Text Similarity through Graph Matching

February 17, 2021

Abstract

The aim of this paper is to extract common pattern from a collection of short text document by calculating text similarity through graph matching. We first build a semantic graph for each text using dependency parsing. We used the graph from (Pan et al., 2020). Then we calculate distance between text by using graph structure, and cluster text pieces to form groups of text that have similar semantic meaning. Finally, we extract the common pattern in each cluster.

Introduction

Our first experiment is inspired by the Word Mover's Distance (Kusner et al., 2015). WMD calculates the distance between a sentence D_1 to a query sentence D_0. For each non-stop word $\{w_i\}_{i=1}^n$ in D_1, n is the number of non-stop words in D_1, calculate its distance $c(i,j)$ to each non-stop word $\{w_j\}_{j=1}^m$ in D_0 using Word2Vec embeddings. Then for each w_i, find the word w_{j^*} in D_0 with the shortest distance, where $j^* = argmin_j c(i,j)$. The final distance between D_0 and D_1 is $\sum_i d_i c(i,j^*)$, d_i is the nBOW vector.

Although WMD was able to achieve state of art results in many text similarity tasks, it ignores relationship among words in a sentence. For example, WMD is able to match "*Obama speaks to the media in Illinois*" to a query sentence "*The President greets the press in Chicago*" (Kusner et al., 2015) pretty well, but it fails to distinguish them from "*Obama, working for a press, was born in Chicago.*". This sentence matches many words with the query sentence, but the semantic meaning is completely different. We propose to leverage graph structure to enhance word-matching based edit distance by considering the dependency relationship among words in a sentence.

Methodology

Dependency Parsing Graph

We first construct a semantic graph for each sentence using dependency parsing (Pan et al., 2020). Each node is an entity, consisting of one or more tokens. Each edge is directed with edge type being the dependency parsing label. We use v_i and $\mathbf{x}_i$ to denote node i and its node features.

We use BERT (Devlin et al., 2019) to embed each word, and use mean pooling to represent a node with multiple words. We enhance WMD by incorporating both node match and edge match. The first step is to match nodes by calculating node similarity. To do that, we borrow the idea of travel distance proposed in WMD. We first define a query graph G_0 and a comparison graph G_1. For each vertex $\{v_i\}_{i=1}^n$ in a graph G_1, we first calculate the cosine similarity $s(i,j)$ with each vertex $\{v_j\}_{j=1}^m$ in the query graph G_0. Then for each v_i, find the vertex $v_{j_i^*}$ in G_0 with the maximum similarity, where $j_i^* = argmax_j s(i,j)$. If $s(i,j_i^*)$ is above a certain threshold, we see the vertex pair as matched. We denote the matched vertex pair as

$M(v_i, v_{j_i^*})$. The second step is to match edges. For two edges to match, the associated two node pairs must match, and the edge types and directions should also match. For example, if we have two vertex pair matches $M(v_{i1}, v_{j_{i1}^*})$ and $M(v_{i2}, v_{j_{i2}^*})$, then we can proceed to see if the edge $e(v_{i1}, v_{i2})$ and the edge $e(v_{j_{i1}^*}, v_{j_{i2}^*})$ have the same direction and edge type. In the previous example, "*Obama*" and "*The President*" is a vertex pair match, and "*speaks*" and "*greets*" is another vertex pair match. The connection between "*Obama*" and "*speaks*" is $(speaks \rightarrow$ *nsubj* $\rightarrow Obama)$, and the connection between "*The President*" and "*greets*" is $(greets \rightarrow$ *nsubj* $\rightarrow ThePresident)$, meaning the connections between the two vertex pairs also match. We call this an edge match.

The sentence level similarity is:

$$sim_{graph} = \frac{1}{|V_{graph}|} \sum_i w_i s(i, j_i^*) \tag{1}$$

$$dist_{graph} = 1 - sim_{graph} \tag{2}$$

$$w_i = \begin{cases} 1, \text{ if } v_i \text{ is part of an edge match} \\ \text{a constant number less than 1, if } v_i \text{ is not part of an edge match} \end{cases} \tag{3}$$

$$s(i, j) = \frac{\mathbf{x}_i \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|} \tag{4}$$

Once we have both WMD and graph distance, we compute the final distance as the weighted average of them. When $weight$ equals 0, the distance is WMD; when $weight$ equals 1, the distance is the graph distance.

$$dist = (1 - weight) \cdot \text{WMD} + weight \cdot dist_{graph} \tag{5}$$

For the first stage, we treat the thresholds and w_i as hyperparameters and manually tune them. In the second stage, we will use metric learning to learn them.

One of the challenges with using dependency parsing graph is that with long sentences where a lot of information is decorative but not contribute to the main meaning of the sentence, it is hard to extract and compare the main information, but compares a lot of unimportant nodes.

