Vector space Retrieval model:

Problem 2:

This assignment is aimed at designing and developing text based information retrieval system. The assignment aims

at building the searching model according to vector space information retrieval.

Programming languages:

The assignment can be implemented in any programming language of your choice. Inbuilt packages can be used

only for Normalization (Python's NLTK Packages). You are expected to code the core functionality of the model

that you choose (TF-IDF and cosine similarity)

The task is to build a search engine for Maha sivrathri. You have to feed your IR model with documents containing

information about the Maha sivrathri. It will then process the data and build indexed. Once this is done, the user will

give a query as an input. You are supposed to return top 2 relevant documents as the output. Your result should be

explainable.

Text Collections:

Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of

the year since millions of people celebrate this momentous occasion with a lot of fervour and glee.

Lord Shiva devotees celebrate this occasion with a lot of grandness. It is accompanied by folk dances, songs,

prayers, chants, mantras etc. This year, the beautiful occasion of Maha Shivratri will be celebrated on February 18.

People keep a fast on this Maha shivratri, stay awake at night and pray to the lord for blessings, happiness, hope and

prosperity. This festival holds a lot of significance and is considered to be one of the most important festivals in

India.

The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu

festival celebrates the power of Lord Shiva. Lord Shiva protects his devotees from negative and evil spirits. He is the

epitome of powerful and auspicious energy.

Task: Build Vector based retrieval model: Apply tokenization / lemmatization, find vocabulary of terms, calculate

term frequency, Inverse document frequency and cosine similarity and Jaccard similarity.

Query: Maha Shivratri will be celebrated on February 18.

-
++

Application Description:

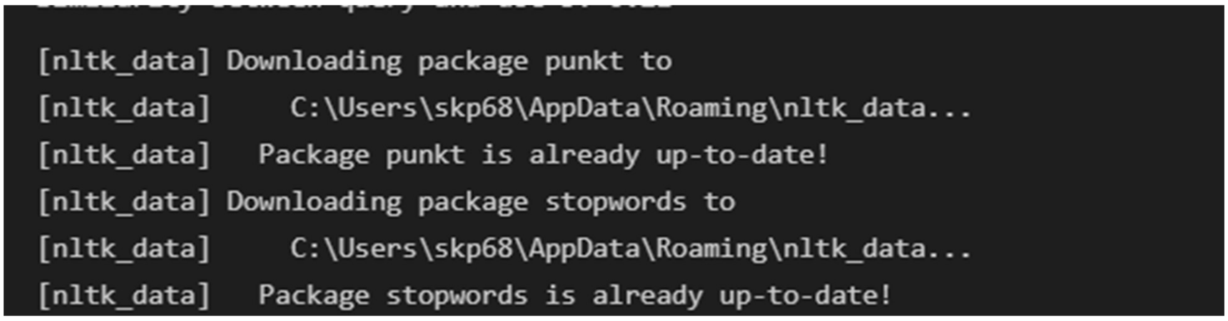
Based on a specified query, this code runs a similarity search on a set of documents. The documents and the query are represented as vectors using the TF-IDF (Term Frequency-Inverse Document Frequency) method. To calculate the similarity scores between the query vector and the document vectors, cosine similarity and Jaccardian similarity are computed.

Dataset Description:

