


```
# Importing necessary libraries
import os
import numpy as np
import math
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

```
# Function to load text files from the specified directory
def fetch_text_documents():
    folder_path = '/content/drive/MyDrive/Tech400IR/week2'
    documents = []
    file_map = {}

    # Iterate through all the files in the folder
    for index, file_name in enumerate(os.listdir(folder_path)):
        if file_name.endswith('.txt'): # Ensure only .txt files are processed
            with open(os.path.join(folder_path, file_name), 'r', encoding='utf-8') as f:
                documents.append(f.read()) # Append file content to documents list
                file_map[index] = file_name # Map the index to the file name

    return documents, file_map
```

```
# List of queries
queries = [
    'customer complaint about iOS update',
    'how to fix battery issues after update',
    'Wi-Fi disconnecting frequently after software update',
    'customer support response about internet outage',
    'MacBook performance issues after update',
    'streaming issue troubleshooting with smart TV',
    'Bluetooth speaker connection problems with Android',
    'delayed flight status update',
    'Spotify playback issue on MacBook',
    'frequent app crashes after iOS update'
]
```

```
# Tokenization function: Preprocess documents and queries (lowercase and split into words)
def tokenize(text):
    return text.lower().split()
```

```
# Load documents from your folder
docs, doc_map = fetch_text_documents()
```

```
# Tokenize documents and queries
tokenized_docs = [tokenize(doc) for doc in docs]
tokenized_queries = [tokenize(query) for query in queries]
```

```
# Tokenize documents and queries
tokenized_docs = [tokenize(doc) for doc in docs]
tokenized_queries = [tokenize(query) for query in queries]
```

```
# Build a vocabulary from all tokenized documents
vocab = set([word for doc in tokenized_docs for word in doc])
vocab = sorted(vocab) # Sorting vocabulary for consistency
print("Vocabulary:", vocab)
```

```
➦ Vocabulary: ['"took', '#apple', '#apple,', '#hpcswus', '#hpprnt', "#hpprnt'", '#ios1102
```



```
# Function to calculate term frequency (TF)
def term_frequency(term, document):
    return document.count(term) / len(document)
```

```
# Function to calculate inverse document frequency (IDF)
def inverse_document_frequency(term, all_documents):
    num_docs_containing_term = sum(1 for doc in all_documents if term in doc)
    return math.log(len(all_documents) / (1 + num_docs_containing_term))
```

```
# Function to compute the TF-IDF vector for a document
def compute_tfidf(document, all_documents, vocab):
    tfidf_vector = []
    for term in vocab:
        tf = term_frequency(term, document)
        idf = inverse_document_frequency(term, all_documents)
        tfidf_vector.append(tf * idf)
    return np.array(tfidf_vector)
```

```
# Function to compute cosine similarity between two vectors
def cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2)
```

```

# Calculate TF-IDF vectors for documents and queries
doc_tfidf_vectors = [compute_tfidf(doc, tokenized_docs, vocab) for doc in tokenized_docs]
query_tfidf_vectors = [compute_tfidf(query, tokenized_docs, vocab) for query in tokenized_queries]

# Path for saving output
output_file_path = "/content/drive/MyDrive/Tech400IR/cosine_similarities_output.txt"

# Open the file in write mode and calculate cosine similarities
with open(output_file_path, 'w') as f:
    cosine_similarities = []

    # Calculate cosine similarities between queries and documents
    for query_vector in query_tfidf_vectors:
        similarities = [cosine_similarity(query_vector, doc_vector) for doc_vector in doc_tfidf_vectors]
        cosine_similarities.append(similarities)


# Write the results in ascending order of cosine similarity
for i, query in enumerate(queries):
    f.write(f"\nCosine similarities for query '{query}':\n")

    # Pairing document indices with corresponding similarities
    doc_sim_pairs = list(enumerate(cosine_similarities[i]))

    # Sort document similarities in ascending order
    doc_sim_pairs_sorted = sorted(doc_sim_pairs, key=lambda x: x[1])

    # Write the sorted similarities to the file
    for doc_idx, similarity in doc_sim_pairs_sorted:
        f.write(f"Document {doc_idx + 1}: {similarity:.4f}\n")

```

 <ipython-input-12-9ba007465456>:6: RuntimeWarning: invalid value encountered in scalar c
return dot_product / (norm_vec1 * norm_vec2)



```

# Output confirmation
print(f"Output has been saved to {output_file_path}")

```

 Output has been saved to /content/drive/MyDrive/Tech400IR/cosine_similarities_output.txt



