Text Semantic Similarity

Anurag Dhaipule and Arpit Saini

May 11, 2016

Abstract

In this paper we present a method to compute text similarity between two sentences using different versions of LCS and word similarity. Most of the text similarity measure algorithms is about computing similarity between two documents that are large. Even though the model presented in this paper can be used on large size documents, but all testing and analysis is done considering similarity is measured between two sentences.

1 Introduction

The idea behind the algorithm is to make use of word similarity score which in a way measures semantic similarity and add to it other scores which will take into account the structure of the sentences. Longest common subsequence in a way provides the measure of similarity between the structure of two sentences. There are couple of modifications also that will be done to get some score. Finally, combining all these, the algorithm presented here gives a score between 0 and 1 indicating the probability of of similarity. Higher value indicates a high probability of similarity between the two sentences. This core idea is taken from a paper named Semantic Similarity of Short Text by Aminul Islam and Diana Inkpen, Univeristy of Ottawa.

Some of the applications of text similarity are:

- Filtering particular set of sentences that matches given sentence from a given set of sentences. One example would be to extract tweets similar to a tweet of particular type.
- Extracting images based on the short description associated with images on web.
- Evaluation of machine translation by comparing its output with right translation.
- Rank pages based on similarity score.

2 Previous work

Previously lot of papers have been presented on measuring similarity between two documents. Some of which were developed decades ago.

One naive way to measure similarity would be measure similarity using bag of words. This involves finding cosine similarity with tf-idf weighting. This has

been implemented in code and we are considering this model as base case for comparison.

Other method includes implementing using machine learning techniques wherein the features are extracted out of text, algorithms are run and models are developed that with the help of two feature vector gives a score.

The last class of algorithm on which our algorithm is also based is called hybrid algorithms wherein different measures are taken. The different measures are weighted and the algorithm gives out a score of similarity.

3 Semantic Similarity Algorithm

These algorithms work on different set of parameters, takes their weighted sum and provides a score. Here we present how different parameters are calculated. Before we proceed with algorithms we also do what is called trimming. Trimming involves two step:

- Removal of all stop words like a , an, the, have etc.
- Removal of words that occur in both the documents. Count of these words is referred as δ

We determine the Least Common Subsequence (LCS) between given two strings. It is obvious that this doesn't help much if the consecutive characters in LCS are from different words but it helps when we have set of same words one after the other in both the sentences. Notice that just providing the length also is not a correct measure. The length of LCS with respect to the length of the entire sentence and hence we normalize it by using the following formulae.

$$v1 = \frac{(length(LCS(r,s)))^2}{(length(r)) * (length(s))}$$

Where r and s are two sentences whose similarity we are determining.

As we mentioned above the problem with LCS is that the characters of LCS need not be consecutive and hence we calculate another form of LCS, modified LCS that finds the Least Common Subsequence of characters that are consecutive in both the sentences. We consider two versions of this. One wherein the consecutive LCS starts at character 1 of smaller sentence and in the second version the consecutive LCS can start at any character of the shorter sentence.

Below is the pseudo code for implementing the two. These are modified LCS(MLCS) 1.

```
def MLCS1(s,r):
        #len(s) <= len(r).
        while len(s) > 0:
        if s in r:
            return len(s)
        else:
            s = s[:-1]
    return len(s)
```

.

Below is the implementation of Modified LCS(MLCS2).

```
def MLCS2(s,r):
        mclcsn = 0
    for len_gram in range(1, min_len+1):
        for i in range(min_len-len_gram+1):
            gram = small_sent[i:i+len_gram]
            mclcsn = max(mclcsn, MCLCS1(gram, long_sent))
    return mclcsn
```

.
All these values are given different weights and added as:

$$v = w_1 v_1 + w_2 v_2 + w_3 v_3$$

where $w_1 + w_2 + w_3 = 1$; v_2 and v_3 are MLCS1 and MLCS2. We found that setting $w_1 = 0.1$, $w_2 = 0.45$ and $w_3 = 0.45$ gives good results.

As explained before we also will be considering a word similarity into account in order to determine final score for the similarity of the two sentences.

We create a word similarity matrix, M1 with words of sentence r along rows and words of sentence s along columns. Every cell in matrix represents a pair of words. Matrix cells are filled by finding word similarity score between the pair of words that represent that cell using gensim module.

We also create another matrix,M2 that stores the v score between every pair of words.

Both these matrix are combined

$$M = \phi_1 M1 + \phi_2 M1$$

with $\phi_1 + \phi_2 = 1$.We found that setting equal weights gives good results.

