[Team ‘sso_baseline’: Sanghyun Hong, Octavian Suciu, Snehesh Shrestha]

Part 1: Establish and Analyze a Baseline

Method

We create a word embeddings dictionary by using ‘paragram-phrase-XXL’ and backoff to ‘paragram_300_sl999’ for computing the baseline performance. We, first, tokenize the each sentence in the pair by using **NLTK**. Then, we search each word in the word embeddings dictionary and return the sentence embedding as the average of all individual embedding vectors. (c.f., note that the words not in the word embeddings dictionary are ignored). After that, we compute the cosine similarity of the two sentences in the pair and scale the value to the 0-5 scale required by the task. Lastly, we use the “*correlation-noconfidence.pl*” script to obtain the accuracy scores.

Results

answer-answer	headlines	plagiarism	postediting	question-question	Average performance
0.67397	0.71794	0.82125	0.83391	0.65056	**0.739526**

Analysis Part

In the following 2 sections, we highlighted exact matches in yellow and variations in other colors for all the examples that we enlisted.

[Analysis 1] Categories that the baseline works well:

1. It works well for the cases where the words in the first sentence mostly overlaps to the words in the second other sentence. As you can see in the following examples, the second sentences include extra words for the extra informations.
    a. The baby is laughing and crawling.
       A baby is laughing and crawling on the floor.
    b. A parrot is talking.
       A parrot is talking into a microphone.
2. The second cases in which it works well consists of similar sentences with variation of words. In here, the variation is using the different grammatical forms, synonyms, or hypernyms, and the extra informations can be added to each sentences as we’ve seen in the above.
    a. Someone is slicing a tomato.
       Someone is slicing fruit.
    b. Someone is slicing two uncooked racks of ribs apart.
       Someone is cutting a piece of meat.
3. It works well with the cases where the words in two sentences are similar, but the orders are different. The examples are stated as follows:
    a. What can I do about out of square rough opening for new windows?
       What can I do about a Rough opening that is REALLY out of square?
    b. Vector space representation results in the loss of the order which the terms are in the document.
       The order in which terms appear in the document is lost in a vector space representation.

[Analysis 2] Categories that the baseline does not work well:

1. It cannot work well in the cases where one sentence has a lot of overlappings of words, but the core meaning of the sentence is negated delivering opposite meanings. The examples are follows:
    a. It's not a good idea.
       It's not just a good idea, it's an excellent idea.

b. You should do it.
You should never do it.

c. It's not a good idea.
It's a good question.

d. Yes, you have to file a tax return in Canada.
You are not required to file a tax return in Canada if you have no taxable income.

Reason: This can happen because words flipping the meaning is not captured well by the embeddings that we used. (i.e., the embedding cannot express the opposite meanings well, since it’s not proposed to capture the synonyms, antonyms, but it captures the relations between words) Unless there is not enough overlappings, the two sentences have similar scores.

2. The cases where two sentences share all the words in common, but the key word is different.

a. What is the significance of the cat?
What is the significance of the artwork?

b. What kind of socket is this?
What type of faucet is this?

Reason: This could happen because the sum of the embeddings for the common words overweights the embedding difference in the different words. In the case ‘a’, ‘cat’ and ‘artwork’ can be close depending on our training data. As ‘cat’ might have appeared in some artwork or an artist working on artwork might have owned cats, so during the training, embeddings might capture relationship. For the second one, given that most of the words are same or similar (type ~= kind) they are scored as similar while the topic words are completely different.

3. If the sentence has opposite sentiments, the baseline does not work well.

a. The legislators thus refused to support President George Bush's call to the project.
Renièrent legislators and supports the call to the project of President George Bush.

b. "Yes, we have a serious problem.
"We have made great progress.

Reason: The sentiment of ‘refused to support’ and ‘support’ or ‘serious problem’ and ‘great progress’ are exactly opposites. However, even though it seems that the embeddings should have captured this, it does not. Since it is not uncommon to see these appearing in similar contexts thereby being captured in the embeddings as being closely related in our application.

