PRACTICAL NO 1 [A]

AIM: Document Indexing and Retrieval

1. Implement an inverted index construction algorithm.

2. Build a simple document retrieval system using the constructed index.

SOLUTION:

1. Implement an inverted index construction algorithm.

INPUT:

```
document1 = "The quick brown fox jumped over the lazy dog"

document2 = "The lazy dog slept in the sun"

import nltk

from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

# Tokenize and filter stopwords

tokens1 = [word for word in document1.lower().split() if word not in stop_words]

tokens2 = [word for word in document2.lower().split() if word not in stop_words]

# Build inverted index and occurrence counts

inverted_index = {}

occurrences = {"Document 1": {}, "Document 2": {}}

for term in set(tokens1 + tokens2):

  inverted_index[term] = []

  if term in tokens1:

    inverted_index[term].append("Document 1")

    occurrences["Document 1"][term] = tokens1.count(term)

  if term in tokens2:

    inverted_index[term].append("Document 2")

    occurrences["Document 2"][term] = tokens2.count(term)

# Print results

print("Inverted Index:", inverted_index)

print("Occurrences in Document 1:", occurrences["Document 1"])

print("Occurrences in Document 2:", occurrences["Document 2"])

# Print inverted index with occurrences

for term, docs in inverted_index.items():

  print(f"{term} ->", end="")

  print(", ".join(f"{doc}({occurrences[doc].get(term, 0)})" for doc in docs))

print("Performed by 740_Pallavi & 743_Deepak")
```

OUTPUT:

```
Inverted Index: {'jumped': ['Document 1'], 'dog': ['Document 1', 'Document 2'], 'brown': ['Documen
t 1'], 'slept': ['Document 2'], 'sun': ['Document 2'], 'quick': ['Document 1'], 'lazy': ['Document
1', 'Document 2'], 'fox': ['Document 1']}
Occurrences in Document 1: {'jumped': 1, 'dog': 1, 'brown': 1, 'quick': 1, 'lazy': 1, 'fox': 1}
Occurrences in Document 2: {'dog': 1, 'slept': 1, 'sun': 1, 'lazy': 1}
jumped ->Document 1(1)
dog ->Document 1(1), Document 2(1)
brown ->Document 1(1)
slept ->Document 2(1)
sun ->Document 2(1)
quick ->Document 1(1)
lazy ->Document 1(1), Document 2(1)
fox ->Document 1(1)
Performed by 740_Pallavi & 743_Deepak
```

PRACTICAL NO 1 [B]

2. Build a simple document retrieval system using the constructed index.

INPUT:

```
import re
from collections import defaultdict
class DocumentRetrievalSystem:
  def __init__(self):
    self.index = defaultdict(list)
    self.documents = []
  def add_documents(self, documents):
    for doc_id, document in enumerate(documents):
      self.documents.append(document)
      for term in self.tokenize(document):
        self.index[term].append(doc_id)
  def search(self, query):
    query_terms = self.tokenize(query)
    result_docs = set(self.index[query_terms[0]]) if query_terms and query_terms[0] in self.index else set()
    for term in query_terms[1:]:
      result_docs &= set(self.index[term])
    return [self.documents[doc_id] for doc_id in result_docs]
  def tokenize(text):
    return re.findall(r'\b\w+\b', text.lower())
if __name__ == "__main__":
  retrieval_system = DocumentRetrievalSystem()
  retrieval_system.add_documents([
    "This is the first document",
    "Python is a popular programming language",
    "Document retrieval systems are important"])
  query = "is"
  results = retrieval_system.search(query)
  if results:
    print(f"Search results for '{query}':")
    for result in results:
      print("-", result)
  else:
    print(f"No results found for '{query}'.")
```

OUTPUT:

```
Search results for 'is' :
- This is the first document
- Python is a popular programming language
```

PRACTICAL NO 2[A]

AIM : Retrieval Models

Implement the Boolean retrieval model and process queries.

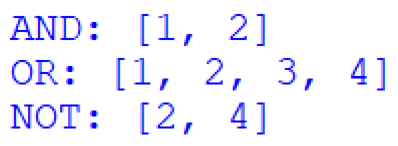
Implement the vector space model with TF-IDF weighting and cosine similarity.

