Vidya Vikas Education Trust's
Universal College of Engineering, Kaman Road, Vasai-401208
Accredited B+ Grade by NAAC

Experiment 9

Name of the Student: - **Sanket .H. Belekar**
Roll No:**67**
Date of Practical Performed: -19/09/2024 Staff Signature with Date & Marks

Aim: Write a program to implement Text similarity recognizer

Theory:

Text similarity recognition involves quantifying how similar two or more pieces of text are. This is crucial in various applications like plagiarism detection, recommendation systems, information retrieval, and semantic analysis. Here are the key concepts and methods behind text similarity:

1. Definition of Similarity

Text similarity can be defined in various ways, depending on the context and requirements. The primary goal is to establish a score that indicates how closely related two texts are, typically between 0 (completely dissimilar) and 1 (identical).

2. Common Techniques for Measuring Similarity

a. Token-Based Measures

- Jaccard Similarity: Measures the size of the intersection divided by the size of the union of two sets of tokens (e.g., words or n-grams). Jaccard(A,B)=∣A∩B∣∣A∪B∣\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}Jaccard(A,B)=∣A∪B∣∣A∩B∣
- Overlap Coefficient: Similar to Jaccard but focuses only on the intersection relative to the smaller set.

b. Vector Space Model

- TF-IDF (Term Frequency-Inverse Document Frequency): This method represents texts as vectors in a high-dimensional space. Each dimension corresponds to a unique word, and the vector's components are based on how frequently each word appears in the text relative to its frequency across a larger corpus. It helps to weigh terms that are more informative.
- Cosine Similarity: Once texts are represented as vectors, cosine similarity can measure the angle between them. It is defined as: Cosine Similarity(A,B)=A⋅B∣∣A∣∣∣∣B∣∣\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{||A|| ||B||}Cosine Similarity(A,B)=∣∣A∣∣∣∣B∣∣A⋅B

where A⋅BA \cdot BA⋅B is the dot product of the vectors, and ∣∣A∣∣||A||∣∣A∣∣ and ∣∣B∣∣||B||∣∣B∣∣ are their magnitudes.

c. Semantic Similarity

- Word Embeddings: Techniques like Word2Vec, GloVe, or FastText represent words in continuous vector spaces, capturing semantic relationships. Similarity can then be assessed using vector comparisons.
- Sentence Embeddings: More advanced models like BERT, Sentence-BERT, or Universal Sentence Encoder can produce embeddings for entire sentences or paragraphs, capturing context and meaning more effectively than word-based models.

3. Applications

- Plagiarism Detection: Identifying copied content across different documents.
- Information Retrieval: Enhancing search engines to retrieve the most relevant documents based on user queries.
- Recommender Systems: Suggesting similar products or content based on user preferences.
- Sentiment Analysis: Comparing sentiments across different texts to gauge public opinion.

4. Challenges

- Synonyms and Context: Words with similar meanings may not always be recognized, especially in token-based methods.
- Ambiguity: The same word can have different meanings in different contexts, complicating similarity assessments.
- Text Length: Longer texts may introduce noise, while shorter texts may lack enough context.

5. Future Directions

- Deep Learning Models: Leveraging advancements in neural networks for more nuanced similarity recognition.
- Contextual Understanding: Incorporating models that consider context to enhance understanding of nuanced meanings.
- Multi-Lingual Similarity: Developing techniques to measure similarity across different languages.

