tf-Idf vectorizer and cosine similarity for the sentence similarity.

4.3 Knowledge-based Similarity

- **Path Similarity**: It is the default method in Wordnet. The score means how similar two word senses are, based on the shortest path that connects the senses in the is-a (hypernym/hypnoym) taxonomy.

- **Leacock-Chodorow Similarity**: Leacock Chodorow(1998) explains that the word similarity is:

$$Sim_{lch} = -\log \frac{length}{2*D} \quad (2)$$

where length is the length of the shortest path between two concepts using node-counting, and D is the maximum depth of the taxonomy.

- **Wu-Palmer Similarity**: Wu Palmer(1994)'s similarity is:

$$Sim_{wup} = \frac{2depth(LCS)}{depth(concept_1)+depth(concept_2)} \quad (3)$$

which measures the depth of least common subsumer, and depth of the concept in the Wordnet taxonomy.

- **Resnik Similarity**: Resnik(1995) introduces the similarity by the information content(IC) of LCS.

$$Sim_{res} = IC(LCS) \quad (4)$$

- **Jiang-Conrath Similarity**: Jiang Conrath(1997) introduces the equation by:

$$Sim_{jcn} = \frac{1}{IC(concept_1)+IC(concept_2)-2IC(LCS)} \quad (5)$$

- **Lin Similarity**: And finally, Lin(1998) introduces it as:

$$Sim_{lin} = \frac{2IC(LCS)}{IC(concept_1)+IC(concept_2)} \quad (6)$$

5 Results

Table 1 shows the accuracy of the model containing different combinations of methods. Surprisingly, the String similarity method has the best accuracy. The hypothesis was that a model containing all the predictors will perform the best. However, the table shows it obviously not. The vector-based string similarity performs the worst. It also shows that as long as the model contains string similarity predictors, the model's accuracy will be relatively high.

Methods	Accuracy
String	**73.58**
Vector	68.77
Wordnet	70.05
String + Vector	73.06
String + Wordnet	72.19
Wordnet + Vector	70.45
String + Vector + Wordnet	72.31

Table 1: Sentence Similarity Accuracy by Logistic Regression

6 Evaluation and Discussion

Comparing to Mihalcea et al. (2006)'s result, this method improves the accuracy. The accuracy for vector-based method increased from 65.4 to 68.77. The combined model presents the accuracy of 72.31, higher than 70.3.

Here are also some reason that why string similarity might work better than wordnet method. Since the label for this paraphrase dataset is done by human, so some sentence pairs might have disagreement, causing the label being ambiguous. Also, after check the dataset, it shows that most of them are pretty similar in their meaning, which shows that wordnet method might not be better than string similarity.

References

Fernando, S., and Stevenson, M. (2008). A semantic similarity approach to paraphrase detection, Computational Linguistics UK (CLUK 2008) 11th Annual Research Colloquium.

Islam, A., and Inkpen, D. (2007). Semantic similarity of short texts, Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2007), Borovets, Bulgaria, pp. 291-297.

Mihalcea, R., Corley, C., and Strapparava, C. (2006). Corpus-based and knowledge-based measures of text semantic similarity, Proceedings of the National Conference on Artificial Intelligence (AAAI 2006), Boston, Massachusetts, pp. 775-780.

Word error rate In Wikipedia. Retrieved May 2, 2019, from https://en.wikipedia.org/wiki/Word_error_rate