

Program4: Measure similarity between texts. Use vectorization techniques (e.g., TF-IDF, word embeddings). i. Implement cosine similarity ii. Compare the effectiveness of different similarity measures

Objective:

To measure the similarity between texts using different vectorization techniques (TF-IDF and word embeddings) and implement cosine similarity to compare the effectiveness of different similarity measures.

Approach:

Text Vectorization Techniques: TF-IDF (Term Frequency-Inverse Document Frequency): Converts text into numerical features based on term importance.

Word Embeddings (Word2Vec, GloVe, or BERT): Captures semantic meaning by mapping words into high-dimensional vector space.

Similarity Measures:

Cosine Similarity: Measures the cosine of the angle between two vectors (ranges from 0 to 1, where 1 means identical).

Euclidean Distance: Measures the absolute difference between vector points in space.

Jaccard Similarity: Measures the intersection over the union of word sets.

TF-IDF is a numerical statistic used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It is widely used in information retrieval, search engines, and text similarity tasks to represent text as numerical vectors.

TF-IDF consists of two main parts:

a) Term Frequency (TF):

TF measures how frequently a word appears in a document. The assumption is that words appearing frequently in a document are important.

B) Inverse Document Frequency (IDF):

IDF measures the importance of a word across all documents. Common words (e.g., "the", "is", "and") have high TF but are not informative, so IDF penalizes them.

$$TF(w) = \frac{\text{Number of times the word } w \text{ appears in the document}}{\text{Total number of words in the document}}$$

Suppose a document contains 100 words, and the word "machine" appears 5 times.

$$TF(\text{"machine"}) = \frac{5}{100} = 0.05$$

$$IDF(w) = \log \left(\frac{\text{Total number of documents}}{\text{Number of documents containing the word } w} \right)$$

If a word appears in all documents, IDF becomes low (close to 0), reducing its importance.



If a word appears in very few documents, IDF is high, increasing its weight. Example:

Suppose we have 10,000 documents, and "machine" appears in 100 documents

$$IDF(\text{"machine"}) = \log\left(\frac{10,000}{100}\right) = \log(100) = 2$$

TF-IDF Score Calculation:

$$TF - IDF(w) = TF(w) \times IDF(w)$$

Example:

If the TF of "machine" is 0.05 and its IDF is 2, then

$$TF - IDF(\text{"machine"}) = 0.05 \times 2 = 0.1$$

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Sample texts
text1 = "Machine learning is a field of artificial intelligence."
text2 = "Deep learning is a branch of artificial intelligence and machine learning."

# Convert texts to TF-IDF vectors
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform([text1, text2])

# Compute cosine similarity
cosine_sim = cosine_similarity(tfidf_matrix[0], tfidf_matrix[1])

print(f"Cosine Similarity (TF-IDF): {cosine_sim[0][0]:.4f}")
```

🔗 Cosine Similarity (TF-IDF): 0.6416

TF-IDF transforms text into numerical vectors.

Cosine similarity is computed as:

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

A higher value means the texts are more similar.

Advantages of TF-IDF

- ✅ Handles common vs. rare words well – Reduces the importance of words like "the" and "is".
- ✅ Simple and efficient – Works well for search engines and document classification.
- ✅ Effective for keyword-based similarity – Helps in document ranking, topic modeling, and text clustering.

Implementing Cosine Similarity using Word Embeddings (Word2Vec - Pretrained GloVe vectors)

```
import numpy as np
import gensim.downloader as api
from sklearn.metrics.pairwise import cosine_similarity
```



```
# Load pretrained GloVe model
glove_model = api.load("glove-wiki-gigaword-50")

def get_embedding(text, model):
    words = text.lower().split()
    word_vectors = [model[word] for word in words if word in model]


    if not word_vectors:
        return np.zeros(model.vector_size)

    return np.mean(word_vectors, axis=0)

# Compute embeddings
embedding1 = get_embedding(text1, glove_model)
embedding2 = get_embedding(text2, glove_model)

# Compute cosine similarity
cosine_sim = cosine_similarity([embedding1], [embedding2])

print(f"Cosine Similarity (Word Embeddings - GloVe): {cosine_sim[0][0]:.4f}")
```

 [=====] 100.0% 66.0/66.0MB downloaded
Cosine Similarity (Word Embeddings - GloVe): 0.9705

- Pretrained **GloVe** vectors capture word meanings.
- Each text is represented as an **average of word vectors**.
- Cosine similarity is computed on these dense representations.

TF-IDF works well for keyword-based similarity but lacks meaning.

Word Embeddings capture semantics better, making them more effective for NLP tasks like sentiment analysis, search, and recommendation systems.

When to Use TF-IDF?

✅ Best for traditional NLP tasks like:

Information retrieval (search engines, document ranking)

Keyword-based text similarity (plagiarism detection, news categorization)

Text classification with small datasets

When to Use Word Embeddings?

✅ Best for deep NLP applications like:

Chatbots and virtual assistants (Google Assistant, Siri)

Machine Translation (Google Translate)

Sentiment analysis (understanding emotions in reviews, social media)

Text summarization, question-answering systems