Semantic Role Labeling

For each verb in the sentence, we can use semantic role labeling (SRL) to get its subject and object, and for the subject-verb-object triple. We use AllenNLP to do SRL. We are considering the following ways to compute similarity using SRL. We define the first level of verbs $\{V^1\}$ to be the the highest level verbs, meaning that they do not exist in any subject or object. The second level verbs $\{V^2\}$ are verbs that exist in the subject or object linked by the first level verbs, and so on.

We use word2vec embedding for each word.

Method1

1. Get the list of all triples $\{T\}$, and the list of triples $\{T^1\}$ linked by $\{V^1\}$ for each sentence

2. For each $v_c^1 \in \{V_c^1\}$ in the candidate sentence, calculate its cosine similarity $s_v(v_c^1, v_q^1)$ with each $v_q^1 \in \{V_q^1\}$ in the query sentence. Find the v_q^{1*} that has the highest similarity $s_v(v_c^1, v_q^{1*})$.

3. Compare $s_v(v_c^1, v_q^{1*})$ with a threshold t. If $s_v(v_c^1, v_q^{1*}) \geq t$, we call them matched triples. For matched triples, calculate the similarity $s_s(s_c^1, s_q^1)$ between subject s_c^1 and subject s_q^1. We can use either WMD to calculate the similarity, or calculate the cosine similarity of the mean of word2vec embedding of each phrase. Likewise, se can calculate the similarity of objects $s_o(o_c^1, o_q^1)$. By adding the similarity of subjects, verb, and objects, we get the similarity score of the triple $s_t(t_c^1, t_q^1)$. If $s_v(v_c^1, v_q^{1*}) < t$, $s_t(t_c^1, t_q^1) = 0$

$$s_t = \mathbb{1}\{s_v(v_c^1, v_q^{1*}) \geq t\}[s_s(s_c^1, s_q^1) + s_v(v_c^1, v_q^{1*}) + s_t(o_c^1, o_q^1)] \tag{6}$$

4. Remove all $\{T^1\}$ from $\{T\}$ from both sentences. For the matched triples, remove all of their child triples, which is, the triples linked by lower level verbs

5. With $\{T^1\}$ removed, $\{T^2\}$ now become the highest level triples. Repeat step 2-4 until no triples left in the candidate sentence.

6. Add up all s_t

$$s = \sum s_t \tag{7}$$

Method2

Leverage sentence simplification. For some complicated sentences, SRL may return very long subjects or objects, which is not good for calculating similarity. By doing sentence simplification, we can break down long sentences into shorter sentences, thus reducing the length and complexity of the components of SRL triples. One of the limitations of WMD is that it cannot generate well to sentences with different length. With sentence simplification, we can keep both sentences in relatively similar length.

Experiment Protocol

The experiment will compare the quality of text similarity generated by the proposed method and the benchmark WMD method.

We use the Kaggle financial news dataset, and pick one sentence as the query sentence. Then we manually go over the dataset and pick 30 sentences that share at least one semantically similar key phrase as the query sentence. Once we have the 30 sentences, we manually give the 4 scores to each sentence: whole sentence similarity, subject similarity, action similarity, and object similarity. For each similarity criteria, the group with positive scores (the true positive group) should have higher calculated similarity than the group with 0 scores (the true negative group). We denote one sentence from the true positive group and one sentence from the true negative group as a pair. For each method, we compare the similarity scores for each pair of sentences and record the number of pairs that is correct.

Pre-Training

Pretrained models like BERT consistently perform better than our previous methods, so we can pre-train a model to encode the semantic information in the embeddings. Even with better performance than unsupervised learning, BERT model is not perfect in calculating sentence similarity. In some cases, it is even worse than GloVe **?**. In addition, BERT model is not able to encode the sentence based on different aspects, but rather gives a fixed encoding for each sentence. We will train a content-aware model which is able to encode and retain the key information of interest, thus at inference time, we can give different aspects to the sentences and the model will return different embeddings.

For complicated sentences, similarity sometimes depends on the aspect we are looking at. For example, let query sentence be *Google disclosed its third quarter net profits to be* 5% *higher than the last quarter, which is due to the declining of US dollar*, let candidate1 be *The fed is announcing a new policy which will cause a decline in the US dollar*, and let candidate2 be *Microsoft is having a huge revenue increase after announcing its new product*. Candidate1 can be more similar to the query sentence than candidate2 if we are focusing on the US dollar value, but candidate2 can be more similar to the query sentence if we are focusing on the technology company's revenue. With a fixed sentence encoding, we lose the flexibility to compare sentence similarity based on different aspects. In this work, we propose a pre-training method in an interactive mode, such that at inference time, we are able to input an "aspect phrase" to let the model know the key information to retain.