SOLUTION: 1) Implement the Boolean retrieval model and process queries

INPUT:

```
documents = {1: "apple banana orange",
  2: "apple banana",
  3: "banana orange",
  4: "apple",}
def build_index(docs):
  index = {}
  for doc_id, text in docs.items():
    for term in set(text.split()):
      index.setdefault(term, set()).add(doc_id)
  return index
inverted_index = build_index(documents)
def boolean_and(operands, index):
  result = index.get(operands[0], set())
  for term in operands[1:]:
    result &= index.get(term, set())
  return list(result)
def boolean_or(operands, index, total_docs):
  result = set()
  for term in operands:
    result |= index.get(term, set())
  return list(result | set(range(1, total_docs + 1)))
def boolean_not(operand, index, total_docs):
  return list(set(range(1, total_docs + 1)) - index.get(operand, set()))
query1, query2, query3 = ["apple", "banana"], ["apple", "orange"], "orange"
print("AND:", boolean_and(query1, inverted_index))
print("OR:", boolean_or(query2, inverted_index, len(documents)))
print("NOT:", boolean_not(query3, inverted_index, len(documents)))
```

OUTPUT:



```
AND: [1, 2]
OR: [1, 2, 3, 4]
NOT: [2, 4]
```

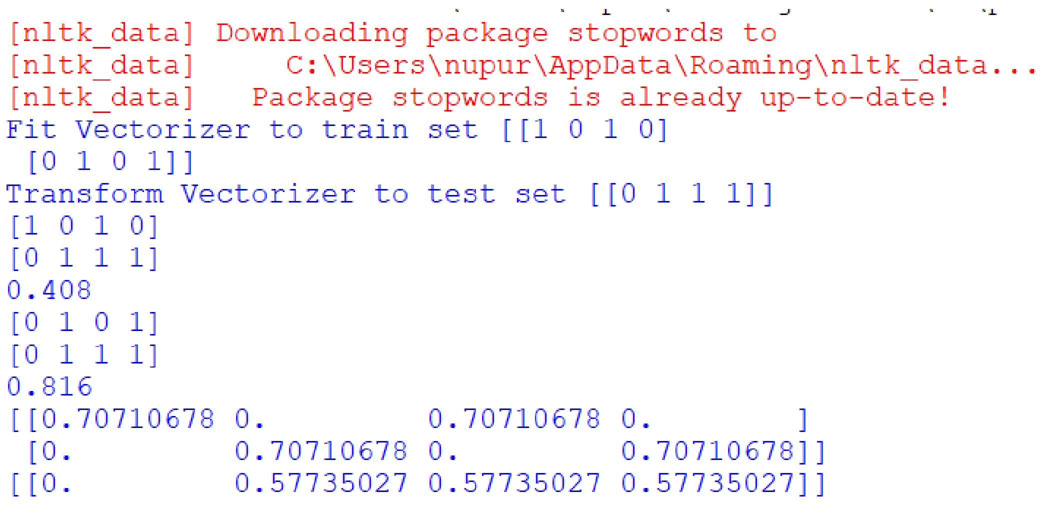
PRACTICAL NO 2[B]

2) Implement the vector space model with TF-IDF weighting and cosine similarity.

INPUT:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

import nltk

from nltk.corpus import stopwords

import numpy as np

from numpy.linalg import norm

train_set = ["The sky is blue.", "The sun is bright."]

test_set = ["The sun in the sky is bright."]

nltk.download('stopwords')

stopWords = stopwords.words('english')

vectorizer = CountVectorizer(stop_words=stopWords)

trainVectorizerArray = vectorizer.fit_transform(train_set).toarray()

testVectorizerArray = vectorizer.transform(test_set).toarray()

print('Fit Vectorizer to train set', trainVectorizerArray)

print('Transform Vectorizer to test set', testVectorizerArray)

cx = lambda a, b: round(np.inner(a, b) / (norm(a) * norm(b)), 3)

for vector in trainVectorizerArray:

  print(vector)

  for testV in testVectorizerArray:

    print(testV)

    cosine = cx(vector, testV)

    print(cosine)

transformer = TfidfTransformer()

print(transformer.fit_transform(trainVectorizerArray).toarray())

print(transformer.fit_transform(testVectorizerArray).todense())
```

OUTPUT:



