Comparing Sentence Similarity Methods

Finding the similarity between two sentences is central to many NLP applications.

Yves Peirsman • May 2, 2018

Word embeddings have become widespread in Natural Language Processing. They allow us to easily compute the semantic similarity between two words, or to find the words most similar to a target word. However, often we're more interested in the similarity between two sentences or short texts. In this blog post, we'll compare the most popular ways of computing sentence similarity and investigate how they perform. For people interested in the code, there's a companion Jupyter Notebook with all the details.

Sentence similarity is typically calculated by first embedding the sentences and then taking the cosine similarity between them. (Source)

Many NLP applications need to compute the similarity in meaning between two short texts. Search engines, for example, need to model the relevance of a document to a query, beyond the overlap in words between the two. Similarly, question-and-answer sites such as Quora need to determine whether a question has already been asked before. This type of text similarity is often computed by first embedding the two short texts and then calculating the cosine similarity between them. Although word embeddings such as word2vec and GloVe have become standard approaches for finding the semantic similarity between two words, there is less agreement on how sentence embeddings should be computed. Below we'll review some of the most common methods and compare their performance on two established benchmarks.

Data

We'll evaluate all methods on two widely used datasets with human similarity judgements:

- The STS Benchmark brings together the English data from the SemEval sentence similarity tasks between 2012 and 2017.
- The SICK data contains 10,000 English sentence pairs labelled with their semantic relatedness and entailment relation.

The table below contains a few examples from the STS data. As you can see, the semantic relationship between the two sentences is often quite subtle: the sentences `a man is playing a harp` and `a man is playing a keyboard` are judged as very dissimilar, although they have the same syntactic structure and the words in them have very similar embeddings.

	sent_1	sent_2	sim
0	A girl is styling her hair.	A girl is brushing her hair.	2.5
1	A group of men play soccer on the beach.	A group of boys are playing soccer on the beach.	3.6
2	One woman is measuring another woman's ankle.	A woman measures another woman's ankle.	5.0
3	A man is cutting up a cucumber.	A man is slicing a cucumber.	4.2
4	A man is playing a harp.	A man is playing a keyboard.	1.5

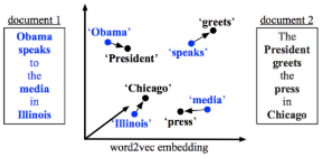
Example sentence pairs from the STS Benchmark.

Similarity methods

There is a wide range of methods for calculating the similarity in meaning between two sentences. Here we take a look at the most common ones.

Baselines

The easiest way of estimating the semantic similarity between a pair of sentences is by taking the average of the word embeddings of all words in the two sentences, and calculating the cosine between the resulting embeddings. Obviously, this simple baseline leaves considerable room for variation. We'll investigate the effects of ignoring stopwords and computing an average weighted by tf-idf in particular.



The Word Mover's Distance between two documents

Word Mover's Distance

is the minimum cumulative distance that all words in document 1 need to travel to exactly match document 2.

One interesting alternative to our baseline is Word Mover's Distance. WMD uses the word embeddings of the words in two texts to measure the minimum distance that the words in one text need to "travel" in semantic space to reach the words in the other text.

Smooth Inverse Frequency

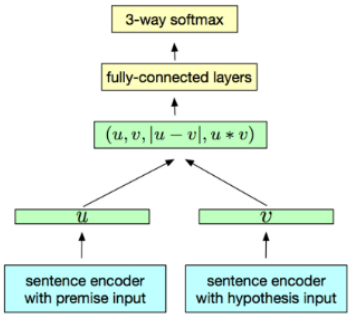
Taking the average of the word embeddings in a sentence tends to give too much weight to words that are quite irrelevant, semantically speaking. Smooth Inverse Frequency tries to solve this problem in two ways:

1. Weighting: like our tf-idf baseline above, SIF takes the weighted average of the word embeddings in the sentence. Every word embedding is weighted by `a/(a + p(w))`, where `a` is a parameter that is typically set to `0.001` and `p(w)` is the estimated frequency of the word in a reference corpus.
2. Common component removal: next, SIF computes the principal component of the resulting embeddings for a set of sentences. It then subtracts from these sentence embeddings their projections on their first principal component. This should remove variation related to frequency and syntax that is less relevant semantically.

As a result, SIF downgrades unimportant words such as `but`, `just`, etc., and keeps the information that contributes most to the semantics of the sentence.

Pre-trained encoders

All methods above share two important characteristics. First, as simple bag-of-word methods, they do take not word order into account. Second, the word embeddings they use have been learned in an unsupervised manner. Both these traits are potentially harmful. Since differences in word order often go hand in hand with differences in meaning (compare `the dog bites the man` with `the man bites the dog`), we'd like our sentence embeddings to be sensitive to this variation. Additionally, supervised training can help sentence embeddings learn the meaning of a sentence more directly.



