JNAN VIKAS MANDAL'S
PADMASHREE DR. R.T.DOSHI DEGREE COLLEGE OF INFORMATION TECHNOLOGY MOHANLAL RAICHAND MEHTA COLLEGE OF COMMERCE DIWALIMAA DEGREE COLLEGE OF SCIENCE

CERTIFICATE

This is to certify that the Mr./Miss._____of T.Y.B.Sc.(CS) Semester-V has completed the practical work in the subject of **INFORMATION RETRIEVAL** during the Academic year 2024-2025 under the guidance of **Mrs. Sarita Sarang.** being the partial requirement for the fulfilment of the curriculum of Degree of Bachelor of Science in Computer Science, University of Mumbai.

Place: **Date:**

Sign of Subject Incharge

Sign of External Examiner

Sign of In charge / H.O.

INDEX

Sr. No	Topic	Dates	Sign.
1	Document Indexing and Retrieval • Implement an inverted index construction algorithm. • Build a simple document retrieval system using the constructed index.		
2	Retrieval Models • Implement the Boolean retrieval model and process queries. • Implement the vector space model with TF-IDF weighting and cosine similarity.		
3	Spelling Correction in IR Systems • Develop a spelling correction module using edit distance algorithms. • Integrate the spelling correction module into an information retrieval system.		
4	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.		
5	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.		
6	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.		
7	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.		
8	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.		

Practical No: 1

Aim: Document Indexing and Retrieval

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

Theory :

- An Inverted Index is a data structure used in information retrieval systems to efficiently retrieve documents or web pages containing a specific term or set of terms.
- In an inverted index, the index is organised by terms (words), and each term points to a list of documents or web pages that contain that term.
- Inverted indexes are widely used in search engines, database systems, and other applications where efficient text search is required.
- They are especially useful for large collections of documents, where searching through all the documents would be prohibitively slow. An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents.

Rules to create an inverted index -

1) The text of each document is first preprocessed by removing stop words : Stop words are the most occurring and useless words in documents like “I”, “the”, “we”, “is”, and “an”.
2) The text is tokenized, meaning that it is split into individual terms.
3) The terms are then added to the index, with each term pointing to the documents in which it appears

Practical:

(A) document1 ="The quick brown fox jumped over the lazy dog." document2 = "The lazy dog slept in the sun."

tokens1 = document1.lower().split() tokens2 = document2.lower().split() terms = list(set(tokens1 + tokens2))

inverted_index = {} for term in terms:
documents = []

if term in tokens1:

documents.append("Documents 1")
if term in tokens2:

documents.append("Documents 2")
inverted_index[term] = documents

for term, documents in inverted_index.items():
print(term, "->",",".join(documents))

Output:

```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python311/IR_PT1.p
y
slept -> Documents 2
dog -> Documents 2
dog. -> Documents 1
the -> Documents 1,Documents 2
over -> Documents 1
fox -> Documents 1
brown -> Documents 1
lazy -> Documents 1,Documents 2
jump -> Documents 1
in -> Documents 2
quick -> Documents 1
sun. -> Documents 2
```

File.txt

B) # this will open the file file =
open('file.txt', encoding='utf8') read =
file.read() file.seek(0) read

to obtain the

number of lines

in file line = 1 for

word in read: if

word == '\n':

line += 1

print("Number of lines in file is: ", line)

create a list to

store each line as # an element

of list array = [] for i in

range(line):

array.append(file.readline())

Output:

```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python311/IT_PT!(B)
.py
Number of lines in  file is : 3
```

(C) from nltk.tokenize import

word_tokenize import nltk from nltk.corpus

import stopwords

nltk.download('stopwords')

nltk.download('punkt')

for i in range(1):

    read="This is a simple sentence"

text_tokens = word_tokenize(read)

tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]
print(tokens_without_sw)

Output:

```
======= RESTART: C:\Users\sahil\OneDrive\Desktop\TY PRACTICAL\IR\
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\sahil\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\sahil\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
['This', 'simple', 'sentence']
>>
```

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.**

A) 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():
        terms = set(text.split())
        for term in terms:
            if term not in index:
                index[term] = {doc_id} # Set with document ID
            else:
                index[term].add(doc_id) # Add to existing set
        return index
inverted_index = build_index(documents)

def boolean_and(operands, index):
    if not operands:
        return list(range(1, len(documents) + 1))
    result = index.get(operands[0], set()) # First operand
    for term in operands[1:]:
        result = result.intersection(index.get(term, set())) # Intersection with sets of
document IDs
    return list(result)
def boolean_or(operands, index):
```