Entailment Model

The model inputs are two sentences, and one aspect phrase, and the model output is a similarity score. We will use attention and have the model to give higher attention weights to the part that corresponds to the aspect phrase. We can use siamese modeling, and have one neural network to encode two sentences simultaneously. Loss can refer to the sentence-bert paper.

In this way, at inference time, we can either input one sentence and one aspect phrase to calculate the sentence embedding, or two sentences and one aspect phrase to calculate similarity.

With different aspect phrases, the same sentence pair can have different similarity scores. We will have to construct a dataset for this. Maybe use Amazon Mechanical Turk?

Siamese Network with Adversarial Training

The model will first use Siamese network as an encoder to embed each sentence into a vector, then the two vectors will be used to decide if the two sentences are similar. Each Siamese encoder branch takes one sentence and one shared emphasize phrase. The adversarial network (generator) can only modify the phrase to fool the siamese network.

$$v_1, v_2 = Siamese(s_1, s_2, k) \tag{8}$$

s_1 and s_2 are two sentences, and k is the key phrase. At inference time, we are able to input one sentence and an emphasize phrase to get the representation:

$$v_x = Siamese(x, k) \tag{9}$$

Siamese Network Structure

We fine tune a Bert model, so the network starts from two identical Bert encoders to encode the sentences respectively. We can try (a) using the average pooling of token vectors, or (b) [CLS] to represent the sentence. The weights of the sentence encoder Bert model will be trainable. We could try (a) tune all parameters, or (b) tune only the last 2-3 layers.

We use another Bert model to encode the key phrase. This model will not be trainable. The representation of this key phrase will be the same across all sentence pairs.

$$v_1 = Bert_1(s_1) \tag{10}$$
$$v_2 = Bert_1(s_2) \tag{11}$$
$$v_k = Bert_2(k) \tag{12}$$

Where s_1 and s_2 are the input sentences, k is the key phrase.

We calculate the similarity between s_1 and s_2 using the below equation. It requires both sentences to be similar to the key phrase to get a high similarity score.

$$s(s_1, s_2) = sigmoid((v_1^T v_k)(v_2^T v_k)) \tag{13}$$

Triplet Loss

Consider three-sentence triplet, q is the query sentence, p is a positive sentence, and n is a negative sentence, the triplet loss is

$$L_t = \max(s(v_q, v_n) - s(v_q, v_p) + M, 0) \tag{14}$$

$s(\cdot)$ is a similarity measure of two vectors, M is a margin to ensure that p is at least M closer to q than n.

Training Data

For each sentence pair, generate one or more negative pairs by changing the key vector. We can extract some phrases for each positive sentence pair, and add negative sentence pairs by changing their original phrase to a random other phrase (edited).

Key phrases could be:

1. Common subject
2. Common subject plus predicate
3. Phrases mentioned in object from one sentence, which is also mentioned in the other sentence, not necessarily in object
4. Use WMD to find the most similar words, or use our graph to find the most similar nodes/phrases

we can also group the phrases and create a more standard phrase pairs. For example, microsoft inc, microsoft, MSFT, can all be microsoft; income, profit, revenue can all be revenue, etc. Then we can group all the positive sentence pairs by the standardized phrases. For example, all the sentence pairs mentioning microsoft got grouped together, all the sentence mentioning profit got grouped together. Then we can create more positive pairs by swapping the sentences within one group. (Unsure about this part. We may not need more positive sets)

Joint Loss

We borrow the joint loss idea from nicosia2017accurate.

Total loss is consist of the contrastive loss and the logistic loss. Contrastive loss measures the similarity of the two sentence vectors coming out of the Siamese network, and the logistic loss measures the output of the final entailment prediction.

$$\mathcal{L}_{tot} = \lambda_c \mathcal{L}_c + \mathcal{L}_l \tag{15}$$

Where λ_c is a hyperparameter to tune the weights of contrastive loss.

Given a dataset $X = \{< v_1^i, v_2^i, y^i >\}$, where $y = 1$ if the two sentences are entailment, and 0 otherwise.

$$\mathcal{L}_c = \sum_{i=1}^{N} L_c^i(v_1^i, v_2^i, y^i) \tag{16}$$

Where the contrastive loss L_c^i for each instance is

$$L_c^i = y^i d(v_1^i, v_2^i)^2 + (1 - y^i) \max(M - d(v_1^i, v_2^i), 0)^2 \tag{17}$$

Where $d(s_1, s_2)^2$ is the square of euclidean distance. M is a margin for negative pairs. The square of euclidean distance will only be taken into consideration if it is within a margin M.

$$\mathcal{L}_l = \sum_{i=1}^{N} y^i \log(\tilde{y}^i) + (1 - y^i) \log(1 - \tilde{y}^i) \tag{18}$$