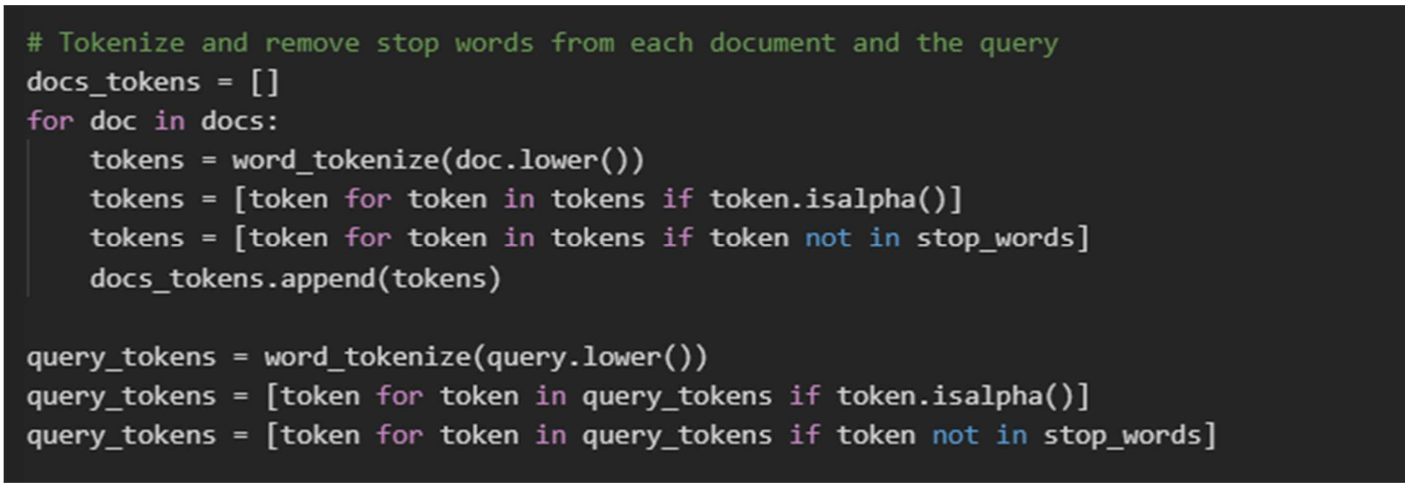
Four documents make up the dataset that is used in this code, as indicated by the variable docs. These documents are merely a few sentences that show how the code works by way of examples.

Preprocessing:

The code performs several preprocessing steps to clean and tokenize the documents and the query. First, it downloads the necessary NLTK (Natural Language Toolkit) resources, including the punkt tokenizer and stopwords corpus. The stopwords corpus is a list of common words that do not contribute significantly to the meaning of a sentence and are removed from the documents and query. Next, the code tokenizes the documents and query using the word_tokenize() function and converts all tokens to lowercase. It then removes all non-alphabetic tokens and stop words from the documents and query using list comprehension. This results in a list of tokens for each document and the query, which are used to compute the TF-IDF vectors.



```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\skp68\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\skp68\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```



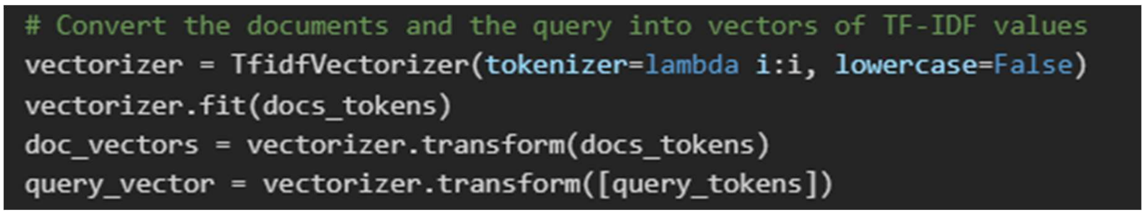
```python
# Tokenize and remove stop words from each document and the query
docs_tokens = []
for doc in docs:
    tokens = word_tokenize(doc.lower())
    tokens = [token for token in tokens if token.isalpha()]
    tokens = [token for token in tokens if token not in stop_words]
    docs_tokens.append(tokens)