```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\nupur\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Fit Vectorizer to train set [[1 0 1 0]
 [0 1 0 1]]
Transform Vectorizer to test set [[0 1 1 1]]
[1 0 1 0]
[0 1 1 1]
0.408
[0 1 0 1]
[0 1 1 1]
0.816
[[0.70710678 0.         0.70710678 0.        ]
 [0.         0.70710678 0.         0.70710678]]
[[0.         0.57735027 0.57735027 0.57735027]]
```

PRACTICAL NO 3

AIM : Spelling Correction in IR Systems

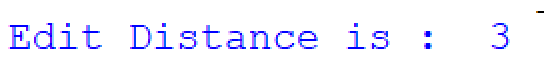
Develop a spelling correction module using edit distance algorithms.

Integrate the spelling correction module into an information retrieval system.

INPUT:

```
def editDistance(str1, str2, m, n):
  if m==0:
    return n
  if n==0:
    return m
  if str1[m-1] == str2[n-1]:
    return editDistance(str1, str2, m-1, n-1)
  return 1 + min(editDistance(str1, str2, m, n-1),
        editDistance(str1, str2, m-1, n),
        editDistance(str1, str2, m-1, n-1) )
str1 = "sunday"
str2 = "saturday"
print("Edit Distance is : ", editDistance(str1, str2, len(str1), len(str2)))
```

OUTPUT:



```
Edit Distance is :   3
```

PRACTICAL NO 4[A]

AIM : Evaluation Metrics for IR Systems

- Calculate precision, recall, and F-measure for a given set of retrieval results.
- Use an evaluation toolkit to measure average precision and other evaluation metrics.

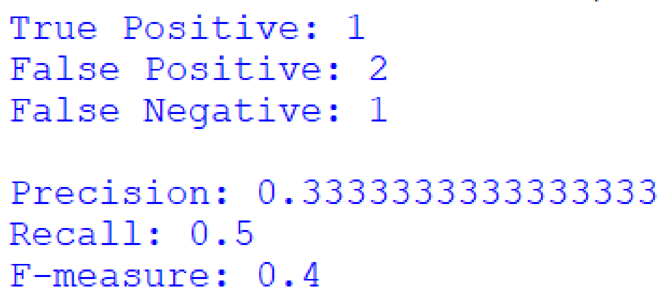
SOLUTION:

1) Calculate precision, recall, and F-measure for a given set of retrieval results.

INPUT:

```
def calculate_metrics(retrieved_set, relevant_set):
  tp = len(retrieved_set & relevant_set)  # True Positives
  fp = len(retrieved_set - relevant_set)  # False Positives
  fn = len(relevant_set - retrieved_set)  # False Negatives

  print(f"True Positive: {tp}\nFalse Positive: {fp}\nFalse Negative: {fn}\n")

  precision = tp / (tp + fp) if (tp + fp) > 0 else 0
  recall = tp / (tp + fn) if (tp + fn) > 0 else 0
  f_measure = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0

  return precision, recall, f_measure

# Example input
retrieved_set = {"doc1", "doc2", "doc3"}  # Predicted set
relevant_set = {"doc1", "doc4"}          # Relevant set

# Calculate and display metrics
precision, recall, f_measure = calculate_metrics(retrieved_set, relevant_set)
print(f"Precision: {precision}\nRecall: {recall}\nF-measure: {f_measure}")
```

OUTPUT:



```
True Positive: 1
False Positive: 2
False Negative: 1

Precision: 0.3333333333333333
Recall: 0.5
F-measure: 0.4
```

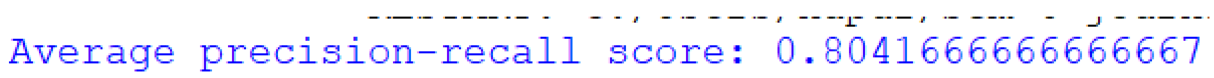
PRACTICAL NO 4[B]

2) Use an evaluation toolkit to measure average precision and other evaluation metrics.

INPUT:

```
from sklearn.metrics import average_precision_score

y_true = [0, 1, 1, 0, 1, 1] #Binary Prediction

y_scores = [0.1, 0.4, 0.35, 0.8, 0.65, 0.9] #Model's estimation score

average_precision = average_precision_score(y_true, y_scores)

print(f'Average precision-recall score: {average_precision}')
```

OUTPUT:



```
Average precision-recall score: 0.8041666666666667
```

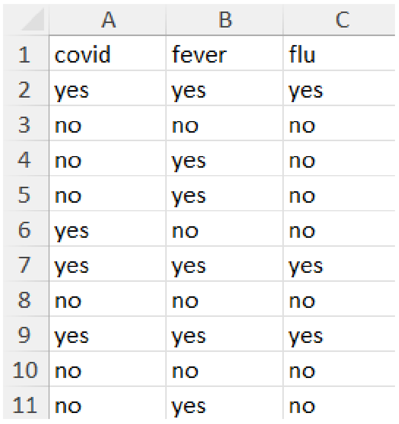
PRACTICAL NO 5

AIM : Text Categorization

- Implement a text classification algorithm (e.g., Naive Bayes or Support Vector Machines).
- Train the classifier on a labelled dataset and evaluate its performance.

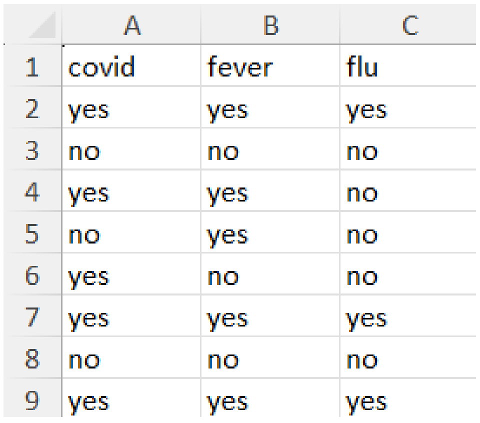
SOLUTION :

Dataset.csv file



	A	B	C
1	covid	fever	flu
2	yes	yes	yes
3	no	no	no
4	no	yes	no
5	no	yes	no
6	yes	no	no
7	yes	yes	yes
8	no	no	no
9	yes	yes	yes
10	no	no	no
11	no	yes	no

Test.csv file



	A	B	C
1	covid	fever	flu
2	yes	yes	yes
3	no	no	no
4	yes	yes	no
5	no	yes	no
6	yes	no	no
7	yes	yes	yes
8	no	no	no
9	yes	yes	yes

INPUT:

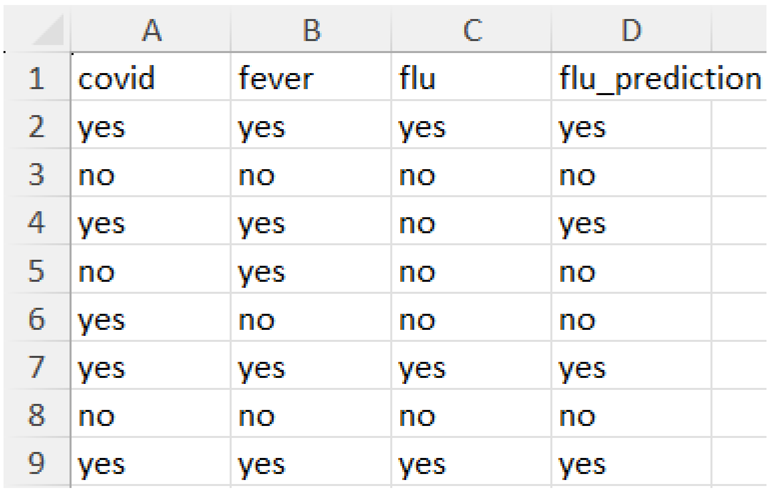
```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report

df=pd.read_csv(r"C:\Users\nupur\sem 6 journals\IR\Dataset.csv")

data= df["covid"]+" "+df["fever"]

X=data.astype(str)

y=df['flu']

X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2, random_state=42)

vectorizer=CountVectorizer()

X_train_counts=vectorizer.fit_transform(X_train)

X_test_counts = vectorizer.transform(X_test)

classifier=MultinomialNB()

classifier.fit(X_train_counts, y_train)
```

```
data1=pd.read_csv(r"C:\Users\nupur\sem 6 journals\IR\Test.csv")

new_data=data1["covid"]+" "+data1["fever"]

new_data_counts=vectorizer.transform(new_data.astype(str))

predictions=classifier.predict(new_data_counts)

new_data=predictions

print(new_data)

accuracy=accuracy_score(y_test, classifier.predict(X_test_counts))

print(f"\nAccuracy : {accuracy:.2f}")

print("Classification Report :")

print(classification_report(y_test,classifier.predict(X_test_counts)))

predictions_df=pd.DataFrame(predictions,columns=['flu_prediction'])

data1=pd.concat([data1,predictions_df],axis=1)


data1.to_csv(r"C:\Users\nupur\sem 6 journals\IR\test1.csv",index=False)
```

OUTPUT:

```
['yes' 'no' 'yes' 'no' 'no' 'yes' 'no' 'yes']

Accuracy : 1.00
Classification Report :
              precision    recall  f1-score   support

          no       1.00      1.00      1.00         2

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

test1.csv



	A	B	C	D
1	covid	fever	flu	flu_prediction
2	yes	yes	yes	yes
3	no	no	no	no
4	yes	yes	no	yes
5	no	yes	no	no
6	yes	no	no	no
7	yes	yes	yes	yes
8	no	no	no	no
9	yes	yes	yes	yes

PRACTICAL NO 6

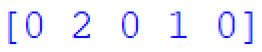
AIM : Clustering for Information Retrieval

- Implement a clustering algorithm (e.g., K-means or hierarchical clustering).
- Apply the clustering algorithm to a set of documents and evaluate the clustering results.

INPUT:

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans


documents=["Cats are known for their agility and grace",

    "Dogs are often called 'Man's best friend'.",

    "Some digs are trained to assist people with disablities.",

    "The sun rises in the east and set on the west.",

    "Many cats enjoy climbing trees and chasing toys.",

    ]


vectorizer = TfidfVectorizer(stop_words='english')

X=vectorizer.fit_transform(documents)


kmeans = KMeans(n_clusters=3, random_state=0).fit(X)

print(kmeans.labels_)
```

OUTPUT:



```
[0 2 0 1 0]
```

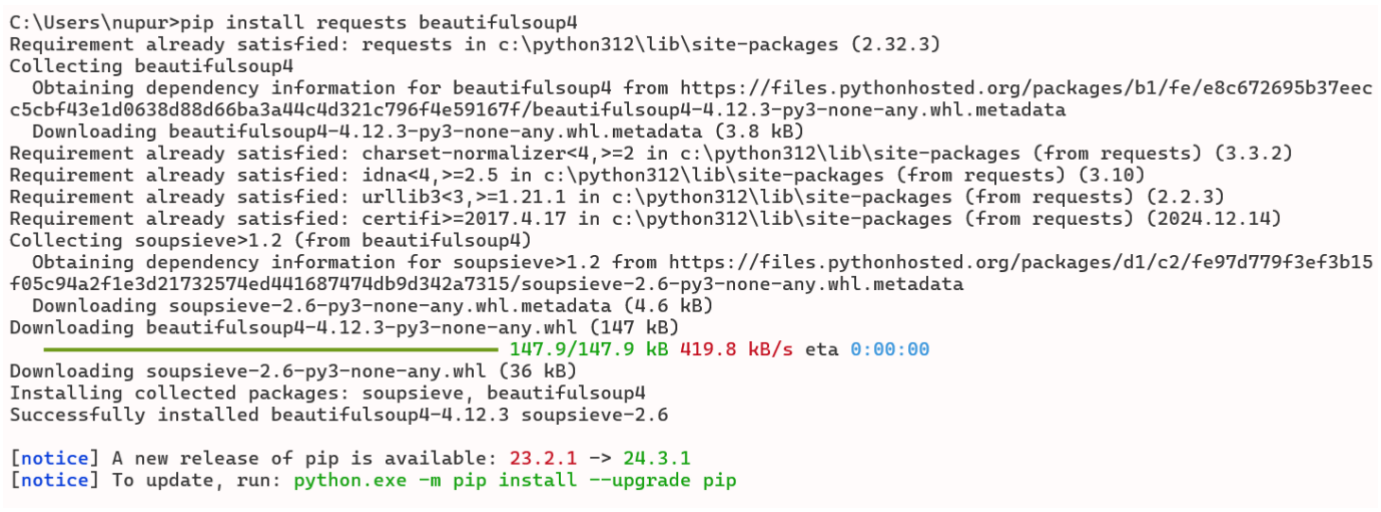
PRACTICAL NO 7

AIM : Web Crawling and Indexing

- Develop a web crawler to fetch and index web pages.
- Handle challenges such as robots.txt, dynamic content, and crawling delays.

SOLUTION:

Install the following modules using pip: pip install requests beautifulsoap4

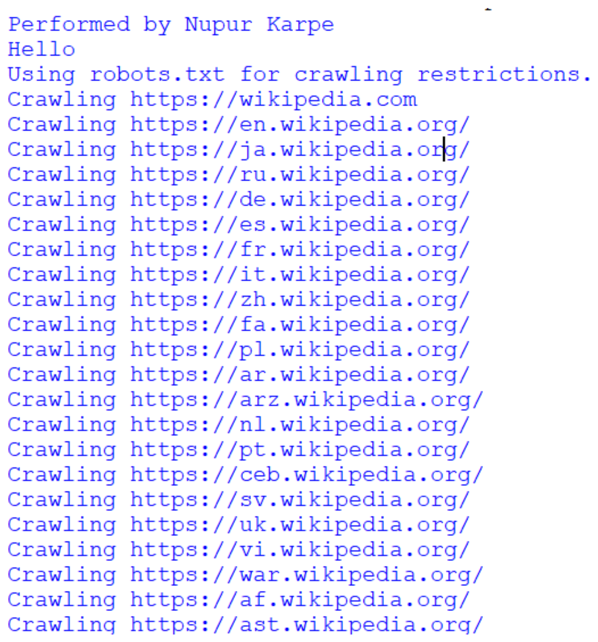


