Practical :- 2

import math

from collections import Counter

def tokenize(text):

# Simple tokenization, you might want to improve it based on your specific needs

return text.lower().split()

def compute\_tf(doc):

# Computes term frequency (TF) for each term in the document

word\_count = Counter(tokenize(doc))

total\_words = len(tokenize(doc))

tf = {word: count/total\_words for word, count in word\_count.items()}

return tf

def compute\_idf(documents):

# Computes inverse document frequency (IDF) for each term in the collection of documents

N = len(documents)

idf = {}

for doc in documents:

terms = set(tokenize(doc))

for term in terms:

idf[term] = idf.get(term, 0) + 1

idf = {term: math.log(N/frequency) for term, frequency in idf.items()}

return idf

def compute\_tf\_idf(tf, idf):

# Computes TF-IDF for each term in the document

tf\_idf = {term: tf[term] \* idf.get(term, 0) for term in tf}

return tf\_idf

def compute\_cosine\_similarity(query\_tf\_idf, doc\_tf\_idf):

# Computes cosine similarity between query and document TF-IDF vectors

common\_terms = set(query\_tf\_idf.keys()) & set(doc\_tf\_idf.keys())

dot\_product = sum(query\_tf\_idf[term] \* doc\_tf\_idf[term] for term in common\_terms)

query\_norm = math.sqrt(sum(value\*\*2 for value in query\_tf\_idf.values()))

doc\_norm = math.sqrt(sum(value\*\*2 for value in doc\_tf\_idf.values()))

if query\_norm == 0 or doc\_norm == 0:

return 0

similarity = dot\_product / (query\_norm \* doc\_norm)

return similarity

similarity = dot\_product / (query\_norm \* doc\_norm)

return similarity

def vector\_space\_model(documents, query):

# Build TF-IDF vectors for documents and query

tf\_idf\_vectors = []

for doc in documents:

tf = compute\_tf(doc)

idf = compute\_idf(documents)

tf\_idf = compute\_tf\_idf(tf, idf)

tf\_idf\_vectors.append(tf\_idf)

query\_tf = compute\_tf(query)

query\_idf = compute\_idf(documents)

query\_tf\_idf = compute\_tf\_idf(query\_tf, query\_idf)

# Compute cosine similarity between query and each document

similarities = []

for doc\_tf\_idf in tf\_idf\_vectors:

similarity = compute\_cosine\_similarity(query\_tf\_idf, doc\_tf\_idf)

similarities.append(similarity)

return similarities

# Example usage:

documents = [

"This is the first document.",

"This document is the second document.",

"And this is the third one.",

"Is this the first document?",

]

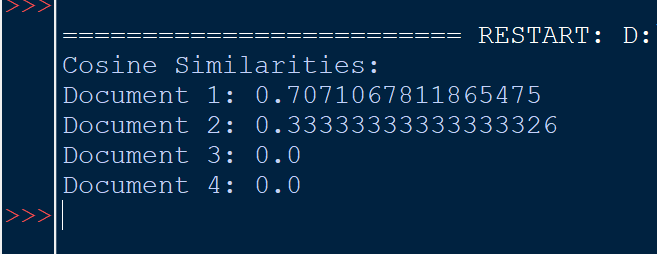
query = "This is a query document."

similarities = vector\_space\_model(documents, query)

print("Cosine Similarities:")

for i, sim in enumerate(similarities):

print(f"Document {i + 1}: {sim}")



Practical :- 3

def edit\_distance(str1, str2):

m, n = len(str1), len(str2)

dp = [[0] \* (n + 1) for \_ in range(m + 1)]

for i in range(m + 1):

for j in range(n + 1):

if i == 0:

dp[i][j] = j

elif j == 0:

dp[i][j] = i

elif str1[i-1] == str2[j-1]:

dp[i][j] = dp[i-1][j-1]

else:

dp[i][j] = 1 + min(dp[i][j-1], # Insert

dp[i-1][j], # Remove

dp[i-1][j-1] # Replace

)

return dp[m][n]

def correct\_spelling(query, dictionary):

corrected\_query = query.lower() # Convert to lowercase for case-insensitive correction

suggestions = []

for word in dictionary:

distance = edit\_distance(corrected\_query, word.lower())

if distance <= 2: # You can adjust this threshold based on your preference

suggestions.append((word, distance))

suggestions.sort(key=lambda x: x[1]) # Sort by distance in ascending order

return suggestions

# Example usage

dictionary = ["information", "retrieval", "spelling", "correction", "system"]

query = input("Enter your query: ")

suggestions = correct\_spelling(query, dictionary)

if suggestions:

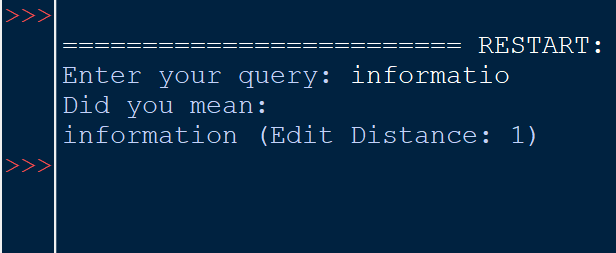
print("Did you mean:")

for suggestion, distance in suggestions:

print(f"{suggestion} (Edit Distance: {distance})")

else:

print("No suggestions found.")



Practical :- 4

4.A

def calculate\_metrics(retrieved\_set, relevant\_set):

true\_positive = len(retrieved\_set.intersection(relevant\_set))

false\_positive = len(retrieved\_set.difference(relevant\_set))

false\_negative = len(relevant\_set.difference(retrieved\_set))

print("True Positive: ", true\_positive,

"\nFalse Positive: ", false\_positive,

"\nFalse Negative: ", false\_negative, "\n")

precision = true\_positive / (true\_positive + false\_positive)

recall = true\_positive / (true\_positive + false\_negative)

f\_measure = 2 \* precision \* recall / (precision + recall)

return precision, recall, f\_measure

retrieved\_set = set(["doc1", "doc2", "doc3"]) # Predicted set

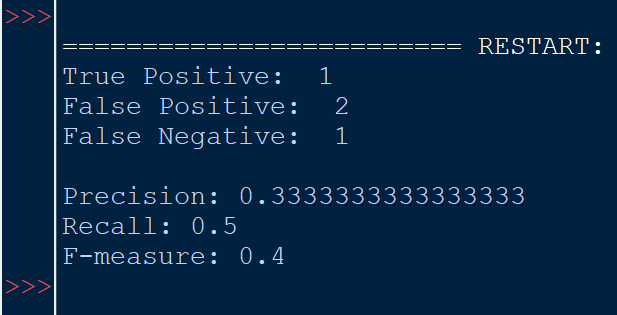
relevant\_set = set(["doc1", "doc4"]) # Actually Needed set (Relevant)

precision, recall, f\_measure = calculate\_metrics(retrieved\_set, relevant\_set)

print(f"Precision: {precision}")

print(f"Recall: {recall}")

print(f"F-measure: {f\_measure}")



4.B

from sklearn.metrics import average\_precision\_score, precision\_recall\_curve, auc

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

import matplotlib.pyplot as plt

import numpy as np

def calculate\_classification\_metrics(y\_true, y\_pred\_prob):

"""

Calculate various classification metrics.

Parameters:

- y\_true: True labels (ground truth).

- y\_pred\_prob: Predicted probabilities for the positive class.

Returns:

- Dictionary containing different evaluation metrics.

"""

# Convert probabilities to binary predictions

y\_pred\_binary = (np.array(y\_pred\_prob) >= 0.5).astype(int)

# Accuracy

accuracy = accuracy\_score(y\_true, y\_pred\_binary)

# Precision, Recall, F1 Score

precision = precision\_score(y\_true, y\_pred\_binary)

recall = recall\_score(y\_true, y\_pred\_binary)

f1 = f1\_score(y\_true, y\_pred\_binary)

# ROC-AUC

roc\_auc = roc\_auc\_score(y\_true, y\_pred\_prob)

# Average Precision and Precision-Recall curve

average\_precision = average\_precision\_score(y\_true, y\_pred\_prob)

precision\_curve, recall\_curve, \_ = precision\_recall\_curve(y\_true, y\_pred\_prob)

area\_under\_pr\_curve = auc(recall\_curve, precision\_curve)

# Print metrics

print(f'Accuracy: {accuracy}')

print(f'Precision: {precision}')

print(f'Recall: {recall}')

print(f'F1 Score: {f1}')

print(f'ROC-AUC: {roc\_auc}')

print(f'Average Precision: {average\_precision}')

print(f'Area under Precision-Recall curve: {area\_under\_pr\_curve}')