4. In case where words may appear in similar contexts with different meaning, the baseline does not work well in differentiating those two sentences. An example is below:

a. A man is burning a flag.
A man is shooting a gun.

Reason: Words like ‘burning’ and ‘shooting’ are similar in the sense that they might come up in the violent contexts. The embedding captures these. The ‘flag’ and ‘gun’ may also appear in patriotic events or stories so similarly embeddings capture them during training. However, acts like ‘burning a flag’ and ‘shooting a gun’ are completely two different things. Thus, the baseline fails to capture this difference and considers them similar.

Part 2: Develop your Own Method

Implementation Summary:

Our implementation for for part 2 builds on the baseline by adding an additional embedding and using the labeled data while we train the classifier. We use the skip-thoughts embeddings as an additional embeddings [https://github.com/ryankiros/skip-thoughts] in an attempt to bootstrap the Paragram embedddings. These sentence-level embeddings used for modelling the surrounding of each text passage analyzed. The reason we chose this is because they proved successful in the same task for a previous year and they are publicly available. Our system builds a neural network with one single softmax activation by training over the data from previous years and validating over a dev set for a decreased variance. We combine both embeddings into 5103 features comprising of linear combination of features from both pairs as well as the cosine similarity computed at the previous task. Even if our methods improves the performance a lot on several datasets (headlines: 0.71 to 0.76, question-question: 0.65 to 0.67), there are some other that shows the decreased performance (answer-answer: 0.67 to 0.64). In addition, there are the datasets showing the tie-performances (e.g., plagiarism and postediting). We computed the averages for the part 2 outputs and it's **0.7491** which is **0.096** higher than the baseline **0.7395**. Moreover, we considered the other embedding methods such as *Word2Vec*, *Doc2Vec*, or *Glove*, however, due to the time limitation, we cannot include them to our current implementation.

Results

answer-answer	headlines	plagiarism	postediting	question-question	Average performance
0.64856	0.76825	0.82025	0.83491	0.67347	**0.749088**

Failed Attempts:

The following were the failed attempts that we explored. We approached developing our method 3 different ways:

1. **Analyzing the baseline data**
   We ran our baseline on all training data and test data and measured the difference in gold scores (GS) to the baseline scores (BS.) We sorted by the difference and reviewed both best performing and the worst performing sentence pairs. We categorized the worst performing pairs into various groups and evaluated approaches that improve the scores. These were helpful in our brainstorming of new ideas, and picking existing implementations that may or may have not been used in the STS context.

2. **Brainstorming new ideas**
   We brainstormed many ideas and narrowed down to a handful based on what was practical given the timeline and workload. Here are are some that we discussed/ evaluated:
   a. **Sentence Alignment** between two sentences can help focus on the key components of the two sentences which makes it easier to compare the similarity. We did not pursue this because, it seemed even harder problem than the STS and getting the annotated dataset to match our problem was not easy.
   b. **Dependency Parser** can be used to figure out which words are modified by and dependent on other words. It seemed that this could help us with the observation of where negation changed the meaning of the sentence but the embedding did not capture it. In our proof of concept tests, dependency parsers did not seem to capture well the dependencies as we expected. The scores in fact became worse so we abandoned this approach.
   c. **POS Tagging** The idea was to detect nouns, verbs, adjectives, and adverbs. Our idea was to ignore the other words. We thought by comparing the actions, the objects, and words that embellished them, we could figure out how similar sentences were. However, after further discussions and evaluations of some arbitrary sentences where the alignments were not so 1-to-1, we quickly saw the issue that there would be far too many ambiguity cause failures. So we did not pursue this.
   d. **FrameNet + WordNet + N-Degrees of Freedom Word Relationships** The idea here was to use a fitting model. Using framenet, we thought we could evaluate the key-words and generate phrase using only words found in both sentences. Then use WordNet to generate synonyms, hypernyms, and antonyms. The notion here was could we in N-degrees of freedom of words, for example antonym of an antonym could be somewhat relatable, can we find a path from sentence one to sentence two vice versa. The short the paths were the more similar the sentences were. However, this ended being too

complicated and with sentences with different structures, there were too many combinations which lead to a weak performance.