```
C:\Users\nupur>pip install requests beautifulsoup4
Requirement already satisfied: requests in c:\python312\lib\site-packages (2.32.3)
Collecting beautifulsoup4
  Obtaining dependency information for beautifulsoup4 from https://files.pythonhosted.org/packages/b1/fe/e8c672695b37eec
c5cbf43e1d0638d88d66ba3a44c4d321c796f4e59167f/beautifulsoup4-4.12.3-py3-none-any.whl.metadata
  Downloading beautifulsoup4-4.12.3-py3-none-any.whl.metadata (3.8 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\python312\lib\site-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\python312\lib\site-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\python312\lib\site-packages (from requests) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\python312\lib\site-packages (from requests) (2024.12.14)
Collecting soupsieve>1.2 (from beautifulsoup4)
  Obtaining dependency information for soupsieve>1.2 from https://files.pythonhosted.org/packages/d1/c2/fe97d779f3ef3b15
f05c94a2f1e3d21732574ed441687474db9d342a7315/soupsieve-2.6-py3-none-any.whl.metadata
  Downloading soupsieve-2.6-py3-none-any.whl.metadata (4.6 kB)
Downloading beautifulsoup4-4.12.3-py3-none-any.whl (147 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 147.9/147.9 kB 419.8 kB/s eta 0:00:00
Downloading soupsieve-2.6-py3-none-any.whl (36 kB)
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.12.3 soupsieve-2.6

[notice] A new release of pip is available: 23.2.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

INPUT:

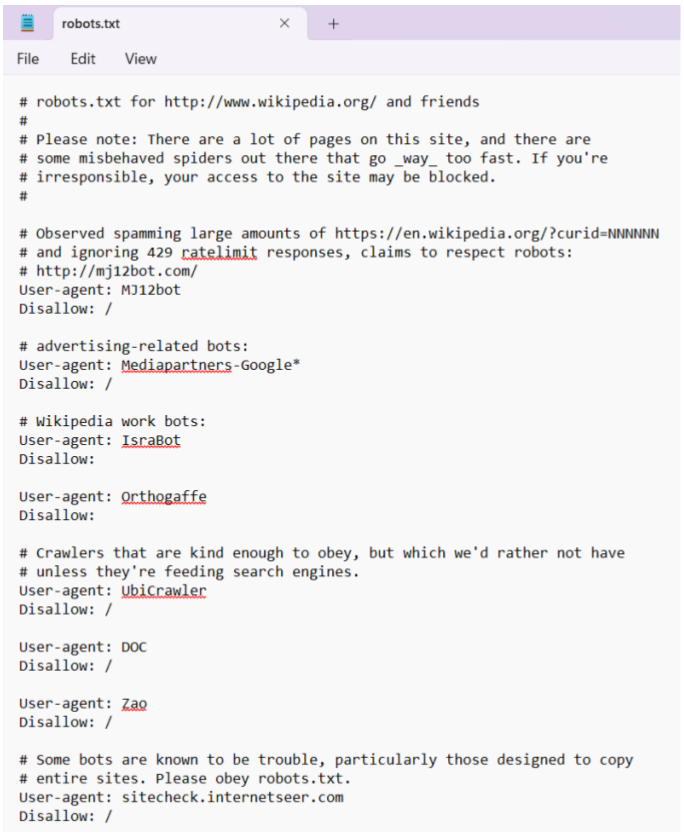
```
import requests

from bs4 import BeautifulSoup

import time

from urllib.parse import urljoin, urlparse

from urllib.robotparser import RobotFileParser

def get_html(url):

  headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/58.0.3029.110 Safari/537.3'}

  try:

    response=requests.get(url,headers=headers)

    response.raise_for_status()

    return response.text

  except requests.exceptions.HTTPError as errh:

    print(f"HTTP Error:{errh}")

  except requests.exceptions.RequestException as err:

    print(f"Request Error:{err}")

  return None

def save_robots_txt(url):

  try:

    robots_url=urljoin(url,'/robots.txt')

    robots_content= get_html(robots_url)

    if robots_content:

      with open('robots.txt','wb') as file:
            file.write(robots_content.encode('utf-8-sig'))
    except Exception as e:
        print(f"Error saving robots.txt: {e}")
def load_robots_txt():
    try:
        with open('robots.txt', 'rb') as file:
            return file.read().decode('utf-8-sig')
    except FileNotFoundError:
        return None
def extract_links(html,base_url):
    soup=BeautifulSoup(html,'html.parser')
    links=[]
    for link in soup.find_all('a',href=True):
        absolute_url=urljoin(base_url,link['href'])
        links.append(absolute_url)
    return links
def is_allowed_by_robots(url,robots_content):
    parser=RobotFileParser()
    parser.parse(robots_content.split('\n'))
    return parser.can_fetch('*',url)
def crawl(start_url,max_depth=3, delay=1):
    print("Hello")
    visited_urls=set()
    def recursive_crawl(url,depth, robots_content):
        if depth>max_depth or url in visited_urls or not is_allowed_by_robots(url,robots_content):
            return
        visited_urls.add(url)
        time.sleep(delay)
        html=get_html(url)
        if html:
            print(f"Crawling {url}")
            links=extract_links(html,url)
            for link in links:
                recursive_crawl(link,depth+1,robots_content)
    save_robots_txt(start_url)
    robots_content=load_robots_txt()
    if not robots_content:
        print("Unable to retrieve robots.txt. Crawling without  restrictions.")
    else:
        print("Using robots.txt for crawling restrictions.")

    recursive_crawl(start_url,1,robots_content)

print("Performed by Nupur Karpe")

crawl('https://', max_depth=2, delay=2)
```

OUTPUT:



```
Performed by Nupur Karpe
Hello
Using robots.txt for crawling restrictions.
Crawling https://wikipedia.com
Crawling https://en.wikipedia.org/
Crawling https://ja.wikipedia.org/
Crawling https://ru.wikipedia.org/
Crawling https://de.wikipedia.org/
Crawling https://es.wikipedia.org/
Crawling https://fr.wikipedia.org/
Crawling https://it.wikipedia.org/
Crawling https://zh.wikipedia.org/
Crawling https://fa.wikipedia.org/
Crawling https://pl.wikipedia.org/
Crawling https://ar.wikipedia.org/
Crawling https://arz.wikipedia.org/
Crawling https://nl.wikipedia.org/
Crawling https://pt.wikipedia.org/
Crawling https://ceb.wikipedia.org/
Crawling https://sv.wikipedia.org/
Crawling https://uk.wikipedia.org/
Crawling https://vi.wikipedia.org/
Crawling https://war.wikipedia.org/
Crawling https://af.wikipedia.org/
Crawling https://ast.wikipedia.org/
```

Robot.txt file



```
# robots.txt for http://www.wikipedia.org/ and friends
#
# Please note: There are a lot of pages on this site, and there are
# some misbehaved spiders out there that go _way_ too fast. If you're
# irresponsible, your access to the site may be blocked.
#

# Observed spamming large amounts of https://en.wikipedia.org/?curid=NNNNNN
# and ignoring 429 ratelimit responses, claims to respect robots:
# http://mj12bot.com/
User-agent: MJ12bot
Disallow: /

# advertising-related bots:
User-agent: Mediapartners-Google*
Disallow: /

# Wikipedia work bots:
User-agent: IsraBot
Disallow:

User-agent: Orthogaffe
Disallow:

# Crawlers that are kind enough to obey, but which we'd rather not have
# unless they're feeding search engines.
User-agent: UbiCrawler
Disallow: /

User-agent: DOC
Disallow: /

User-agent: Zao
Disallow: /

# Some bots are known to be trouble, particularly those designed to copy
# entire sites. Please obey robots.txt.
User-agent: sitecheck.internetseer.com
Disallow: /
```

