Vidyavardhini's College of Engineering and Technology, Vasai

Department of Computer Science & Engineering (Data Science)

AY: 2025-26

Class:	**BE-CSE(DS)**	**Semester:**	**VII**
Course Code:	**CSDOL7011**	**Course Name:**	**NLP Lab**

Name of Student:	**Hitesh Shetye**
Roll No. :	**49**
Experiment No.:	**8**
Title of the Experiment:	**Measuring Semantic Similarity Between Sentences using Sentence Transformers**
Date of Performance:	
Date of Submission:	

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty :

Signature :

Date :

Aim: To compute the semantic similarity between sentence pairs using pre-trained sentence embedding models from the Sentence Transformers library.

Objective: • To measure sentence-level semantic similarity using pretrained sentence transformer models

Tools Required:

1. Python (Jupyter Notebook or Google Colab)
2. sentence-transformers library
3. scikit-learn (for cosine similarity)
4. numpy
5. Install Sentence Transformers (if not already installed):
    a. pip install -U sentence-transformers

Procedure:

1. Import required libraries:
    a. from sentence_transformers import SentenceTransformer
    b. from sklearn.metrics.pairwise import cosine_similarity
    c. import numpy as np

2. Load a pre-trained model:
    a. model = SentenceTransformer('all-MiniLM-L6-v2')

3. Define two or more sentences to compare:
    a. sentences = [
    b. "A man is playing a guitar.",
    c. "A person is playing a musical instrument."
    d.]

4. Generate embeddings:
    a. embeddings = model.encode(sentences)

5. Compute cosine similarity:
    a. similarity = cosine_similarity([embeddings[0]], [embeddings[1]])
    b. print(f"Semantic Similarity Score: {similarity[0][0]:.4f}")

6. Experiment with unrelated sentence pairs and observe similarity values.

Description of the Experiment:

In this experiment, students explore how sentence-level semantic similarity is measured using transformer-based sentence embeddings. By comparing similar and dissimilar sentences, they gain an intuitive understanding of how meaning—not just surface words—affects similarity scores.

Detailed Description of the NLP Technique:

1. Sentence Embeddings:

Sentence embeddings are fixed-length dense vector representations of entire sentences. Unlike word embeddings (e.g., Word2Vec), these models capture the semantic meaning of full sentences.

2. Sentence Transformers:

Built on top of BERT or RoBERTa, the Sentence Transformers framework fine-tunes models to produce high-quality sentence embeddings suitable for:

Semantic textual similarity

Clustering

Semantic search

Question-answer retrieval

The all-MiniLM-L6-v2 model used here is a compact and fast model ideal for educational use.

3. Cosine Similarity:

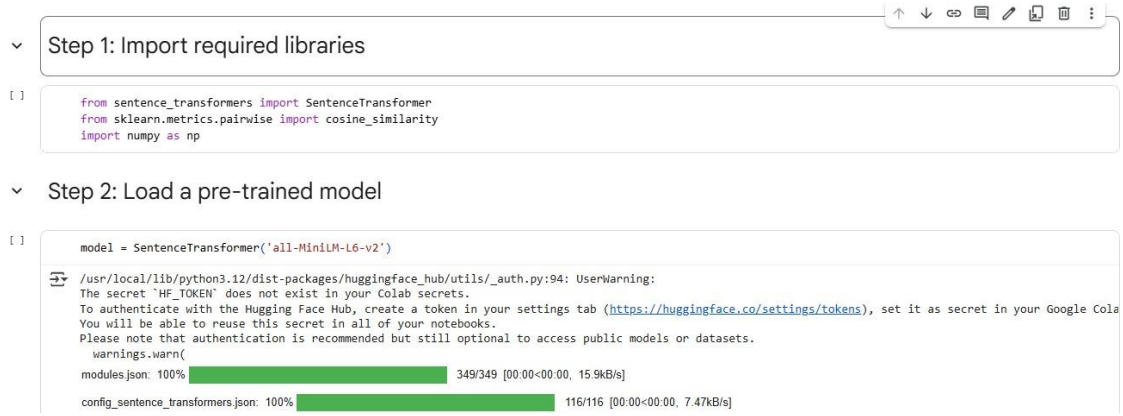
Measures the cosine of the angle between two vectors. Closer to 1 means more semantically similar:

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\|\|B\|}$$

Why Use Sentence Embeddings:

- Capture context and meaning rather than individual words.
- Robust to word order changes and synonyms.
- Highly effective in tasks requiring semantic understanding.

