Practical No: 1

Aim: Document Indexing and Retrieval

- **Implement an inverted index construction algorithm.**
- **Build a simple document retrieval system using the constructed index.**

Practical:

Input:

```
import nltk # Import NLTK to download stopwords
from nltk.corpus import stopwords  # Import stopwords from NLTK

# Define the documents
document1 = "The quick brown fox jumped over the lazy dog"
document2 = "The lazy dog slept in the sun"
# Get the stopwords for English language from NLTK
nltk.download('stopwords')
stopWords = stopwords.words('english')

# Step 1: Tokenize the documents
# Convert each document to lowercase and split it into words
tokens1 = document1.lower().split()
tokens2 = document2.lower().split()

# Combine the tokens into a list of unique terms
terms = list(set(tokens1 + tokens2))

# Step 2: Build the inverted index
# Create an empty dictionary to store the inverted index as well as a dictionary
to store number of occurrences
inverted_index = {}
occ_num_doc1 = {}
occ_num_doc2 = {}

# For each term, find the documents that contain it
```

```
for term in terms:
    if term in stopWords:
        continue
    documents = []
    if term in tokens1:
        documents.append("Document 1")
        occ_num_doc1[term] = tokens1.count(term)
    if term in tokens2:
        documents.append("Document 2")
        occ_num_doc2[term] = tokens2.count(term)

inverted_index[term] = documents

# Step 3: Print the inverted index
for term, documents in inverted_index.items():
    print(term, "->", end=" ")
    for doc in documents:
        if doc == "Document 1":
            print(f"{doc} ({occ_num_doc1.get(term, 0)}),", end=" ")
        else:
            print(f"{doc} ({occ_num_doc2.get(term, 0)}),", end=" ")
    print()
print("Performed by 740_Pallavi & 743_Deepak")
```

Output:

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\deepa\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
quick -> Document 1 (1),
lazy -> Document 1 (1), Document 2 (1),
sun -> Document 2 (1),
jumped -> Document 1 (1),
fox -> Document 1 (1),
slept -> Document 2 (1),
dog -> Document 1 (1), Document 2 (1),
brown -> Document 1 (1),
Performed by 740_Pallavi & 743_Deepak
```

Practical No: 2

Aim: Retrieval Models

- **Implement the Boolean retrieval model and process queries.**
- **Implement the vector space model with TF-IDF weighting and cosine similarity.**

Practical:

A) Implement the Boolean retrieval model and process queries:

Input:

```
documents = {
    1: "apple banana orange",
    2: "apple banana",
    3: "banana orange",
    4: "apple"
}

# Function to build an inverted index using dictionaries
def build_index(docs):
    index = {}  # Initialize an empty dictionary to store the inverted index
    for doc_id, text in docs.items():  # Iterate through each document and its text
        terms = set(text.split())  # Split the text into individual terms
        for term in terms:  # Iterate through each term in the document
            if term not in index:
                index[term] = {doc_id}  # If the term is not in the index, create a new
set with document ID
            else:
                index[term].add(doc_id)  # If the term exists, add the document ID to
its set
    return index  # Return the built inverted index

# Building the inverted index
inverted_index = build_index(documents)
```

```
# Function for Boolean AND operation using inverted index
def boolean_and(operands, index):
    if not operands:  # If there are no operands, return all document IDs
        return list(range(1, len(documents) + 1))

    result = index.get(operands[0], set())  # Get the set of document IDs for the
first operand
    for term in operands[1:]:  # Iterate through the rest of the operands
        result = result.intersection(index.get(term, set()))  # Compute intersection
with sets of document IDs
    return list(result)  # Return the resulting list of document IDs

# Function for Boolean OR operation using inverted index
def boolean_or(operands, index):
    result = set()  # Initialize an empty set to store the resulting document IDs
    for term in operands:  # Iterate through each term in the query
        result = result.union(index.get(term, set()))  # Union of sets of document
IDs for each term
    return list(result)  # Return the resulting list of document IDs

# Function for Boolean NOT operation using inverted index
def boolean_not(operand, index, total_docs):
    operand_set = set(index.get(operand, set()))  # Get the set of document IDs
for the operand
    all_docs_set = set(range(1, total_docs + 1))  # Create a set of all document
IDs
    return list(all_docs_set.difference(operand_set))  # Return documents not in
the operand set

# Example queries
query1 = ["apple", "banana"]  # Query for documents containing both "apple"
```

and "banana"
query2 = ["apple", "orange"] # Query for documents containing "apple" or "orange"

# Performing Boolean Model queries using inverted index
result1 = boolean_and(query1, inverted_index) # Get documents containing both terms
result2 = boolean_or(query2, inverted_index) # Get documents containing either of the terms
result3 = boolean_not("orange", inverted_index, len(documents)) # Get documents not containing "orange"

# Printing results
print("Documents containing 'apple' and 'banana':", result1)
print("Documents containing 'apple' or 'orange':", result2)
print("Documents not containing 'orange':", result3)
print("Performed by 740_Pallavi & 743_Deepak")

Output:

```
Documents containing 'apple' and 'banana': [1, 2]
Documents containing 'apple' or 'orange': [1, 2, 3, 4]
Documents not containing 'orange': [2, 4]
Performed by 740_Pallavi & 743_Deepak
```

B) Implement the vector space model with TF-IDF weighting and cosine similarity:

Input:

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
# Import necessary libraries
import nltk # Import NLTK to download stopwords
from nltk.corpus import stopwords # Import stopwords from NLTK
import numpy as np # Import NumPy library
from numpy.linalg import norm # Import norm function from NumPy's linear algebra module
# Define the training and test sets of text documents

```
train_set = ["The sky is blue.", "The sun is bright."] # Documents
test_set = ["The sun in the sky is bright."] # Query

# Get the stopwords for English language from NLTK
nltk.download('stopwords')
stopWords = stopwords.words('english')

# Initialize CountVectorizer and TfidfTransformer objects
vectorizer = CountVectorizer(stop_words=stopWords) # CountVectorizer to
convert text to matrix of token counts
transformer = TfidfTransformer() # TfidfTransformer to convert matrix of
token counts to TF-IDF representation

# Convert the training and test sets to arrays of TF-IDF features
trainVectorizerArray = vectorizer.fit_transform(train_set).toarray() # Fit-
transform training set
testVectorizerArray = vectorizer.transform(test_set).toarray() # Transform test
set

# Display the TF-IDF arrays for training and test sets
print('Fit Vectorizer to train set', trainVectorizerArray)
print('Transform Vectorizer to test set', testVectorizerArray)

# Define a lambda function to calculate cosine similarity between vectors
cx = lambda a, b: round(np.inner(a, b) / (norm(a) * norm(b)), 3)

# Iterate through each vector in the training set
for vector in trainVectorizerArray:
    print(vector) # Display each vector in the training set
    # Iterate through each vector in the test set
    for testV in testVectorizerArray:
        print(testV) # Display each vector in the test set
```

```
    cosine = cx(vector, testV)  # Calculate cosine similarity between vectors
    print(cosine)  # Display the cosine similarity

# Fit the transformer to the training set and transform it to TF-IDF
representation
transformer.fit(trainVectorizerArray)
print()
print(transformer.transform(trainVectorizerArray).toarray())

# Fit the transformer to the test set and transform it to TF-IDF representation
transformer.fit(testVectorizerArray)
print()
tfidf = transformer.transform(testVectorizerArray)
print(tfidf.todense())
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\deepa\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]]
Performed by 740_Pallavi & 743_Deepak
```

Output:

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.**

Practical:

Input:

```
# A Naive recursive python program to find minimum number
# operations to convert str1 to str2
def editDistance(str1, str2, m, n):
    # If first string is empty, the only option is to insert all characters of second
string into first
    if m == 0:
        return n
    # If second string is empty, the only option is to remove all characters of first
string
    if n == 0:
        return m
    # If last characters of two strings are same, nothing much to do. Ignore last
characters and get count for remaining strings.
    if str1[m-1] == str2[n-1]:
        return editDistance(str1, str2, m-1, n-1)
    # If last characters are not same, consider all three operations on last
character of first string, recursively compute minimum cost for all three
operations and take minimum of three values.
    return 1 + min(editDistance(str1, str2, m, n-1), # Insert
                   editDistance(str1, str2, m-1, n), # Remove
                   editDistance(str1, str2, m-1, n-1) # Replace)

# Driver code
str1 = "sunday"
str2 = "saturday"
print('Edit Distance is: ', editDistance(str1, str2, len(str1), len(str2)))
```

Output:

```
PS C:\Users\Administrator\Documents\Sem 6\IR>
Edit Distance is:  3
Performed by 740 Pallavi & 743 Deepak
```

Practical No: 4

Aim: Evaluation Metrics for IR Systems

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

Practical:

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

Input:

```
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))
    '''
    (Optional)
    PPT values:
    true_positive = 20
    false_positive = 10
    false_negative = 30
    '''
    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}")
```

Output:

```
PS C:\Users\Administrator\Documents\Se
cuments/Sem 6/IR/prac4_1.py"
True Positive:  1
False Positive:  2
False Negative:  1

Precision: 0.3333333333333333
Recall: 0.5
F-measure: 0.4
Performed by 740 Pallavi & 743 Deepak
```

B) 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:

