LAB 4: Program to Find the Similarity Between Documents

In this lab, we will implement a program to find the similarity between documents using two different approaches: Jaccard similarity and Cosine similarity. We'll preprocess the documents by lowercasing, tokenizing, lemmatizing, and removing stopwords. Then, we'll calculate the similarity between documents using the Jaccard similarity formula and the Cosine similarity formula.

Required Libraries:

- `nltk` : For natural language processing tasks such as tokenization, lemmatization, and stopwords removal.
- `sklearn` : For TF-IDF vectorization and cosine similarity calculation.

Definitions and Formulas:

Jaccard Similarity:

The Jaccard similarity measures the similarity between two sets by comparing the intersection and union of the sets.

The Jaccard' s similarity formula : $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$

Cosine Similarity:

Cosine similarity measures the cosine of the angle between two vectors, representing the documents, in a high-dimensional space.

The Cosine Similarity Formula : $cos(\theta) = \frac{A \cdot B}{||A|| \, ||B||}$

Steps:

1. Preprocess documents: lowercase, tokenize, lemmatize, and remove stopwords.
2. Calculate Jaccard similarity between documents.
3. Calculate cosine similarity using TF-IDF vectorization.
4. Find the most similar document for each document.

Ranking with similarity

In [1]:

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

nltk.download('stopwords', quiet=True)
```

Out[1]:

```
True
```

In [2]:

```
documents = [
    "The quick brown fox jumps over the lazy dog.",
    "The dog loves to play fetch in the park.",
    "The cat sleeps all day long.",
    "The sun shines brightly in the sky.",
    "The moon shines dimly in the night."
]
```

In [3]:

```
# Preprocess documents: lowercase, tokenize, lemmatize, remove stopwords
def preprocess_document(document):
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))
    tokens = nltk.word_tokenize(document.lower())
    return [lemmatizer.lemmatize(token) for token in tokens if token not in

# Flatten the list of lists to single strings
processed_documents = [' '.join(words) for words in map(preprocess_document,
```

In [4]:

```
# Define your query
query = 'sun in the sky'

# Tokenize the documents
tokenized_documents = [doc.split() for doc in documents]

# Create a vocabulary
vocabulary = sorted(set(sum(tokenized_documents, [])))

# Jaccard Similarity
jaccard_similarity = []
for doc in tokenized_documents:
    intersection = len(set(doc).intersection(set(query.split())))
    union = len(set(doc).union(set(query.split())))
    jaccard_similarity.append(intersection / union)

# Rank documents by Jaccard Similarity
jaccard_ranking = [doc for _, doc in sorted(zip(jaccard_similarity, document

print("Jaccard Similarity Ranking:")
for doc in jaccard_ranking:
    print(doc)

# Cosine Similarity
# Create vectors
vectors = [[doc.count(word) for word in vocabulary] for doc in tokenized_doc

# Calculate dot product
dot_product = lambda a, b : sum([a[i]*b[i] for i in range(len(a))])

# Calculate magnitude
magnitude = lambda a : sum([a[i]*a[i] for i in range(len(a))]) ** 0.5

# Calculate Cosine Similarity
cosine_similarity = []
epsilon = 1e-7  # small constant
query_vector = [query.count(word) for word in vocabulary]
for doc_vector in vectors:
    cosine_similarity.append(dot_product(doc_vector, query_vector) / (magnit


# Rank documents by Cosine Similarity
cosine_ranking = [doc for _, doc in sorted(zip(cosine_similarity, documents)

print("\nCosine Similarity Ranking:")
for doc in cosine_ranking:
    print(doc)
```

```
Jaccard Similarity Ranking:
The sun shines brightly in the sky.
The moon shines dimly in the night.
The dog loves to play fetch in the park.
The quick brown fox jumps over the lazy dog.
The cat sleeps all day long.

Cosine Similarity Ranking:
The sun shines brightly in the sky.
The moon shines dimly in the night.
The dog loves to play fetch in the park.
The quick brown fox jumps over the lazy dog.
The cat sleeps all day long.
```

Checking Similarity Between docs

In [5]:

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

nltk.download('stopwords', quiet=True)
```

Out[5]:

```
True
```

In [6]:

```
# Define dummy documents
documents = [
    "The quick brown fox jumps over the lazy dog.",
    "The dog loves to play fetch in the park.",
    "The cat sleeps all day long.",
    "The sun shines brightly in the sky.",
    "The moon shines dimly in the night."
]
```

In [7]:

```
# Preprocess documents: lowercase, tokenize, lemmatize, remove stopwords
def preprocess_document(document):
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))
    tokens = nltk.word_tokenize(document.lower())
    return [lemmatizer.lemmatize(token) for token in tokens if token not in

# Flatten the list of lists to single strings
processed_documents = [' '.join(words) for words in map(preprocess_document,
```

In [8]:

```
# Jaccard similarity
def jaccard_similarity(doc1, doc2):
    intersection = len(set(doc1).intersection(set(doc2)))
    union = len(set(doc1).union(set(doc2)))
    return intersection / union
```

In [9]:

```
# Print similarity matrices
print("Jaccard similarity matrix:")
for i in range(len(documents)):
    for j in range(len(documents)):
        print(f"{jaccard_similarity(processed_documents[i], processed_docume
    print()
```

```
Jaccard similarity matrix:
1.0000 0.4815 0.4231 0.4074 0.3704
0.4815 1.0000 0.6316 0.4545 0.4762
0.4231 0.6316 1.0000 0.4500 0.6471
0.4074 0.4545 0.4500 1.0000 0.6111
0.3704 0.4762 0.6471 0.6111 1.0000
```

In [10]:

```
# Cosine similarity using TF-IDF
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(processed_documents)
similarity_matrix = cosine_similarity(tfidf_matrix)
```

In [11]:

```
print("\nCosine similarity matrix:")
print(similarity_matrix)
```

```

Cosine similarity matrix:
[[1.         0.1269686  0.         0.         0.        ]
 [0.1269686  1.         0.         0.         0.        ]
 [0.         0.         1.         0.         0.        ]
 [0.         0.         0.         1.         0.17828843]
 [0.         0.         0.         0.17828843 1.        ]]
```

In [12]:

```
# Find most similar document for each document
for i in range(len(documents)):
    most_similar_index = similarity_matrix[i].argsort()[-2]
    most_similar_doc = documents[most_similar_index]
    similarity = similarity_matrix[i, most_similar_index]
    print(f"\nDocument {i+1} is most similar to document {most_similar_index
    print(f"Similar document: {most_similar_doc}")
```

```

Document 1 is most similar to document 2 with similarity: 0.1270
Similar document: The dog loves to play fetch in the park.

Document 2 is most similar to document 1 with similarity: 0.1270
Similar document: The quick brown fox jumps over the lazy dog.

Document 3 is most similar to document 5 with similarity: 0.0000
Similar document: The moon shines dimly in the night.

Document 4 is most similar to document 5 with similarity: 0.1783
Similar document: The moon shines dimly in the night.

Document 5 is most similar to document 4 with similarity: 0.1783
Similar document: The sun shines brightly in the sky.
```