Pre-trained sentence encoders rely on tasks such as Natural Language Inference to learn sentence embeddings that can be used in transfer tasks.

This is where pre-trained encoders come in. Pre-trained sentence encoders aim to play the same role as word2vec and GloVe, but for sentence embeddings: the embeddings they produce can be used in a variety of applications, such as text classification, paraphrase detection, etc. Typically they have been trained on a range of supervised and unsupervised tasks, in order to capture as much universal semantic information as possible. Several such encoders are available. We'll take a look at InferSent and the Google Sentence Encoder.

InferSent is a pre-trained encoder that was developed by Facebook Research. It is a BiLSTM with max pooling, trained on the SNLI dataset, 570k English sentence pairs labelled with one of three categories: entailment, contradiction or neutral.

The Google Sentence Encoder is Google's answer to Facebook's InferSent. It comes in two forms:

- an advanced model that takes the element-wise sum of the context-aware word representations produced by the encoding subgraph of a Transformer model.
- a simpler Deep Averaging Network (DAN) where input embeddings for words and bigrams are averaged together and passed through a feed-forward deep neural network.

The Transformer-based model tends to give better results, but at the time of writing, only the DAN-based encoder was available. In contrast to InferSent, the Google Sentence Encoder was trained on a combination of unsupervised data (in a skip-thought-like task) and supervised data (the SNLI corpus).

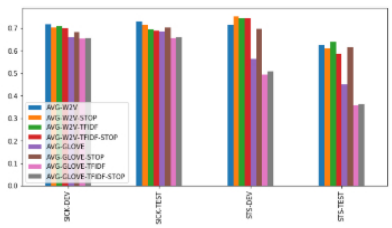
Results

We tested all the methods above by getting their similarities for the sentence pairs in the development and test sets of the SICK and STS data, and computing their correlation with the human judgements. We'll mostly work with Pearson correlation, as is standard in the literature, except where Spearman correlation gives different results.

Baselines

Despite their simplicity, the baseline methods that take the cosine between average word embeddings can perform surprisingly well. Still, a few conditions have to be met:

- Simple word2vec embeddings outperform GloVe embeddings.
- With word2vec, it is unclear whether using a stoplist or tf-idf weighting helps. On STS it sometimes does, on SICK it does not. Simply computing an unweighted average of all word2vec embeddings consistently does

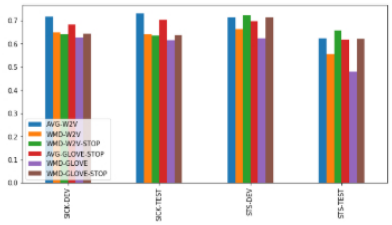


Our simple baselines can perform surprisingly well.

pretty well.
- With GloVe, using a stoplist is crucial to obtaining good results. Using tf-idf weights does not help, with or without a stoplist.

Word Mover's Distance

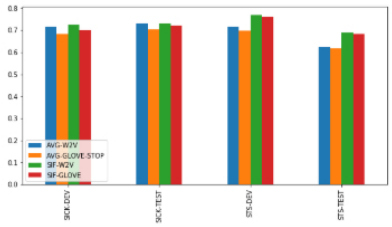
Based on our results, there's little reason to use Word Mover's Distance rather than simple word2vec averages. Only on STS-TEST, and only in combination with a stoplist, can WMD compete with the simpler baselines.



Word Mover's Distance disappoints.

Smooth Inverse Frequency

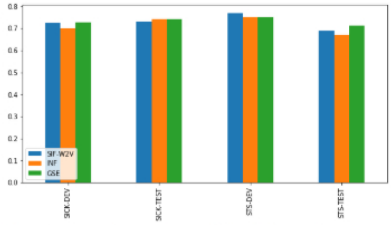
Smooth Inverse Frequency is the most consistent performer in our tests. On the SICK data, it does about as well as its baseline competitors, on STS it outranks them by a clear margin. Note there is little difference between SIF with word2vec embeddings and SIF with GloVe embeddings. This is remarkable, given the large differences between the two we observed above. It shows SIF's weighting and common component removal effectively reduces uninformative noise from the embeddings.



Smooth Inverse Frequency is the most consistent performer.

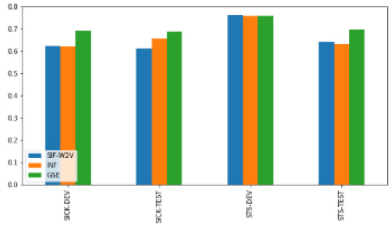
Pre-trained encoders

Pre-trained encoders have a lot to be said for them. However, our results indicate they are not yet able to capitalize fully on their training regime. Google's Sentence Encoder looks like a better choice than InferSent, but the Pearson correlation coefficient shows very little difference with Smooth Inverse Frequency.



Pre-trained encoders perform well, but SIF gives them a run for their money.