PRACTICAL NO 8

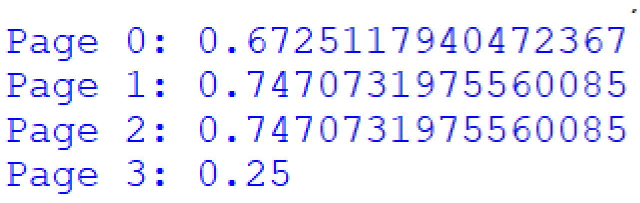
AIM: Link Analysis and PageRank

- Implement the PageRank algorithm to rank web pages based on link analysis.
- Apply the PageRank algorithm to a small web graph and analyze the results.

INPUT:

```
import numpy as np

def page_rank(graph, damping_factor=0.85, max_iterations=100, tolerance=1e-6):

    num_nodes = len(graph)

    page_ranks = np.ones(num_nodes) / num_nodes

    for _ in range(max_iterations):

        prev_page_ranks = np.copy(page_ranks)

        for node in range(num_nodes):

            incoming_links = [i for i, v in enumerate(graph) if node in v]

            if not incoming_links:

                continue

            page_ranks[node] = (1 - damping_factor) / num_nodes + \

                damping_factor * sum(prev_page_ranks[link] / len(graph[link]) for link in incoming_links)

        if np.linalg.norm(page_ranks - prev_page_ranks, 2) < tolerance:

            break

    return page_ranks

if __name__ == "__main__":

    web_graph = [

        [1, 2],

        [0, 2],

        [0, 1],

        [1, 2],

    ]

    result = page_rank(web_graph)

    for i, pr in enumerate(result):

        print(f"Page {i}: {pr}")
```

OUTPUT:



```
Page 0: 0.6725117940472367
Page 1: 0.747073197560085
Page 2: 0.747073197560085
Page 3: 0.25
```

PRACTICAL NO 9

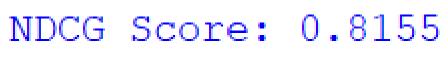
AIM: Learning to Rank

- Implement a learning to rank algorithm (e.g., RankSVM or RankBoost).
- Train the ranking model using labelled data and evaluate its effectiveness.

INPUT:

```
import numpy as np

from sklearn.svm import SVC

from sklearn.model_selection import train_test_split

from sklearn.metrics import ndcg_scor

# Example dataset

X = np.array([[3, 2, 1], [2, 1, 0], [0, 1, 2], [1, 2, 0], [2, 1, 3], [1, 0, 2]])

y = np.array([1, 0, 0, 1, 0, 1])


# Split data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Train RankSVM model

model = SVC(kernel='linear', C=1.0)

model.fit(X_train, y_train)


# Evaluate model

ndcg = ndcg_score([y_test], [model.predict(X_test)])

print(f"NDCG Score: {ndcg:.4f}")
```

OUTPUT:



```
NDCG Score: 0.8155
```

PRACTICAL NO 10

AIM : Advanced Topics in Information Retrieval

- Implement a text summarization algorithm (e.g., extractive or abstractive).
- Build a question-answering system using techniques such as information extraction

INPUT:

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from collections import Counter
def generate_summary(text, num_sentences=2):
    sentences = sent_tokenize(text)
    words = [word.lower() for word in word_tokenize(text) if word.isalnum()]
    words = [word for word in words if word not in stopwords.words("english")]
    word_freq = Counter(words)
    sentence_scores = {sent: sum(word_freq[word] for word in word_tokenize(sent.lower()) if word in word_freq) for sent in sentences}
    summary = ' '.join(sorted(sentence_scores, key=sentence_scores.get, reverse=True)[:num_sentences])
    return summary


# Example usage
text = """
Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages.
As such, NLP is related to the area of human–computer interaction.
Many challenges in NLP involve natural language understanding, that is, enabling computers to derive meaning from human or natural language input, and others involve natural language generation.
"""
print(generate_summary(text))
```

OUTPUT:

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\nupur\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\nupur\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Many challenges in NLP involve natural language understanding, that is, enabling computers t
o derive meaning from human or natural language input, and others involve natural language g
eneration.
Natural language processing (NLP) is a field of computer science, artificial intelligence, a
nd computational linguistics concerned with the interactions between computers and human (na
tural) languages.
```