

EXP NO: 4a	INFORMATION RETRIEVAL USING NLTK
DATE: 12/8/25	

Aim:

To implement an information retrieval system that searches and ranks Amazon product reviews based on relevance using NLTK and TF-IDF.

Program:

Step 1: Import Required Libraries

```
import pandas as pd
import numpy as np
import nltk
import string
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

Step 2: Download NLTK Resources

```
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words("english"))
```

Step 3: Load Dataset

```
import os
import kagglehub

# Download Amazon Fine Food Reviews dataset
path = kagglehub.dataset_download("snap/amazon-fine-food-reviews")
# Load the CSV file
df = pd.read_csv(os.path.join(path, "Reviews.csv"))

# Select review column and limit to 1000 entries reviews
reviews = df['Text'].dropna()[:1000]
```

Step 4: Define Text Preprocessing Function

```
def preprocess(text):
    text = text.lower()
    # lowercase
    text = re.sub(r'^a-z\s', '', text)
    # remove punctuation/special chars
    tokens = word_tokenize(text)
    # tokenize
    tokens = [w for w in tokens if w not in stop_words]
    # remove stopwords
    cleaned_tokens = ".join(tokens)"
    # join cleaned tokens
    return cleaned_tokens
```

Step 5: Apply Preprocessing to Reviews

```
cleaned_reviews = reviews.apply(preprocess)
```

Step 6: Vectorize Reviews using TF-IDF

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(cleaned_reviews)
```

Step 7: Define Search Function

```
def search(query, top_k=2):

    query_cleaned= preprocess(query)
    query_vec = vectorizer.transform([query_cleaned])
    similarity = cosine_similarity(query_vec, X).flatten()
    indices = similarity.argsort()[ - top_k:][::-1 - #] top k results
    results= []
    for i in indices:
        results.append({
            "original": reviews.iloc[i],
            "cleaned": cleaned_reviews.iloc[i],
            "score": similarity[i]
        })
    return results
```

Step 8: TestQueries

```
# Query 1
print("    Query: great product with fast shipping")
for res in search("great product with fast shipping"):
    print("\nOriginal:", res["original"])
    print("Cleaned :", res["cleaned"])
    print("Score   :", res["score"])

# Query 2
print(" \n Query: disappointed")
for res in search("disappointed"):
    print(" \nOriginal:", res["original"])
    print("Cleaned :", res["cleaned"])
    print("Score   :", res["score"])
```

Output:

```
• Query: great product with fast shipping
Original: Use frequently as we like to do Asian dishes at least once a week. Love this product. Fast shipping, as usual. Would buy again.
Cleaned : use frequently like asian dishes least week love product fast shipping usual would buy
Score   : 0.36518443846886385

Original: This stuff is great because it's low glycemic. Substitute this to sugar and you'll be doing your body a great favor. This size is economical and shipping is fast, too.
Cleaned : stuff great low glycemic substitute sugar youll body great favor size economical shipping fast got mine soon
Score   : 0.3431526468487778

• Query: disappointed
Original: I was disappointed in this product because I thought it would be bigger.<br />Also, it did not come with enough icing. I had to use my own.
Cleaned : disappointed product thought would biggerbr also come enough icing use
Score   : 0.32947843843711086

Original: My husband and I were very disappointed in this coffee, very weak, watery cup of coffee. A definite waste of $13.00.
Cleaned : husband disappointed coffee weak watery cup coffee definite waste
Score   : 0.2874266474182988
```

Result:

The system successfully returned the top relevant reviews for queries by computing cosine similarity on preprocessed text, demonstrating effective retrieval of related documents.