Different Approaches to Find Semantic Textual Similarity (STS)

SUBHADIP MAITY

28.05.2021

Introduction

Semantic textual similarity deals with determining how similar two pieces of texts are. Here, we quantify the similarity between a pair of texts of text1 and text2 columns of the given data set for each row with 5 different approaches. We represent the similarity score between a pair of texts with a number lying between 0 and 1 where 0 means no relation and 1 means high relation.

Pre-processing

After loading and inspecting the data set, we cleaned the data set through some preprocessing techniques. As well as lower casing and tokenizing, we removed stopwords and stemmed each token using the NLTK library. We also removed all mentions, hyperlinks, and characters other than alphanumeric.

Approaches

We used the following methodologies:

- Count Vectorizer method
- Pre-trained Word2Vec embedding
- Own Doc2vec embedding
- Bert model
- Transformer model

The main idea behind all the approaches is to get the vector representation of a whole text\paragraph and then use cosine similarity between such a pair of vectors to find the similarity between them.

Vector Representation

For Count Vectorizer Method, we used the Bag of Words technique for the vector representation of a paragraph. To implement this Sklearn's CountVectorizer has been used. In the case of the pre-trained Word2vec embedding, we used google news vectors which provide the semantic vector representation of each word/token with the Gensim model. To get the vector representation of the whole paragraph we simply average all the word vectors using the Spacy library. This technique also resolves Out of Vocabulary problem. For Own Doc2vec embedding, we simply create the vector representation of a whole document using Gensim's Doc2Vec model. Since it is trained on our own corpus, it does not suffer from OOV issues. To implement the Bert model we used WebBertSimilarity of Semantic-text-similarity package. We have used SentenceTransformers, a simple library that provides an easy method to calculate dense vector representations (e.g. embeddings) for texts. It contains many state-of-the-art pre-trained models that are fine-tuned for various applications. One of the primary tasks that it supports is Semantic Textual Similarity.

Cosine similarity

Vector Space Model(VSM) is a mathematical model that represents text units as vectors. We already discussed the VSM models that we used in this problem. In this setting, the most common way to calculate the similarity between two text blobs is using cosine similarity: the cosine of the angle between their corresponding vectors. The cosine of 0° is 1 and the cosine of 180° is –1, with the cosine monotonically decreasing from 0° to 180°. Given two vectors, A and B, each with n components, the similarity between them is computed as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} \mathrm{A}_i^2} \sqrt{\sum_{i=1}^{n} \mathrm{B}_i^2}}$$

where A_i and B_i are the ith components of vectors A and B, respectively. Sometimes, people also use Euclidean distance between vectors to capture similarity. To calculate cosine distance we have used Scipy's spatial.distance.cosine.

I.e. cosine distance =scipy.spatial.distance.cosine
and cosine similarity=1- cosine distance.

Transformation

Since We want our final similarity score in [0,1] scale we have to transform our obtained similarity score to this scale. The primary output of the Bert model's similarity scores is on the scale of [0,5]. All the other methods' similarity score lies in-between -1 to +1. For transforming similarity score to [0,1] scale, we use the transformation formulae

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

Where x_i is the ith observation in the old scale and z_i is the new value of x_i in the new scale. x denotes the previous scale.

Conclusion

Since the Count Vectorization does not retain any semantic information, we can not expect a good result from it. Although pre-trained Word2Vec and Doc2Vec keep semantic meanings, averaging word vectors to get the representation of a whole paragraph might lose semantic information and hence perform very poorly. We can expect a better result from the Bert and Transformer model but they are time-consuming and need more computational resources.