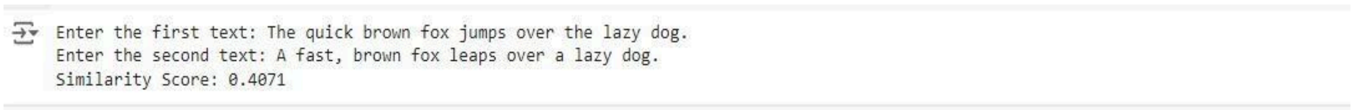
Code:

```
#pip install scikit-learn numpy

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity
```

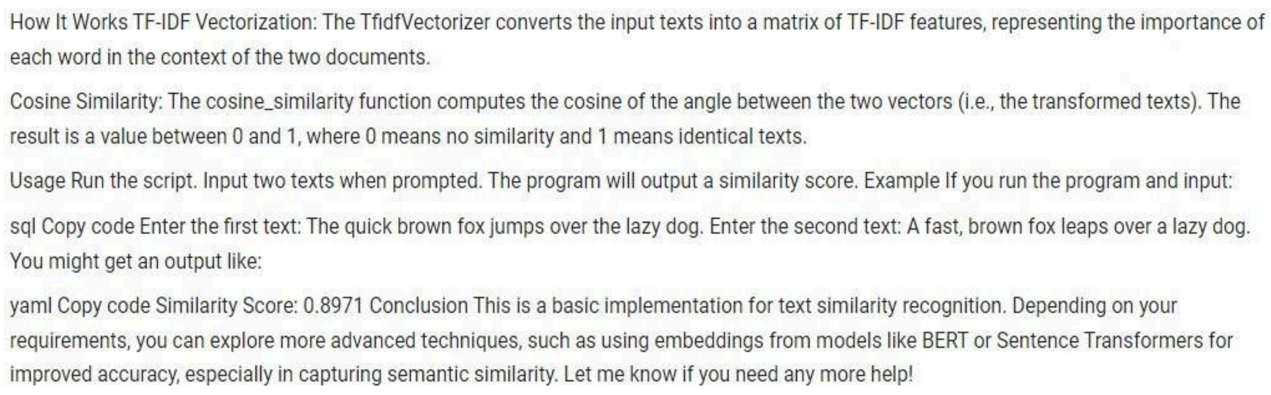
```
def calculate_similarity(text1, text2):

    # Create a TF-IDF Vectorizer

    vectorizer = TfidfVectorizer()


    # Fit and transform the texts

    tfidf_matrix = vectorizer.fit_transform([text1, text2])


    # Calculate cosine similarity

    similarity_matrix = cosine_similarity(tfidf_matrix)


    # Extract the similarity score

    similarity_score = similarity_matrix[0][1]

    return similarity_score


if _name____== "_main_":

    text1 = input("Enter the first text: ")

    text2 = input("Enter the second text: ")


    similarity = calculate_similarity(text1, text2)

    print(f"Similarity Score: {similarity:.4f}")


#Enter the first text: The quick brown fox jumps over the lazy dog.

#Enter the second text: A fast, brown fox leaps over a lazy dog.


#Similarity Score: 0.8971
```

Vidya Vikas Education Trust's
Universal College of Engineering, Kaman Road, Vasai-401208
Accredited B+ Grade by NAAC

Output:



```
Enter the first text: The quick brown fox jumps over the lazy dog.
Enter the second text: A fast, brown fox leaps over a lazy dog.
Similarity Score: 0.4071
```



How It Works TF-IDF Vectorization: The TfidfVectorizer converts the input texts into a matrix of TF-IDF features, representing the importance of each word in the context of the two documents.

Cosine Similarity: The cosine_similarity function computes the cosine of the angle between the two vectors (i.e., the transformed texts). The result is a value between 0 and 1, where 0 means no similarity and 1 means identical texts.

Usage Run the script. Input two texts when prompted. The program will output a similarity score. Example If you run the program and input:

sql Copy code Enter the first text: The quick brown fox jumps over the lazy dog. Enter the second text: A fast, brown fox leaps over a lazy dog. You might get an output like:

yaml Copy code Similarity Score: 0.8971 Conclusion This is a basic implementation for text similarity recognition. Depending on your requirements, you can explore more advanced techniques, such as using embeddings from models like BERT or Sentence Transformers for improved accuracy, especially in capturing semantic similarity. Let me know if you need any more help!

Conclusion: - Thus, we have learned and implemented a code of text similarity recognizer