Experiment No. 8
Implement Text Similarity Recognizer for the chosen text documents.
Date of Performance:
Date of Submission:

Aim: Implement Text Similarity Recognizer for the chosen text documents.

Objective: Understand the importance of Implementing Text Similarity Recognizer for the chosen text documents.

Theory:

1. **Preprocess the Text Data:**
    - Tokenization
    - Stopwords removal
    - Lemmatization or stemming
2. **Feature Extraction:**
    - TF-IDF Vectorization
    - Word Embeddings (e.g., Word2Vec, GloVe)
    - Sentence Embeddings (e.g., Sentence-BERT)
3. **Compute Similarity:**
    - Cosine Similarity
    - Euclidean Distance
4. **Evaluate Similarity:**
    - Compare and interpret the similarity scores.

Implementation:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
sentenceOne = 'My house is empty today'
sentenceTwo = 'Nobody is at my home'
documents = [sentenceOne, sentenceTwo]
tfidf = TfidfVectorizer ()
sparseMatrix = tfidf.fit_transform (documents)
docTermMatrix = sparseMatrix.todense ()
```

```
df = pd.DataFrame (
    docTermMatrix,
    columns = tfidf.get_feature_names_out (),
    index = ['sentenceOne', 'sentenceTwo']
)
simScore = cosine_similarity (df, df)[0, 1]
match_keys = df.isin ([0]).sum (axis = 0)
match_words = match_keys[match_keys.values == 0].keys ()
print (f'Cosine Similarity : {round (simScore, 2)}')
print (f'Matching Words : {list (match_words)}')
```

```
Cosine Similarity : 0.25
Matching Words : ['is', 'my']
```

Conclusion:

Implementing a Text Similarity Recognizer involves several crucial steps: preprocessing the text data, extracting meaningful features, computing similarity measures, and interpreting the results. In this example, we utilized TF-IDF Vectorization for feature extraction and Cosine Similarity for computing similarity between text documents.