e. **Important Words Extraction** Since in the worst case scenarios, there were lots of words overlapping between the sentences and even though the keywords were different, there was a big disparity in the GS and the BS. To mitigate this we wanted to find the important words and compare the similarity. By creating a frequency table of all words and using the notion that the less frequent a word is, the more important they are. For example "what", "and", "the" etc will appear many times, while "crocodile", "toyota", “delicious” etc will not. Additionally another table of negation words could help count negation or double negation. We did not use this as we were not confident how well this would work.
f. **Sentiment Analysis** We noticed in the data that there were sentence pairs with opposite sentiments, however, the baseline seems to bucket them as similar due to words overlap and words that represent opposite sentiments appear in similar contexts. This was a clear opportunity which we thought could help our worst cases and help the overall performance. We implemented this and tested in our data. Unfortunately, there were two issues:
    i. Sentiment Analysis itself is not perfect:
    We used NLTK packages’ Vader sentiment dataset as our starting point as it was readily available and easy to test our hypothesis. While it helped in some cases, it was clear that the trained data was not very relatable to our dataset. The sentiment analysis did not always perform well. For new words (which was often) the neutral score was high. So using the sentiment analysis did not give much boost in STS task.
    ii. Effect on already performing good scores:
    While it had some positive results, there were far too many negative effects on already well performing results. Unfortunately, this caused the overall score to drop even further.

    We think using a better dataset could perhaps give a better sentiment analysis. Since we did not have a better dataset we could use, we did not use this as the results were worse.

3. **Evaluating existing implementations in the STS area**
    a. **CHARGRAM**: Embedding Words and Sentences via Character n-grams
    This paper was by the same authors Wieting et. al. It looked promising. Here are the most important reasons why:
        i. Chargram is a n-gram character embedding. Because Chargram is not limited by word boundaries, it can capture sub-word or even cross-words representations.
        ii. Paragram-Phrase does not handle new words not in the vocabulary. So when a new word shows up, it does not work at all. Since Chargram is character based so it has no concept of word per se, it essentially embeds character sequences. So in these kinds of cases, chargrams performs better than baseline.
        iii. Paragram does not model word order or repeated words. Chargram handles this better as n-gram of character would model both spaces and word order and also repeat words.
        iv. Where Chargram seems to be in par or even slightly less is when there are lots of words overlap but key words meaning is different compared to the other. So this is why we decided to continue to explore sentiment analysis approach.

    Unfortunately, the base code for CHARGRAM does not come with pre-trained data and we need to train it. The chargram uses NN learning approach. We ran the training for about a week in our computer and training did not complete. Due to lack of time and resources, unfortunately, even though this method showed promise, we could not explore fully.
    b. **Other approaches (Future works)** The most promising direction of this work is to use the neural network as a ensemble framework for training and, inside the network, all the embedding methods are considered as perceptrons as Samsung Poland Research team showed us in their paper. In the same line of the solutions, people consider the other embedding methods such as *Word2Vec*, *Doc2Vec*,

Glove, etc., which blends more insights to their framework like considering the relationships between words. We take this approach as well, however, due to the time limitation, we simply put it to our future work. More to the future work, we can do other computation when we calculate the embeddings for each sentence instead of using average such as multiplication or log-likelihood.

Part 3: Crosslingual Extension

We use a very simple approach for this part. We first identify the language of each individual sentence using the ‘langdetect’ python package, then we extract the complement of the one detected as english (because the package has the best accuracy in english). We translate the spanish sentences using Google Translate and replace the sentences with the translations. This gives us the ability to use the baseline on the fully english MTed corpus. This simple trick gives us an average performance of **0.85**.