Project Report

(By: Rashid Ali)

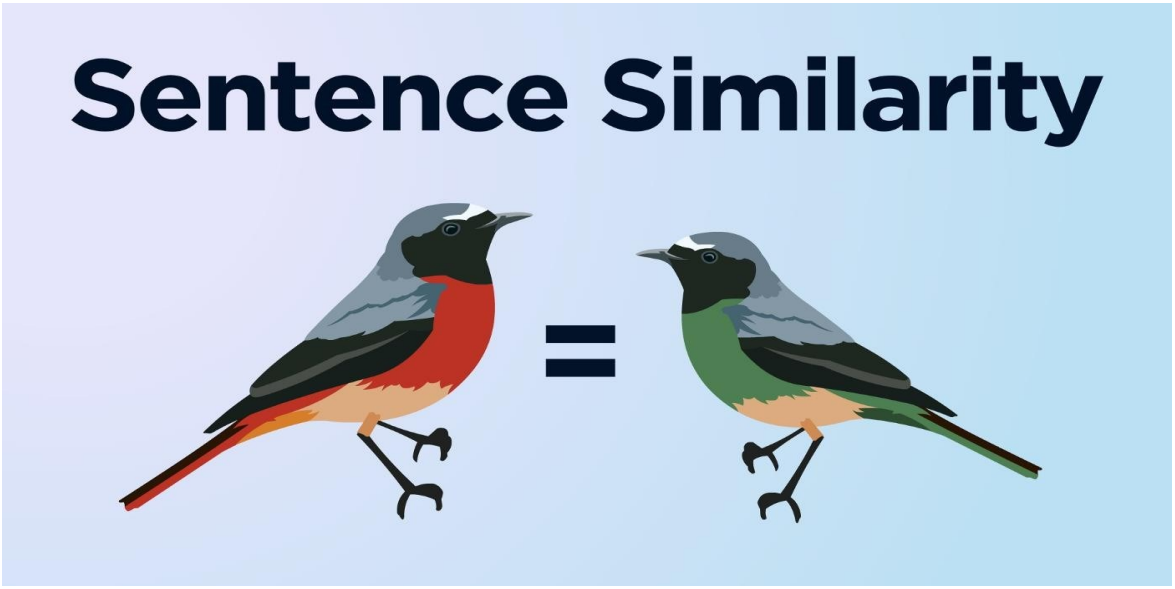
Objective :

- The data contains a pair of paragraphs. These text paragraphs are randomly sampled from a raw dataset. Each pair of the sentence may or may not be semantically similar. The candidate is to predict a value between 0-1 indicating a degree of similarity between the pair of text paras. 0 means highly similar 1 means highly dissimilar.
- STS is the assessment of pairs of sentences according to their degree of semantic similarity. The task involves producing real-valued similarity scores for sentence pairs.

Approach : Measuring text similarity using BERT

Introduction :

High-performance semantic similarity with BERT



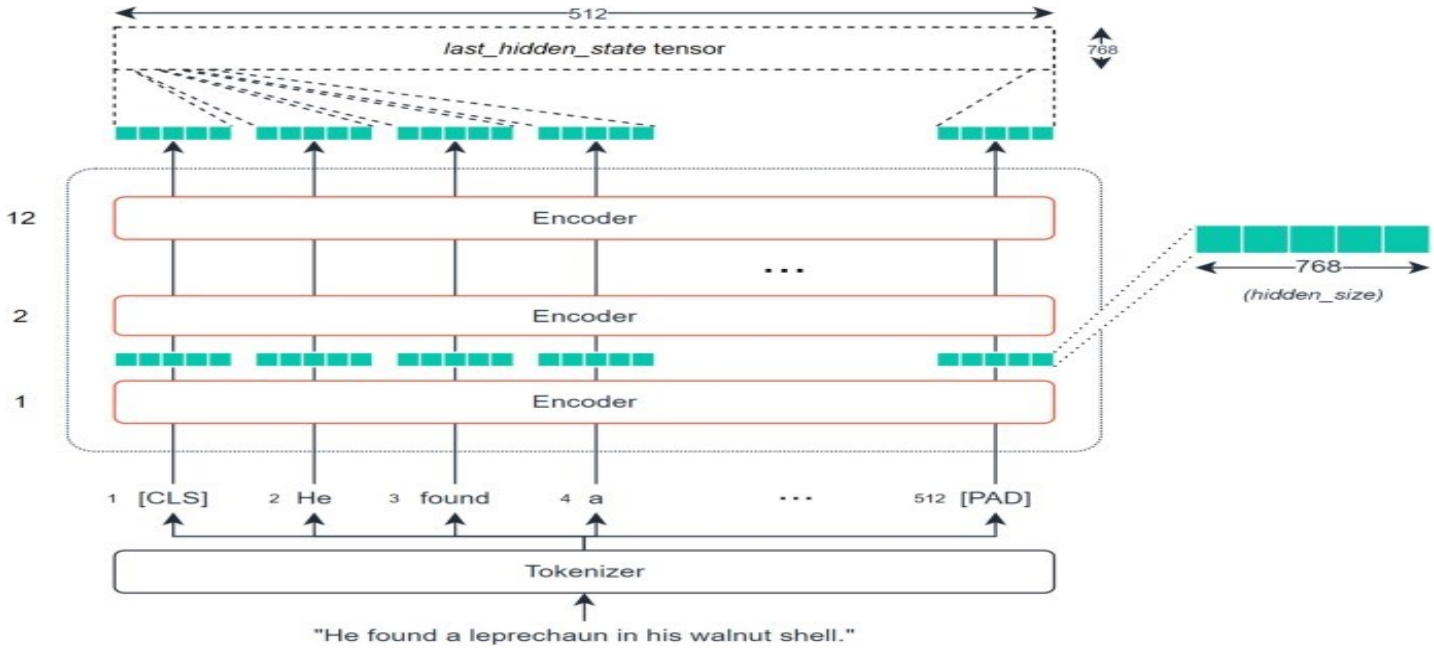
BERT is the MVP of NLP. And a big part of this is down to BERTs ability to embed the meaning of words into densely packed vectors.