```
    result = set()
    for term in operands:
        result = result.union(index.get(term, set()))
    return list(result)

def boolean_not(operand, index, total_docs):
    operand_set = set(index.get(operand, set())) # Set for the operand
    all_docs_set = set(range(1, total_docs + 1)) # IDs for all documents
    return list(all_docs_set.difference(operand_set)) # Return documents not in the
operand set
query1 = ["apple", "banana"] # Query for documents containing both "apple" and
"banana"
query2 = ["apple", "orange"] # Query for documents containing "apple" or "orange
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"

print("Documents containing 'apple' and 'banana':", result1)
print("Documents containing 'apple' or 'orange':", result2)
print("Documents not containing 'orange':", result3)
```

Output:

```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python311/IR_PT2.p
y
Documents containing 'apple' and 'banana': [1]
Documents containing 'apple' or 'orange': [1]
Documents not containing 'orange': [2, 3, 4]
```

B) 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."] # Documents test_set = ["The sun in the sky is bright."] # Query


nltk.download('stopwords') stopWords = stopwords.words('english')


vectorizer = CountVectorizer(stop_words=stopWords) transformer = TfidfTransformer()


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(f"Cosine similarity:
{cosine}") **Output:**

```
======= RESTART: C:\Users\sahil\OneDrive\Desktop\TY PRACTICAL\IR\2ndb.py =======
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\sahil\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 0 1]
[0 1 1 1]
Cosine similarity: 0.816
```

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.**

Theory:

Edit Distance :

- Edit distance is a measure of the similarity between two strings by calculating the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into the other.
  - The smaller the edit distance, the more similar the strings are.

Consider two strings str1 and str2 of length M and N respectively.
For finding edit distance there are performed below operations -

1. Operation 1 (INSERT): Insert any character before or after any index value
2. Operation 2 (REMOVE): Remove a character
3. Operation 3 (Replace): Replace a character at any index value with some other character

Practical:

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:

```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python311/IR_PT3.p
y
 Edit Distance is: 4
```

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.

Theory:

1. Precision:

- Precision is the ratio of correctly predicted positive observations to the total predicted positives.
- It is also called Positive Predictive Value (PPV).
- Precision is calculated using the following formula:

Precision = TP / TP+FP

Where:

• TP (True Positives) is the number of instances correctly predicted as positive. • FP (False Positives) is the number of instances incorrectly predicted as positive.

High precision indicates that the model has a low rate of false positives. In other words, when the model predicts a positive result, it is likely to be correct.

2. Recall:

- Recall is the ratio of correctly predicted positive observations to all observations in actual class.
- Recall is calculated using the following formula:

  Recall= TP /TP+FN Where:

- TP (True Positives) is the number of instances correctly predicted as positive. • FN (False Negatives) is the number of instances incorrectly predicted as negative.

High recall indicates that the model has a low rate of false negatives. In other words, the model is effective at capturing all the positive instances.

Confusion Matrix :

	Predicted No	Predicted Yes
Actual No	True Negative	False Positive
Actual Yes	False Negative	True Positive

3. F-measure:

- The F-measure is a metric commonly used in performance evaluation.
- It combines precision and recall into a single value, providing a balanced measure of a model's performance.

- The formula for F-measure is:

$$F - measure = 2 * \frac{Precision * Recall}{Precesion + Recall}$$

- The F-measure ranges from 0 to 1, where 1 indicates perfect precision and recall.

4. Average Precision:

Average Precision is used to find the Average of the model precision based on relevancy of result. • Algorithm:

In order to find Average Precision:

1) Take 2 variables X and Y as 0
2) We will then go through the prediction from left to right: 3) In case the prediction is 0, we will only increment Y by 1 and not find prediction score
4) In case the prediction is 1, we will increment both X and Y by 1 5) After incrementing, we use the formula X/Y to get the current position prediction score.
6) Lastly we will find summation of all prediction scores and divide them by total number of positive predictions.

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))

    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:

```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python311/IR_PT4(A
).py
True Positive: 1
False Positive: 2
False Negative: 1

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

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]


average_precision = average_precision_score(y_true, y_scores)


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

output:

```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python311/IR_PT4(B
).py
Average precision-recall score: 0.8041666666666667
```

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.