```
PS C:\Users\Administrator\Documents\Sem 6\IR> & C:/
cuments/Sem 6/IR/prac4_2.py"
Average precision-recall score: 0.8041666666666667
Performed by 740 Pallavi & 743 Deepak
```

Practical No: 5

Aim: Text Categorization

A) **Implement a text classification algorithm (e.g., Naive Bayes or Support Vector Machines).**

B) **Train the classifier on a labelled dataset and evaluate its performance.**

Practical:

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
# Load the CSV file
df = pd.read_csv(r"C:\Users\Administrator\Documents\Sem 6\IR\Dataset.csv")
data = df["covid"] + "" + df["fever"]
X = data.astype(str)      # Test data
y = df['flu']             # Labels
# Splitting the data into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 42)
# Converting data into bag-of-data format to train the model
vectorizer = CountVectorizer()
# initializing the converter
X_train_counts = vectorizer.fit_transform(X_train)
# converting the training data
X_test_counts = vectorizer.transform(X_test)
# converting the test data
# using and training the multinomial model of naive bayes algorithm
classifier = MultinomialNB()            # initializing the classifier
classifier.fit(X_train_counts, y_train) # training the classifier
```

```
# loading another dataset to test if the model is working properly
data1 = pd.read_csv(r"C:\Users\Administrator\Documents\Sem 6\IR\Test.csv")
new_data = data1["covid"] + "" + data1["fever"]
new_data_counts = vectorizer.transform(new_data.astype(str))      # converting
the new data
# making the model to predict the results for new dataset
predictions = classifier.predict(new_data_counts)
# Output the results
new_data = predictions
print(new_data)
# retrieving the accuracy and classification report
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)))
# Convert the predictions to a DataFrame
predictions_df = pd.DataFrame(predictions, columns = ['flu_prediction'])
# concatenate the original DataFrame with the predictions DataFrame
data1 = pd.concat([data1, predictions_df], axis = 1)
# write the DataFrame back to CSV data1.to_csv(r"C:
\Users\Administrator\Documents\Sem 6\IR\Test1.csv", index
= False)
```

Output:

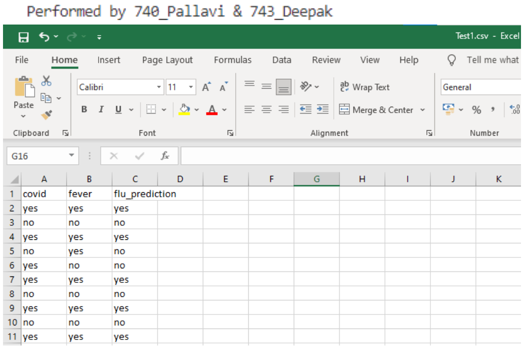
```
Documents/Sem 6/IR/prac_5.py"
['yes' 'no' 'yes' 'no' 'no' 'yes' 'no' 'yes' 'no' 'yes']

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

Performed by 740_Pallavi & 743_Deepak
```



	A	B	C
1	covid	fever	flu_prediction
2	yes	yes	yes
3	no	no	no
4	yes	yes	yes
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	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.**

Practical

Input:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
documents = ["Cats are known for their agility and grace", #cat doc1
            "Dogs are often called 'man's best friend'.", #dog doc1
            "Some dogs are trained to assist people with disabilities.", #dog doc2
            "The sun rises in the east and sets in the west.", #sun doc1
            "Many cats enjoy climbing trees and chasing toys.", #cat doc2
            ]
# Create a TfidfVectorizer object
vectorizer = TfidfVectorizer(stop_words='english')
# Learn vocabulary and idf from training set.
X = vectorizer.fit_transform(documents)
# Perform k-means clustering
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
# Print cluster labels for each document
print(kmeans.labels_)
```

Output:

```
[0 1 1 2 0]
Performed by 740_Pallavi & 743_Deepak
```

Practical No: 7

Aim: Web Crawling and Indexing

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

Practical

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):
    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.")

    recursive_crawl(start_url, 1, robots_content)

# Example usage:
print("Performed by 740_Pallavi & 743_Deepak") crawl('https://
wikipedia.com', max_depth=2, delay=2)
```

Output:

```
Performed by 740_Pallavi & 743_Deepak
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/
```

robot.txt file:

robots.txt

robots.txt

```
# 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: /
```

Practical No: 8

Aim: Link Analysis and PageRank

A) Implement the PageRank algorithm to rank web pages based on link analysis.

B) Apply the PageRank algorithm to a small web graph and analyse the results.

Practical

Input:

```
import numpy as np


def page_rank(graph, damping_factor=0.85, max_iterations=100, tolerance=1e-
6):
    # Get the number of nodes
    num_nodes = len(graph)
    # Initialize PageRank values
    page_ranks = np.ones(num_nodes) / num_nodes
    # Iterative PageRank calculation
    for _ in range(max_iterations):
        prev_page_ranks = np.copy(page_ranks)
        for node in range(num_nodes):
            # Calculate the contribution from incoming links
            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)


        # Check for convergence
        if np.linalg.norm(page_ranks - prev_page_ranks, 2) < tolerance:
            break
```

```
    return page_ranks

# Example usage
if __name__== "__main__":
    # Define a simple directed graph as an adjacency list
    # Each index represents a node, and the list at that index contains nodes to
which it has outgoing links
    web_graph = [
        [1, 2],     # Node 0 has links to Node 1 and Node 2
        [0, 2],     # Node 1 has links to Node 0 and Node 2
        [0, 1] , # Node 2 has links to Node 0 and Node 1
        [1,2],  # Node 3 has links to Node 1 and Node 2
    ]

    # Calculate PageRank
    result = page_rank(web_graph)

    # Display PageRank values
    for i, pr in enumerate(result):
        print(f"Page {i}: {pr}")
```

Output:

```
Page 0: 0.6725117940472367
Page 1: 0.7470731975560085
Page 2: 0.7470731975560085
Page 3: 0.25
Performed by 740_Pallavi & 743_Deepak
```