

# **EX6 - Information Retrieval Using Vector Space Model in Python**

## **DATE:**

## **AIM:**

To implement Information Retrieval Using Vector Space Model in Python.

## **Description:**

Implementing Information Retrieval using the Vector Space Model in Python involves several steps, including preprocessing text data, constructing a term-document matrix, calculating TF-IDF scores, and performing similarity calculations between queries and documents. Below is a basic example using Python and libraries like nltk and sklearn to demonstrate Information Retrieval using the Vector Space Model.

## **Procedure:**

1. Define sample documents.
2. Preprocess text data by tokenizing, removing stopwords, and punctuation.
3. Construct a TF-IDF matrix using TfidfVectorizer from sklearn.
4. Define a search function that calculates cosine similarity between a query and documents based on the TF-IDF matrix.
5. Execute a sample query and display the search results along with similarity scores.

## **Program:**

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

# Sample documents
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?",
]

# Preprocessing function to tokenize and remove stopwords/punctuation
def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    tokens = [token for token in tokens if token not in stopwords.words("english")]
    return " ".join(tokens)
    print(tokens)

# Preprocess documents
preprocessed_docs = [preprocess_text(doc) for doc in documents]

# Construct TF-IDF matrix
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(preprocessed_docs)

# Calculate cosine similarity between query and documents
def search(query, tfidf_matrix, tfidf_vectorizer):
    preprocessed_query = preprocess_text(query)
    query_vector = tfidf_vectorizer.transform([preprocessed_query])

    # Calculate cosine similarity between query and documents
    similarity_scores = cosine_similarity(query_vector, tfidf_matrix)

    # Sort documents based on similarity scores
    sorted_indexes = similarity_scores.argsort()[0][::-1]

    # Return sorted documents along with their similarity scores
    results = [(documents[i], similarity_scores[0, i]) for i in sorted_indexes]
    return results

# Example query
query = "This is the second document."

# Perform search
search_results = search(query, tfidf_matrix, tfidf_vectorizer)

# Display search results
i=1
```

```
print("Query:", query)
for result in search_results:
    print("Document:",{i}, result[0])
    print("Similarity Score:", result[1])
    print("-----")
    i=i+1
    x=result[1]
for result in search_results:
    if(result[1]>=x):
        x=result[1]
print("The high rank cosine score is",x)
```

## Output:

```
Query: This is the second document.
Document: {1} This document is the second document.
Similarity Score: 0.943354856790084
-----
Document: {2} Is this the first document?
Similarity Score: 0.338542631049127
-----
Document: {3} This is the first document.
Similarity Score: 0.338542631049127
-----
Document: {4} And this is the third one.
Similarity Score: 0.0
-----
The high rank cosine score is 0.943354856790084
```

## Result:

Thus to implement Information Retrieval Using Vector Space Model in Python is successfully executed.