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

Objective:

To study and write program for text similarity recognition

Theory:

Text Similarity is the process of comparing a piece of text with another and finding the similarity between them. It's basically about determining the degree of closeness of the text. Dealing with text, sentences or words brings us to the region of Natural Language Processing (NLP), where we are going to use different NLP approaches to process the raw text and help the model to detect the similarity more swiftly and efficiently. Text similarity is needed for following reasons,

- Search engines need to model the relevance of a document to a query, beyond the overlap in words between the two. For instance, question-and-answer sites such as Quora or Stack Overflow need to determine whether a question has already been asked before.
- Selecting the most similar product for a customer shopping in any online platform if that exact product is unavailable.
- Checking similarity of multiple documents or letters.
- Choosing the most appropriate or closest job role or profile a person's resume.

Program:

```
import numpy as np  from sklearn.feature_extraction.text

import TfidfVectorizer  from sklearn.metrics.pairwise

import cosine_similarity  from nltk.tokenize import

word_tokenize  from nltk.metrics import edit_distance


text1 = "I was sitting by the river"

text2 = "I was standing by the lake"


# Jaccard Similarity  def

jaccard_similarity(str1, str2):  a

=
```

```
set(word_tokenize(str1.lower()))
b =
set(word_tokenize(str2.lower()))
c = a.intersection(b)  return
float(len(c)) / (len(a) + len(b) -
len(c))

 jaccard_sim = jaccard_similarity(text1, text2)

 # Cosine Similarity  vectorizer = TfidfVectorizer()
tfidf = vectorizer.fit_transform([text1, text2])
cosine_sim = cosine_similarity(tfidf[0], tfidf[1])[0][0]

 # Levenshtein similarity  distance =
edit_distance(text1, text2)  max_len =
max(len(text1), len(text2))  levenshtein_sim =
(max_len - distance) / max_len

 print("Cosine Similarity:", cosine_sim)
print("Jaccard Similarity:", jaccard_sim)
print("Levenshtein Similarity:", levenshtein_sim)
```

Conclusion:

The implementation of a Text Similarity Recognizer is instrumental in streamlining document management, upholding academic integrity, enhancing user recommendations, improving information retrieval, and facilitating advanced content analysis. By integrating this technology into the natural language processing pipeline, researchers and practitioners can leverage its capabilities to gain deeper insights and make more informed decisions across various domains.