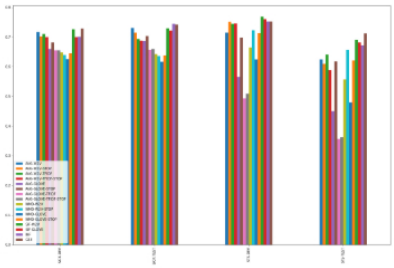
The differences in Spearman correlation are more outspoken. This may indicate that the Google Sentence Encoder more often gets the ordering of the sentences right, but not necessarily the relative differences between them.



Spearman correlation shows a larger difference than Pearson correlation.

Conclusions

Sentence similarity is a complex phenomenon. The meaning of a sentence does not only depend on the words in it, but also on the way they are combined. As the harp-keyboard example above shows, semantic similarity can have several dimensions, and sentences may be similar in one but opposite in the other. Current sentence embedding methods only scratch the surface of what's possible.



The combined results of all sentence similarity methods.

So, what should you do when you're looking to compute sentence similarities? Our results suggest the following:

- Word2vec embeddings are a safer choice than GloVe embeddings.
- Although an unweighted average of the word embeddings in the sentence holds its own as a simple baseline, Smooth Inverse Frequency is usually a stronger alternative.
- If you can use a pre-trained encoder, pick Google's Sentence Encoder, but remember its performance gain may not be all that spectacular.

Yves Peirsman

Yves discovered Natural Language Processing 13 years ago as an MSc student at the University of Edinburgh, and has never looked back. With a background as a researcher and developer in academia (University of Leuven, Stanford University) and industry (Textkernel, Wolters Kluwer), he founded NLP Town to further indulge and spread his love for NLP.

Leonid Boytsov • 2 years ago

Hi, thanks a lot for your work! Did you consider evaluating ELMBO embeddings: http://allennlp.org/elmo? There's been a lot of buzz about them recently: So, we all very curious how they'll perform in your tests.

Many thanks!

2 | 1 • Reply • Share ›

Christian ➚ Leonid Boytsov • 2 years ago

Hi Leonid, we did an extensive evaluation, the paper is out today at https://arxiv.org/abs/1806.... if you're interested.

2 | • Reply • Share ›

Leonid Boytsov ➚ Christian • 2 years ago

awesome, but I nearly missed your comment, b/c of the delay :-)

1 | • Reply • Share ›

encgoo encgoo • 7 months ago

How about Gensim's Doc2vec and Sentence2vec please?

1 | • Reply • Share ›

Moinak Bhattacharya • a year ago

Hi, Thanks for your post, I am trying to run your code - turns out that following URLs are not found,
wget -nc https://raw.githubuserconte...
wget -nc https://s3.amazonaws.com/se...
please let me know where can I find these files alternatively.

1 | • Reply • Share ›

This comment was deleted.

Yves Peirsman Mod ➚ Guest • 2 years ago

Thanks for your feedback! I've added the files to the repository and updated the path in the notebook.

| • Reply • Share ›

Manoj Agarwal • 8 months ago

Hello Sir,
Thanks a lot for this wonderful discussion , i am not able to get word frequency and document frequency tsv , can you please help me how and from where i can have that file

| • Reply • Share ›

Rishi Jain • 10 months ago

Hi team, I have a situation and unable to find the approach and trained data. The problem is that - I would like to give a SCORE to sentences on different classes like Innovation, Collaboration, Agility. Subsequently would like to arrive to an average score. How can I compare sentences to a token (class).
Are there any pre-trained models where I can get an aspect from sentence?
What is the best way to use Context as there would be many related words like Synery / Cooperation/Partnering which are close to Innovation - I got them from Gensim library.

Any help on this is highly appreciated!

| • Reply • Share ›

Yassin Belhareth • a year ago

Hello , thanks for your work. how can i reference it in my paper ?

| • Reply • Share ›

Chirag Jain • 2 years ago

Thank you for this benchmark work! I recently compared w2v_avg, sif and universal sentence encoder on my own datasets that I work with and have found very similar relative results. Although for my datasets removing stop words helped a lot for w2v_avg considering the mean vector can be very sensitive to such words.

| • Reply • Share ›

Abdullah Kiwan • 2 years ago

Hello ... Thanks a lot for this article ...
I am trying to run the code, but I could not fine the files "frequencies.tsv" and "doc_frequencies.tsv"... would you please tell me where I can find them ?

Thanks in advance :)

^ | v • Reply • Share ›

Yves Peirsman Mod ➔ Abdullah Kiwan • 2 years ago

Hi Abdullah, I've added the files to the repository and updated the path in the notebook.

^ | v • Reply • Share ›

You might also enjoy

(View all posts)

- Why computers don't yet read better than us
- Named Entity Recognition and the Road to Deep Learning
- Perplexed by Game of Thrones. A Song of N-Grams and Language Models