query_tokens = word_tokenize(query.lower())
query_tokens = [token for token in query_tokens if token.isalpha()]
query_tokens = [token for token in query_tokens if token not in stop_words]
```

This code is responsible for tokenizing and cleaning the documents and the query.

For each document in the docs list, it first converts it to lowercase using the lower() method, and then tokenizes it using the word_tokenize() method from the nltk library. It then filters out non-alphabetic tokens using a list comprehension and removes any stop words using another list comprehension. Finally, the cleaned tokens for that document are added to the docs_tokens list.

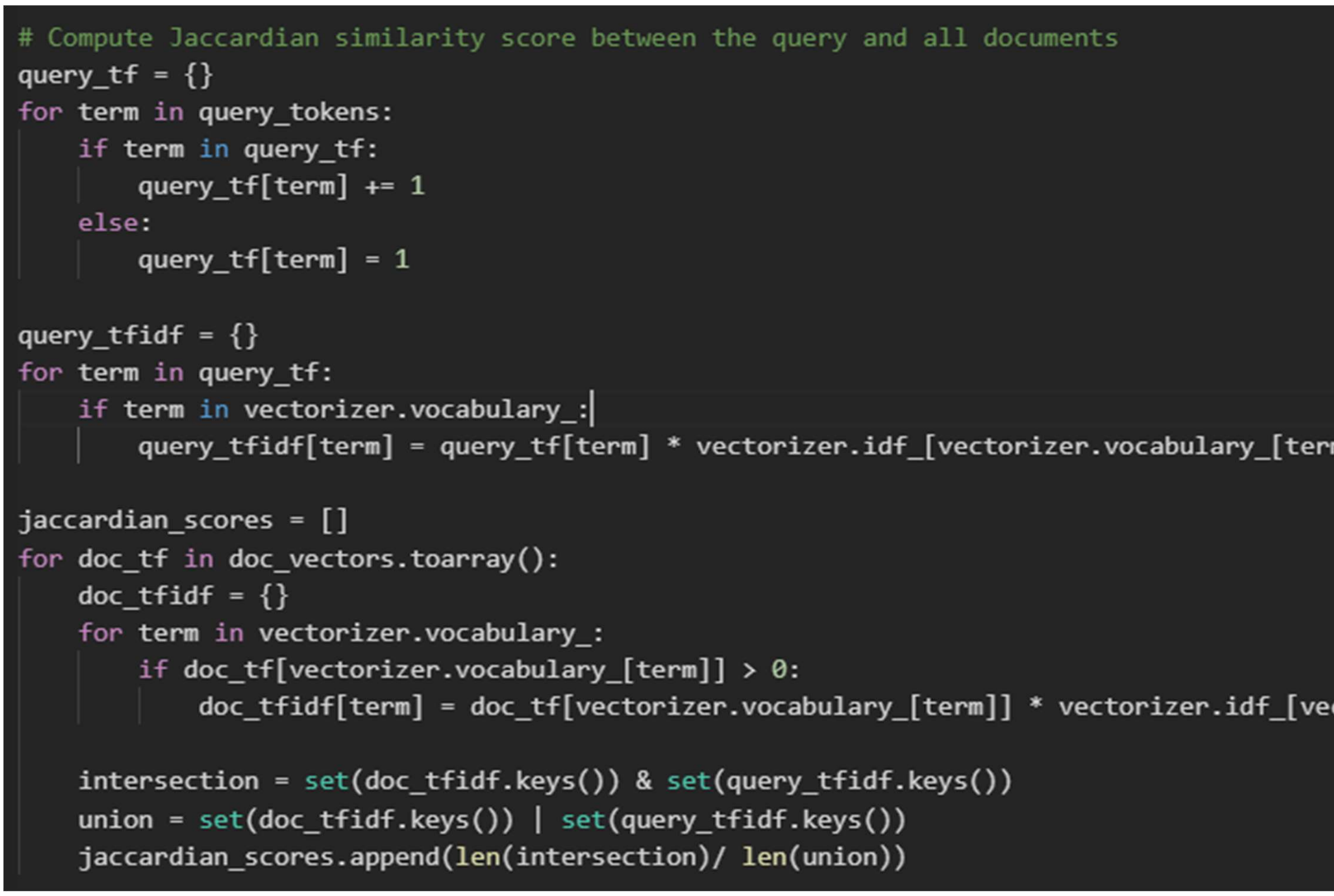
For the query, it follows the same process of converting to lowercase, tokenizing, filtering out non-alphabetic tokens, and removing stop words. The resulting cleaned tokens for the query are stored in the query_tokens variable.



```python
# Convert the documents and the query into vectors of TF-IDF values
vectorizer = TfidfVectorizer(tokenizer=lambda i:i, lowercase=False)
vectorizer.fit(docs_tokens)
doc_vectors = vectorizer.transform(docs_tokens)
query_vector = vectorizer.transform([query_tokens])
```

The TfidfVectorizer class from scikit-learn is used to convert a list of tokenized documents into a matrix of TF-IDF (Term Frequency-Inverse Document Frequency) features.

The query vector matrix is used to determine the Jaccardian similarity between each document and the query, and the resulting doc vectors matrix is used to determine the cosine similarity between each document and the query.