Theory:

Naive Bayes

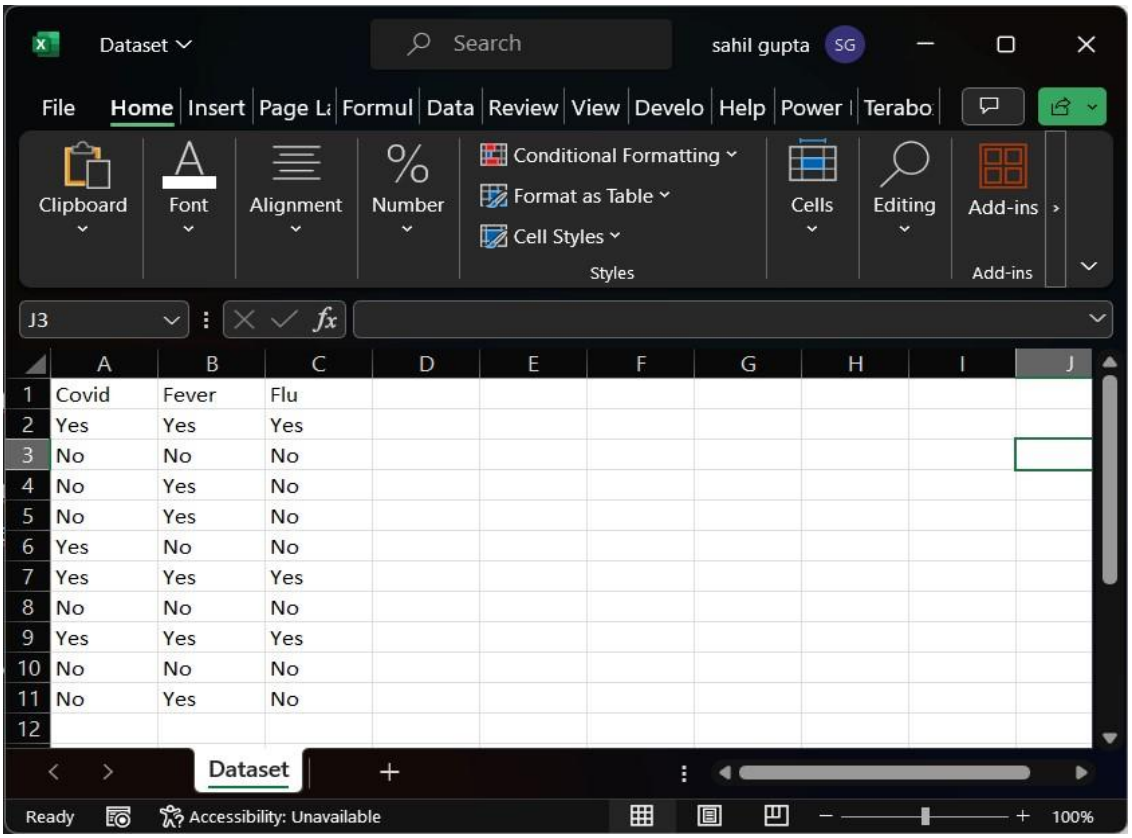
- The Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naive Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naive Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Bayes' Theorem:

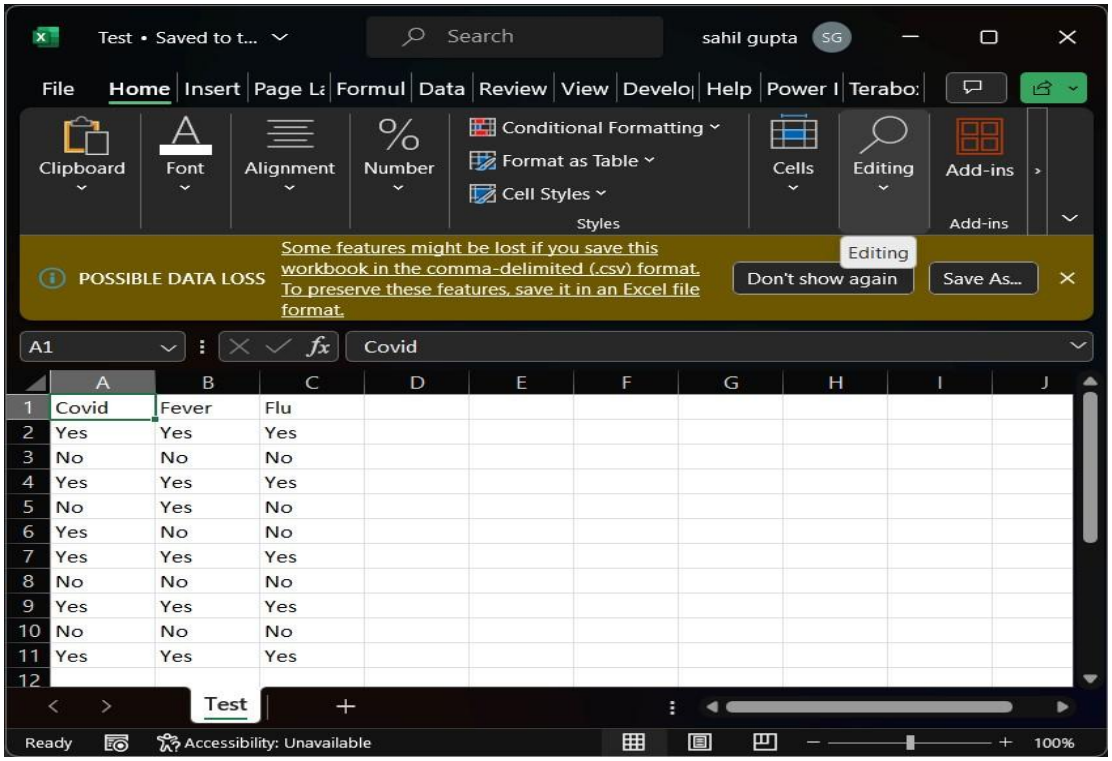
- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as: $P(A|B) = \frac{P(B|A) * P(A)}{\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}$

P(B) **Create two CSV file:**

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
12			



	A	B	C
1	Covid	Fever	Flu
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
12			

Practical:

Input:

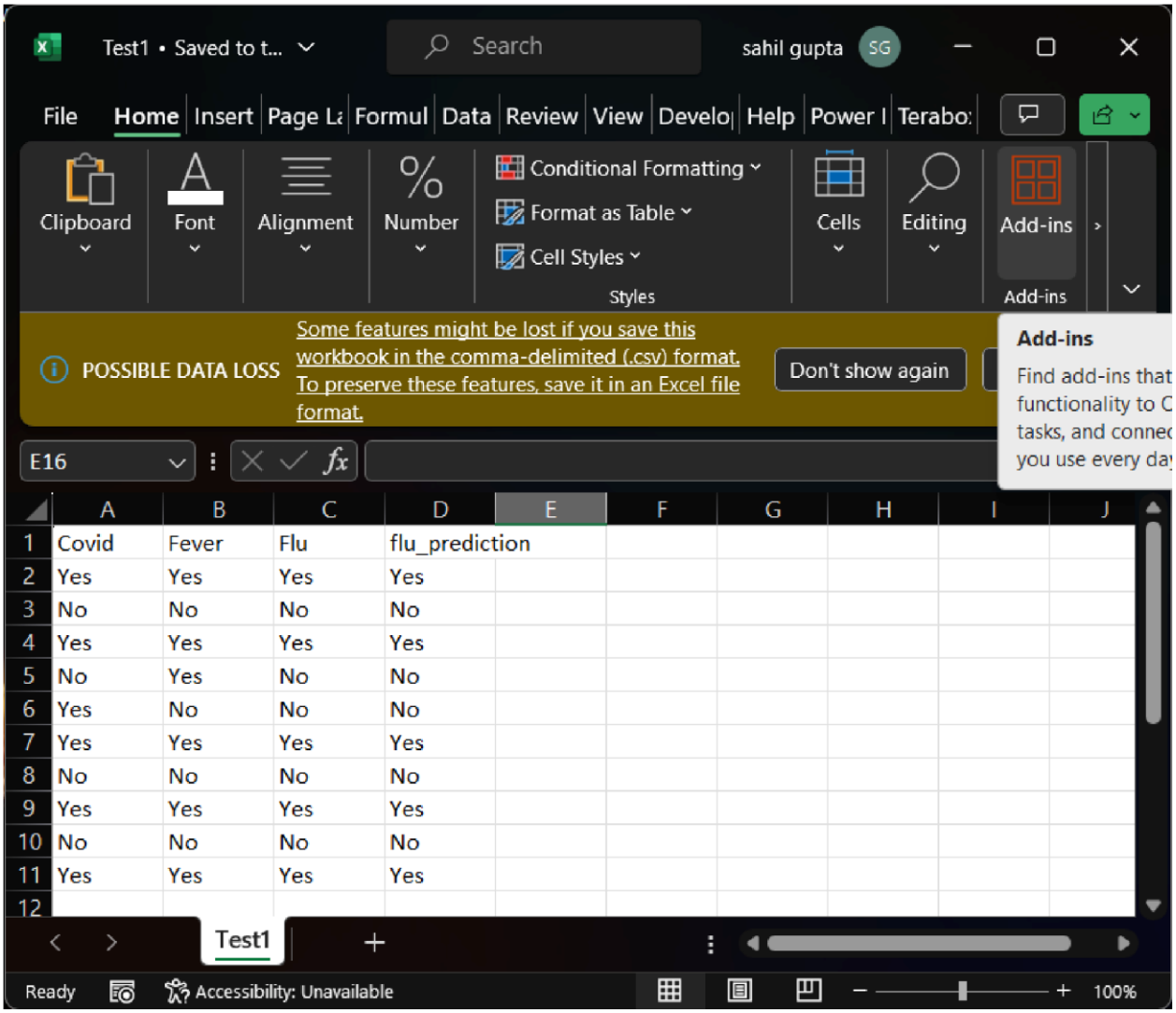
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