# Plot Precision-Recall curve

plt.figure(figsize=(8, 6))

plt.plot(recall\_curve, precision\_curve, color='blue', lw=2, label='Precision-Recall curve')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.legend()

plt.show()

# Example usage:

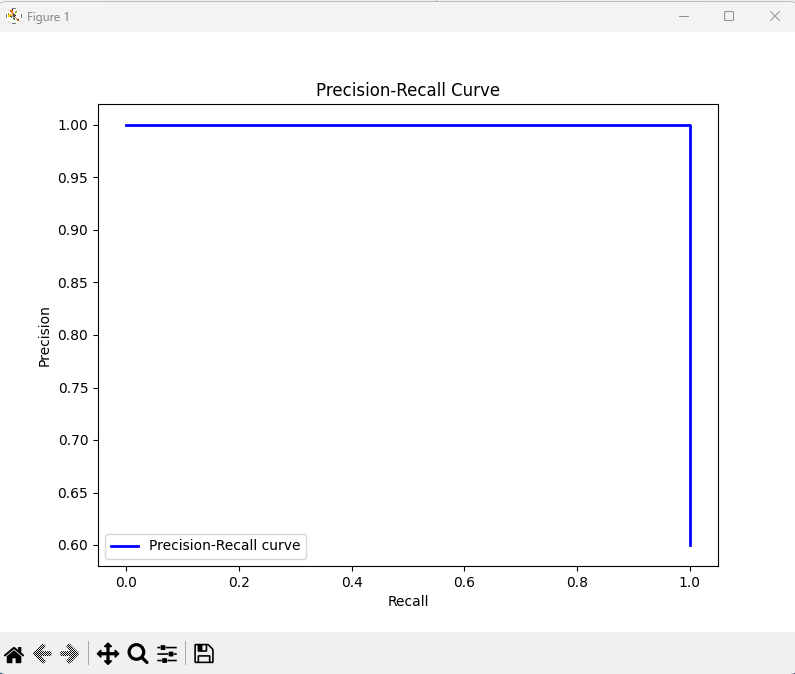
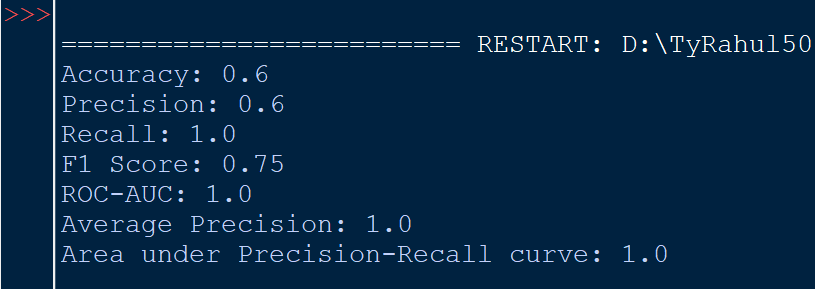
# Replace the following placeholders with your actual data

# Example for binary classification

y\_true\_binary = [1, 0, 1, 0, 1] # True binary labels (1 for positive, 0 for negative)

y\_pred\_prob\_binary = [0.8, 0.6, 0.9, 0.5, 0.7] # Predicted probabilities for the positive class

calculate\_classification\_metrics(y\_true\_binary, y\_pred\_prob\_binary)



Practical :- 5

# Import necessary libraries

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

# Load the 20 Newsgroups dataset

newsgroups = fetch\_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(newsgroups.data, newsgroups.target, test\_size=0.2, random\_state=42)

# Convert text data to TF-IDF features

vectorizer = TfidfVectorizer()

X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = vectorizer.transform(X\_test)

# Train a Support Vector Machine (SVM) classifier

svm\_classifier = SVC(kernel='linear', C=1.0)

svm\_classifier.fit(X\_train\_tfidf, y\_train)

# Make predictions on the test set

y\_pred = svm\_classifier.predict(X\_test\_tfidf)

# Evaluate the performance of the classifier

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=newsgroups.target\_names)

# Print the results

print(f"Accuracy: {accuracy:.2f}")

print("\nClassification Report:")

print(classification\_rep)