```python
# Compute Jaccardian similarity score between the query and all documents
query_tf = {}
for term in query_tokens:
    if term in query_tf:
        query_tf[term] += 1
    else:
        query_tf[term] = 1

query_tfidf = {}
for term in query_tf:
    if term in vectorizer.vocabulary_:
        query_tfidf[term] = query_tf[term] * vectorizer.idf_[vectorizer.vocabulary_[ter

jaccardian_scores = []
for doc_tf in doc_vectors.toarray():
    doc_tfidf = {}
    for term in vectorizer.vocabulary_:
        if doc_tf[vectorizer.vocabulary_[term]] > 0:
            doc_tfidf[term] = doc_tf[vectorizer.vocabulary_[term]] * vectorizer.idf_[ve

    intersection = set(doc_tfidf.keys()) & set(query_tfidf.keys())
    union = set(doc_tfidf.keys()) | set(query_tfidf.keys())
    jaccardian_scores.append(len(intersection)/ len(union))
```

This code computes the Jaccardian similarity score between the query and all documents in doc_vectors.

First, the term frequencies (TF) of each term in the query are computed and stored in query_tf. Then, the TF-IDF score of each term in the query is computed using the inverse document frequency (IDF) values in vectorizer.idf_ and stored in query_tfidf.

Next, a list jaccardian_scores is initialized to store the Jaccardian similarity score between the query and each document in doc_vectors. Then, for each document in doc_vectors.toarray(), the TF-IDF score of each term is computed and stored in doc_tfidf. The intersection and union of the sets of terms in query_tfidf and doc_tfidf are then computed using set operations. The Jaccardian similarity score between the query and the document is computed as the ratio of the length of the intersection to the length of the union, and this value is appended to jaccardian_scores.

Overall, this code is calculating a Jaccardian similarity score for the query and all the documents by first computing the TF-IDF scores of the terms in the query and each document, and then computing the Jaccardian similarity score based on the intersection and union of the sets of terms in the query and each document.

Running Time:

The size of the documents and the query will affect how long preprocessing takes to complete. Preprocessing time is minimal because the code in this instance only processes a small set of documents and a brief query. However, the preprocessing time might be substantial for larger datasets.

The size of the documents and the complexity of the query affect how quickly the similarity search runs. Due to the small document size and straightforward query in this code, the similarity search is completed very quickly. The running time could considerably increase for larger documents or more intricate queries, though.

In conclusion, this code offers a straightforward illustration of how to conduct a similarity search using the TF-IDF method, cosine similarity, and Jaccardian similarity.

+++
+++
++

Documentation:

Modules:

nltk: Natural Language Toolkit is a platform used for building Python programs to work with human language data.

sklearn.feature_extraction.text: Scikit-learn is an open-source Python library used for machine learning tasks. TfidfVectorizer is a module used to convert a collection of raw documents to a matrix of TF-IDF features.

sklearn.metrics.pairwise: Scikit-learn's pairwise module provides a way to calculate pairwise distances between samples in two datasets.

sklearn.metrics: Scikit-learn's metrics module provides various ways to measure the performance of machine learning models.

Functions:

cosine_similarity(): Calculates the cosine similarity between two vectors.

jaccard_score(): Calculates the Jaccard similarity score between two sets.

Classes:

TfidfVectorizer: Converts a collection of raw documents to a matrix of TF-IDF features.

In addition, the code also uses the following functions from the nltk library:

download(): Downloads necessary resources used by the Natural Language Toolkit.

corpus.stopwords(): Accesses the list of stop words in the nltk library.

tokenize(): Tokenizes a string into a list of words.

The code defines the following variables:

docs: A list of documents to be searched.

query: A string representing the query to search for in the documents.

stop_words: A set of stop words used to remove commonly used words from the documents and the query.

docs_tokens: A list of tokenized documents with stop words removed.

query_tokens: A tokenized query with stop words removed.

vectorizer: An instance of TfidfVectorizer used to convert the documents and query into vectors of TF-IDF values.

doc_vectors: A matrix of vectors representing the documents.

query_vector: A vector representing the query.

cosine_sim: A list of cosine similarity scores between the query and the documents.

query_tf: A dictionary of term frequencies for the query.

query_tfidf: A dictionary of TF-IDF values for the terms in the query.

jaccardian_scores: A list of Jaccard similarity scores between the query and the documents.

The code also prints the cosine similarity scores and Jaccardian similarity scores for the query and each document.