df = pd.read_csv(r"C:\Users\admin\Desktop\TYCS-AI Prct\dataSet1.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\admin\Desktop\TYCS-AI Prct\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\admin\Desktop\TYCS-AI Prct\Test1.csv",index=False)
```

Output:

```
= RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python311/IR_PT5.p
y
['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
```

Flu_prediction



	A	B	C	D	E	F	G	H	I	J
1	Covid	Fever	Flu	flu_prediction						
2	Yes	Yes	Yes	Yes						
3	No	No	No	No						
4	Yes	Yes	Yes	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						
10	No	No	No	No						
11	Yes	Yes	Yes	Yes						
12										

Practical No: 6

Aim: Clusterting 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.

Theory:

K-Means Clustering:

- K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters.
- Here K defines the number of predefined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.
- It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
- The main aim of this algorithm is to minimise the sum of distances between the data point and their corresponding clusters.
- The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

- The k-means clustering algorithm mainly performs two tasks:

    1. Determines the best value for K centre points or centroids by an iterative process.
    2. Assigns each data point to its closest k-centre. Those data points which are near to the particular k-centre, create a cluster.

Working of K-means Algorithm -

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be different from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.
Step-4: Calculate the variance and place a new centroid of each cluster.
Step-5: Repeat the third steps, which means assign each datapoint to the new closest centroid of each cluster.
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Practical
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 dogs are trained to assist people with disabilities.",

"The sun rises in the east and sets in 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:

```
= RESTART: C:\Users\admin\AppData\Local\Programs\Python\Python37-32\prac6.py =
[0 1 1 2 0]
```

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.**

Theory

Crawling: Google downloads text, images, and videos from pages it found on the internet with automated programs called crawlers.

Indexing: Google analyses the text, images, and video files on the page, and stores the information in the Google index, which is a large database.

Crawling Process -

1) **Starting Point:** The crawling process usually begins with a set of seed URLs, which can be provided manually or generated through algorithms. These URLs serve as the starting points for the web crawlers.

2) **Queue of URLs:** Web crawlers maintain a queue of URLs, often referred to as the URL frontier. This queue represents the set of URLs that the crawler is yet to visit. New URLs are continuously added to this queue during the crawling process.

3) **Parsing Content:** When a web crawler visits a webpage, it parses the HTML content to extract links (URLs) embedded within the page. This process involves examining HTML tags and, in some cases, executing JavaScript to discover additional links.

4) **Respecting Directives:** Crawlers adhere to rules specified in the robots.txt file, which indicates areas of a website that should not be crawled. Additionally, crawlers may implement policies to filter out certain types of content or URLs.

5) **Avoiding Redundancy:** To prevent redundancy and ensure efficient crawling, duplicate URLs are often identified and removed from the crawling queue.

6) **Traversal Strategy**: Crawlers can follow various traversal strategies, such as depth-first or breadth-first, as they explore the web. The chosen strategy determines the order in which pages are crawled.

7) **Politeness:** To avoid overloading web servers with too many requests, crawlers may introduce a crawl delay between successive requests.

8) **Retrieving Web Pages:** Crawlers download the content of web pages, including HTML, text, images, and other resources. The downloaded content is then processed for indexing.

Indexing Process -

1) **HTML Parsing:** The content retrieved by the crawler is parsed to extract relevant information. This involves analysing the HTML structure to identify text, metadata, and other elements on the page.

2) **Isolating Textual Content:** From the parsed content, the crawler isolates the textual information, such as the body of the page, headings, and other relevant textual data.

3) **Breaking into Tokens**: The textual content is tokenized, breaking it down into smaller units, typically words or phrases. Tokenization facilitates efficient indexing and retrieval based on keywords.

4) **Data Organization:** The extracted information is organized into an index, which is a structured database allowing for fast and efficient retrieval. The index includes details about keywords, their locations, and other relevant metadata.

5) **Optimising for Retrieval:** The index is often organised as an inverted index, mapping each term to the documents or web pages where it

appears. This structure enables quick retrieval of documents containing specific terms.

6) **Additional Information:** In addition to textual content, metadata such as page title, URL, and other relevant details may be included in the index to enhance the search experience.
7) **Real-time Changes:** Search engines continuously update their indexes to reflect changes on the web. This ensures that the search results remain current and accurate.

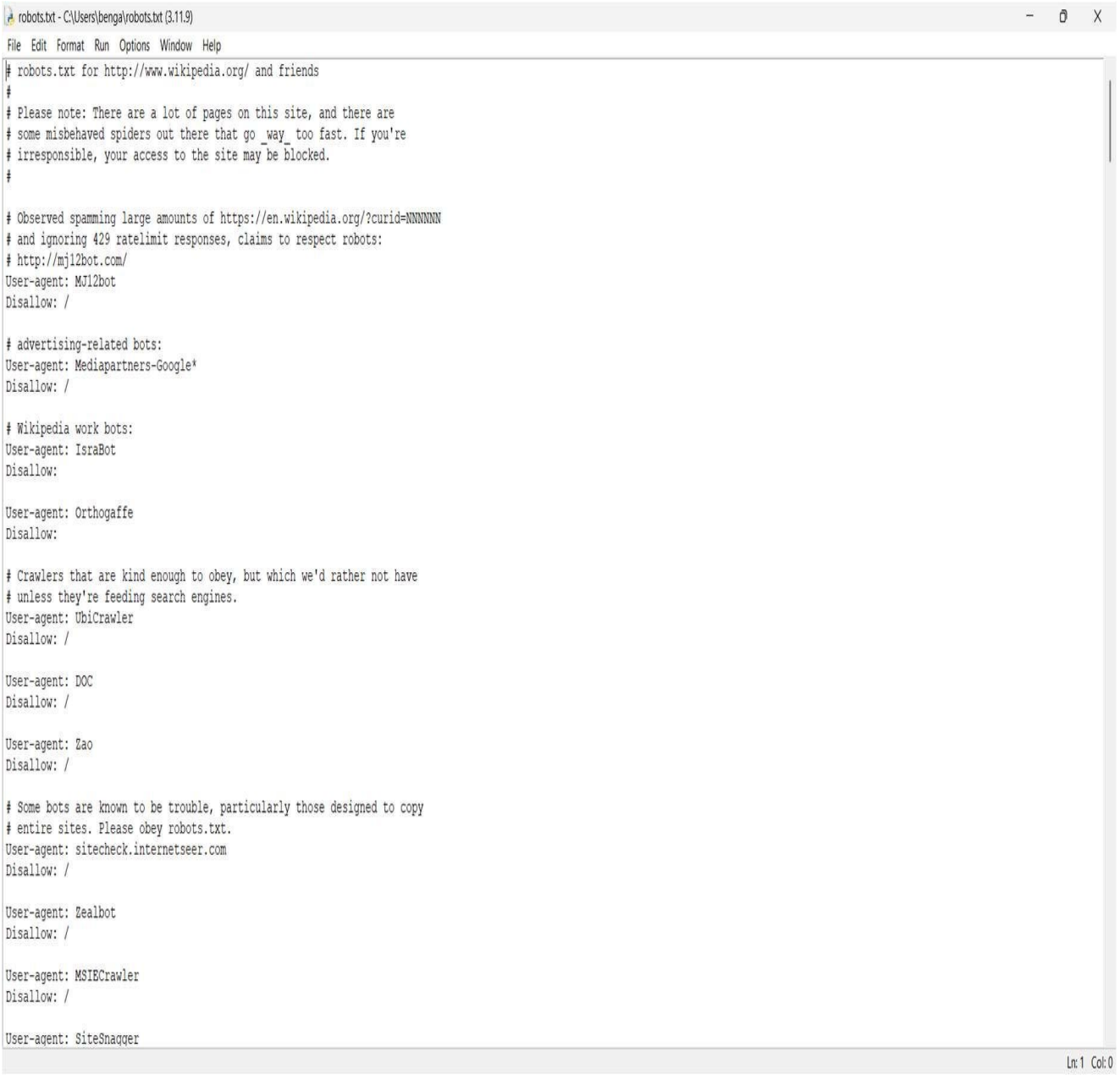
Pratical Input:

```
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("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.get('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_conte
        nt)

  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)
  print("Performed by Raj")
crawl("https://wikipedia.com",max_depth=2,delay=2)
```

Output:


robots.txt - C:\Users\benga\robots.txt (3.11.9)
File Edit Format Run Options Window Help
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: /
User-agent: Zealbot
Disallow: /
User-agent: MSIECrawler
Disallow: /
User-agent: SiteSnagger
Ln: 1 Col: 0

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.**

Theory

Link Analysis:

- Link analysis is a method used to examine relationships and connections between entities in a network.
- It involves studying the links or connections between different elements to uncover patterns, structures, and insights.
- Link analysis is commonly applied in various fields, including information retrieval, social network analysis, fraud detection, and recommendation systems.

PageRank Algorithm -

- The PageRank algorithm is an algorithm used by the Google search engine to rank web pages in its search results.
- It was developed by Larry Page and Sergey Brin, the co-founders of Google, and is named after Larry Page.
- PageRank is based on the idea that the importance of a webpage is determined by the number and quality of other pages that link to it.

Working -

1) **Initialize PageRank Values:**
   Set an initial PageRank value for each node. Commonly, this is initialised to 1 divided by the total number of nodes, making the sum of all PageRank values equal to 1.

2) **Define Damping Factor and Iterations:**
   Choose a damping factor (typically 0.85) to model the probability that a user will continue navigating through the web by following links. Decide on the maximum number of iterations for the algorithm.

3. Iterative PageRank Calculation:

- For each iteration, update the PageRank values for each node based on the PageRanks of the nodes linking to it.
- Use the PageRank formula:

$$PR(i) = (1 - d) + d \left(\frac{PR(1)}{L(1)} + \frac{PR(2)}{L(2)} + \ldots + \frac{PR(n)}{L(n)} \right)$$

where:

- $PR(i)$ is the PageRank of node i,
- d is the damping factor,
- $PR(j)$ is the PageRank of node j linking to node i,
- $L(j)$ is the number of outgoing links from node j, and
- n is the total number of nodes.

1) Convergence Check:
After each iteration, check for convergence. If the difference between the new and previous PageRank values falls below a certain threshold, the algorithm has converged, and you can stop iterating.

2) Repeat Iterations:
Continue iterating until the maximum number of iterations is reached or until convergence is achieved.

3) Final PageRank Values:
The final PageRank values represent the importance of each node in the graph based on the link structure.

Practical

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  # Initialize page ranks

    for _ in range(max_iterations):
        prev_page_ranks = np.copy(page_ranks)

        # Update each node's page rank based on incoming links
        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]) + 1) for
link in incoming_links)

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

    return page_ranks

if __name__ == "__main__":
    web_graph = [
        [1, 2],  # Node 0 links to Node 1 and Node 2
        [0, 2],  # Node 1 links to Node 0 and Node 2
        [0, 1],  # Node 2 links to Node 0 and Node 1
        [1, 2],  # Node 3 links to Node 1 and Node 2
    ]

    result = page_rank(web_graph)
```

```
    for i, pr in enumerate(result):
        print(f"Page rank of node {i}: {pr:.4f}")
```

Output:

```
RESTART: C:\Users\admin\AppData\Local\Programs\Python\Python37-32\IRpract8.py
Page rank of node 0: 0.1587
Page rank of node 1: 0.2139
Page rank of node 2: 0.2139
Page rank of node 3: 0.2500
```