OUTPUT:



```
README.md: 10.5k/? [00:00<00:00, 598kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 5.72kB/s]
config.json: 100% 612/612 [00:00<00:00, 66.1kB/s]
model.safetensors: 100% 90.9M/90.9M [00:01<00:00, 117MB/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 19.4kB/s]
vocab.txt: 232k/? [00:00<00:00, 5.88MB/s]
tokenizer.json: 466k/? [00:00<00:00, 15.4MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 7.25kB/s]
config.json: 100% 190/190 [00:00<00:00, 11.2kB/s]
```

Step 3: Define two or more sentences to compare

```
sentences = [
    "A man is playing a guitar.",
    "A person is playing a musical instrument."
]
```

Step 4: Generate embeddings

[4] ✓ 0s

```
embeddings = model.encode(sentences)
embeddings
```

```
array([[ 2.16640569e-02, -4.86421026e-03, -9.64910816e-03,
        -6.34012297e-02, -1.67493492e-01,  4.67641698e-03,
         8.05998072e-02, -2.41421219e-02, -1.26949102e-02,
         6.05556592e-02, -4.11286904e-03,  1.71893165e-02,
        -3.06097697e-02, -1.94309396e-03,  4.86733094e-02,
         1.59955362e-03, -1.67813781e-03,  3.90221067e-02,
         7.58418888e-02,  1.56994965e-02,  6.18278645e-02,
         1.22607745e-01, -7.43683726e-02, -5.09945489e-02,
        -4.01344784e-02, -2.48654536e-03, -1.35530462e-03,
         1.46005943e-01,  6.17187796e-03, -3.03680031e-03,
         9.34898406e-02,  2.72589549e-02,  8.23174417e-03,
        -7.93025270e-02, -1.28897890e-01, -2.72006206e-02,
        -1.33324578e-01, -3.12699974e-02, -8.56232271e-03,
        -3.82236987e-02,  5.21247229e-03, -2.37680245e-02,
         2.79553384e-02,  9.00139511e-02, -3.45801190e-02,
        -2.95070652e-02, -4.74883281e-02, -1.46947084e-02,
         6.88855909e-03, -5.23216017e-02, -3.58388619e-03,
         5.12124822e-02,  7.11305588e-02, -6.68859184e-02,
         4.27106395e-02,  1.81780141e-02, -1.62300728e-02,
```

Step 5: Compute cosine similarity

[5] ✓ 0s

```
similarity = cosine_similarity([embeddings[0]], [embeddings[1]])
print(f"Semantic Similarity Score: {similarity[0][0]:.4f}")
```

```
Semantic Similarity Score: 0.6920
```

Step 6: Experiment with unrelated pairs

[6] ✓ 0s

```
unrelated_sentences = [
    "The sun is shining brightly today.",
    "A man is playing a guitar."
]

emb_unrelated = model.encode(unrelated_sentences)
similarity_unrelated = cosine_similarity([emb_unrelated[0]], [emb_unrelated[1]])
print(f"Unrelated Sentence Similarity Score: {similarity_unrelated[0][0]:.4f}")
```

```
Unrelated Sentence Similarity Score: -0.0073
```

[7]

```
# Model correctly detects no relation.
```

Conclusion:

- Sentence Transformers effectively understand sentence-level semantics, going beyond word-level matching methods.
- They show strong performance in identifying similar, rephrased, or completely different sentences.
- Such capability makes them valuable for real-world tasks like semantic search, duplicate detection, QA systems, and advanced information retrieval.