Then we determine maximal element amongst all the values from matrix, remove its row and column from matrix and add the maximal element to a list, L. Keep doing this until the maximal element is zero or the size of matrix 0*0. Now sum all the elements in the list, L and store it in ψ.
The similarity score is calculated as

$$Similarity = \frac{(\delta + \psi)(m + n)}{2mn}$$

where m and n are the length of two given sentences without trimming.

4 Results and Analysis on Microsoft Paraphrase Corpus

The dataset used for experiment is Microsoft Paraphrase Corpus.

Along with the algorithm mentioned above we used cosine similarity on bag

of words representation. Following are the results for bag of words.

Cutoff	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
True Positives	2753	2751	2737	2684	2440	2015	1467	742	131	11
True Negatives	0	3	59	145	399	774	1066	1255	1322	1323
False Positives	1323	1320	1264	1178	924	549	257	68	1	0
False Negatives	0	2	16	69	313	738	1286	2011	2622	2742
Accuracy	0.68	0.68	0.69	0.69	0.70	0.68	0.62	0.49	0.36	0.33
Precision	0.68	0.68	0.68	0.69	0.73	0.79	0.85	0.92	0.99	1.00
Recall	1.00	1.00	0.99	0.97	0.89	0.73	0.53	0.27	0.05	0.00
F1 Score	0.81	0.81	0.81	0.81	0.80	0.76	0.66	0.42	0.09	0.01

With cosine similarity of bag of words, we found that higher the ratio of number of similar words compared to total words in both sentences, higher is the probability of similarity.

Following are the results with semantic similarity algorithm.

Cutoff	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
True Positives	2753	2753	2753	2752	2748	2685	2380	1578	552	81
True Negatives	0	0	0	2	40	153	526	1019	1271	1318
False Positives	1323	1323	1323	1321	1283	1170	797	304	52	5
False Negatives	0	0	0	1	5	68	373	1175	2201	2672
Accuracy	0.68	0.68	0.68	0.68	0.68	0.70	0.71	0.64	0.45	0.34
Precision	0.68	0.68	0.68	0.68	0.68	0.70	0.75	0.84	0.91	0.94
Recall	1.00	1.00	1.00	1.00	1.00	0.98	0.86	0.57	0.20	0.03
F1 Score	0.81	0.81	0.81	0.81	0.81	0.81	0.80	0.68	0.33	0.06

.

We noticed that the algorithm was not able to handle the sentences with negation words well. To handle this issue we looked at the wordnet module in the nltk library. To handle the presence of negation words we replaced the negation word and the following word adjective or verb with their respective antonym. We replaced the negation word and the following adjective or verb in the sentence with the antonym and used the new sentence to get semantic similarity.

Although the results were satisfactory, we decided to move forward with the gensim model for word similarity as it was giving better results.

Trying out different cut-offs, we decided to choose the cut-off, 0.697 that gave best precision. The results are shown below.

Cutoff	0.697
True Positives	2400
True Negatives	513
False Positives	810
False Negatives	353
Accuracy	0.71
Precision	0.75
Recall	0.87
F1 Score	0.80

Couple of example where algorithm fails:

sentence1 : However, commercial use of the 2.6.0 kernel is still months off for most customers.

sentence2: Commercial releases of the 2.6 kernel by major Linux distributors still remain months away.

Both the sentences above are similar but our algorithm predicts it as not similar. Firstly notice that the ratio of same words is less. Secondly, there are words like customers and distributors, which differ in meaning but are used to give the same meaning to the sentences.

sentence1 : Webster Police Chief Gerald Pickering said the victims appeared to be customers of the bank.

sentence2: One of the shooting victims died, according to Webster Police Chief Gerald Pickering.

The above two sentences are not similar but due to presence of lot of same words and similar words, algorithm tend to predict a high value of similarity.

5 Conclusion

Even though the results are satisfactory, improvements can be done by extending the core idea of this algorithm. More parameters can be considered and their weighted be taken. Since it can be a difficult task to take weighted sum of many parameters, machine learning algorithms can be used to determine best possible weights for these algorithms.

6 References

.

[1] Aminul Islam and Diana Inkpen, "Semantic Similarity of Short Texts"

[2] L. Allison, T.I. Dix, "A Bit-String Longest-Common- Subsequence Algorithm,"

[3] Fernando, S., and Stevenson, M. (2008). "A semantic similarity approach to paraphrase detection".