We call them dense vectors because every value within the vector has a value and has a reason for being that value — this is in contrast to sparse vectors, such as one-hot encoded vectors where the majority of values are 0.

BERT is great at creating these dense vectors, and each encoder layer (there are several) outputs a set of dense vectors.

For BERT base, this will be a vector containing 768. Those 768 values contain our numerical representation of a single token — which we can use as contextual word embeddings.

Because there is one of these vectors for representing each token (output by each encoder), we are actually looking at a tensor of size 768 by the number of TOKENS



We can take these tensors — and transform them to create semantic representations of the input sequence. We can then take our similarity metrics and calculate the respective similarity between different sequences.

The simplest and most commonly extracted tensor is the LAST_HIDDEN_STATE tensor — which is conveniently output by the BERT model.

Of course, this is a pretty large tensor — at 512x768 — and we want a vector to apply our similarity measures to it.

To do this, we need to convert our LAST_HIDDEN_STATES tensor to a vector of 768 dimensions.

Creating The Vector :

For us to convert our LAST_HIDDEN_STATES tensor into our vector — we use a mean pooling operation.

Each of those 512 tokens has a respective 768 values. This pooling operation will take the mean of all token embeddings and compress them into a single 768 vector space — creating a 'sentence vector'.

At the same time, we can't just take the mean activation as is. We need to consider null padding tokens (which we should not include).

Easy — Sentence-Transformers

The easiest approach for us to implement everything we just covered is through the sentence-transformers library — which wraps most of this process into a few lines of code.

First, we install sentence-transformers using pip install sentence-transformers. This library uses HuggingFace's transformer behind the scenes .

```
In [4]: df.head()
```

Out[4]:

	text1	text2
0	broadband challenges tv viewing the number of ...	gardener wins double in glasgow britain s jaso...
1	rap boss arrested over drug find rap mogul mar...	amnesty chief laments war failure the lack of ...
2	player burn-out worries robinson england coach...	hanks greeted at wintry premiere hollywood sta...
3	hearts of oak 3-2 cotonsport hearts of oak set...	redford s vision of sundance despite sporting ...
4	sir paul rocks super bowl crowds sir paul mcca...	mauresmo opens with victory in la amelie maure...

Similarity of first 20 records

```
In [5]: text1 = df['text1']
        text2 = df['text2']

In [6]: model_name = "bert-base-nli-mean-tokens"

In [7]: model = SentenceTransformer(model_name)

In [8]: from sklearn.metrics.pairwise import cosine_similarity

In [9]: sample1 = text1[:20].values
        sample2 = text2[:20].values

In [10]: sentence_vecs1 = model.encode(sample1)
         sentence_vecs2 = model.encode(sample2)

In [32]: sentence_vecs1.shape
Out[32]: (20, 768)
```

We'll be making use of the bert-base-nli-mean-tokens model — which implements the same logic we've discussed so far.

Great, we now have four sentence embeddings — each containing 768 values.

Now what we do is take those embeddings and find the cosine similarity between each. So for sentence 0:

We can find the most similar sentence using:

In [33]:
```
df_new.head()
```

Out[33]:

	text1	text2	Similarity	Similarity After Processing
0	broadband challenges tv viewing the number of ...	gardener wins double in glasgow britain s jaso...	0.623987	0.755719
1	rap boss arrested over drug find rap mogul mar...	amnesty chief laments war failure the lack of ...	0.647651	0.728845
2	player burn-out worries robinson england coach...	hanks greeted at wintry premiere hollywood sta...	0.574893	0.654460
3	hearts of oak 3-2 cotonsport hearts of oak set...	redford s vision of sundance despite sporting ...	0.550737	0.687529
4	sir paul rocks super bowl crowds sir paul mcca...	mauresmo opens with victory in la amelie maure...	0.645250	0.746392

Similarity of Input define Strings

In [35]:
```
sentence_vecs3 = model.encode(input("Enter the first text1 : "))
sentence_vecs4 = model.encode(input("Enter the first text2 : "))
cosine_similarity([sentence_vecs3],[sentence_vecs4])[0][0]
```

```
Enter the first text1 : My name is Rashid Ali !
Enter the first text2 : my name is ali Rashid
```

Out[35]:
```
0.9240347
```

Similarity After preprocessing the Text data

In [16]:
```
from nltk.corpus import stopwords
```

In [17]:
```
stp = stopwords.words('english')
```

This is the easier — more abstract approach. Seven lines of code to compare our sentences.

END OF REPORT