Web Data Analytics Practical

Subject Code :- 57773

A Practical Report Submitted in

fulfillment of the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

Year 2024 – 2025

By

Ms. Mansi Sanjay Jadhav

(Application Id :-

MU0027920240004612)

Seat Number :- 8176101

Semester II

Under the Guidance of

Prof. Sujatha Iyer

Centre for Distance and Online

Education

Vidya Nagari, Kalina, Santacruz East

400098.

University of Mumbai

PCP Center [Satish Pradhan

Dnyanasadhana College,Thane 400604]

Centre for Distance and Online Education

Vidya Nagari, Kalina, Santacruz East 400098.

CERTIFICATE

This is to certify that

Ms. Mansi Sanjay Jadhav , Application Id :- MU0027920240004612 ,

Student of Master of Science in Computer Science has Satisfactorily

Completed the Practical in

Web Data Analytics

Name	Application Id
Ms Mansi Sanjay Jadhav	MU0027920224000612

Subject In-Charge

Examiner

Sr. No.	Practical	Date	Page No.	Sign
1	Page Rank for link analysis using python			
2	Program to Perform Spam Classifier			
3	Demonstrate Text Mining and Webpage Pre-processing using meta information from the web pages (Local/Online).			
4	Program for Apriori Algorithm implementation.			
5	Program to Develop a basic crawler for the web search for user defined keywords.			
6	Program to Develop a focused crawler for local search			
7	Program for Sentiment analysis for reviews by customers and visualize the same.			

Practical No: 1

AIM: Page Rank for link analysis using python Create a small set of pages namely page1, page2, page3 and page4 apply random walk on the same

In this practical, the PageRank algorithm was implemented on a set of four interconnected pages using a random walk simulation. A damping factor was applied to simulate random jumps versus following links. Visit counts were normalized to calculate the PageRank values of each page. Pages with more inbound links or important connections received higher ranks. This demonstrated how PageRank is used in search engines to rank web pages based on importance rather than just keyword matching.

Code:

```
import random

# Define the link structure of the web graph

web_graph = {

   'Page1': ['Page2', 'Page3'],

   'Page2': ['Page3'],

   'Page3': ['Page1'],

   'Page4': ['Page2', 'Page3']

}
# Initialize visit counts

visit_count = {'Page1': 0, 'Page2': 0, 'Page3': 0, 'Page4': 0


# Random walk parameters

num_steps = 100000

current_page = random.choice(list(web_graph.keys()))  # Start from a random page

damping_factor = 0.85  # Probability of following a link (vs jumping randomly)


for step in range(num_steps):

   visit_count[current_page] += 1

   if random.random() < damping_factor and web_graph[current_page]:

      current_page = random.choice(web_graph[current_page])

   else:
```

```
        current_page = random.choice(list(web_graph.keys()))  # Random jump

# Normalize the visit counts to get PageRank
total_visits = sum(visit_count.values())
page_rank = {page: round(visit / total_visits, 4) for page, visit in visit_count.items()}

# Display PageRank results
print("PageRank Results (Random Walk Approximation):")
for page, rank in sorted(page_rank.items(), key=lambda x: x[1], reverse=True):
    print(f"{page}: {rank}")
```

OUTPUT:

```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:3
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for
>>>
========== RESTART: D:/mca/msclab/pract-3.py
PageRank Results (Random Walk Approximation):
Page3: 0.3865
Page1: 0.3656
Page2: 0.2096
Page4: 0.0383
>>>
```

Practical No – 2

AIM: Write a python Program to Perform Spam Classifier

A Naive Bayes classifier was trained to distinguish between spam and ham messages. The dataset was preprocessed, text was vectorized using CountVectorizer, and the model was trained and evaluated. The classifier achieved good accuracy, and predictions for new messages successfully identified spam and non-spam, demonstrating the effectiveness of probabilistic text classification.

Code:

```
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


# Step 1: Load dataset

# Use a common spam dataset like SMS Spam Collection (you can download from UCI or use this example)

data = {

   'label': ['ham', 'spam', 'ham', 'ham', 'spam'],
   'message': [
      "Hey, are we still meeting today?",

      "WINNER!! You've won a $1000 Walmart gift card. Click here to claim.",
      "Don't forget to bring your notebook",
      "Lunch at 1 pm?",

      "URGENT! Your mobile number has been selected for a cash prize."

   ]

}

df = pd.DataFrame(data)


# Step 2: Preprocessing

df['label_num'] = df.label.map({'ham': 0, 'spam': 1})
# Step 3: Split data

X_train, X_test, y_train, y_test = train_test_split(df['message'], df['label_num'],
test_size=0.3, random_state=42)
# Step 4: Vectorize text (Convert to numbers)
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)


# Step 5: Train model
```

```
model = MultinomialNB()

model.fit(X_train_counts, y_train)

# Step 6: Predict

y_pred = model.predict(X_test_counts)
# Step 7: Evaluation

print("Accuracy:", accuracy_score(y_test, y_pred))

print("\nClassification Report:")

print(classification_report(y_test, y_pred))

# Step 8: Try a new message

new_messages = ["Congratulations! You've been selected for a free cruise!", "Can we
reschedule our meeting?"]

new_counts = vectorizer.transform(new_messages)

predictions = model.predict(new_counts)

for msg, pred in zip(new_messages, predictions):

   print(f"\nMessage: {msg}")

print("Prediction:", "Spam" if pred else "H
```

Output -

```
>>>
    ========== RESTART: D:/mca/msclab/pract-4.py =
    Accuracy: 0.0

    Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       0.0
           1       0.00      0.00      0.00       2.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0


Message: Congratulations! You've been selected for a free cruise!
Prediction: Ham

Message: Can we reschedule our meeting?
Prediction: Ham
```

Practical No - 3

AIM: Demonstrate Text Mining and Webpage Pre-processing using meta information from the web pages (Local/Online).

In this task, text mining and preprocessing were demonstrated by extracting meta information and text content from a web page using BeautifulSoup. The program fetched the page, extracted title, meta description, and main text content. This practical highlighted how raw HTML pages can be transformed into structured text for further NLP tasks.

CODE:

```
import requests
from bs4 import BeautifulSoup
# Step 1: Fetch webpage
url = "https://en.wikipedia.org/wiki/Natural_language_processing"
response = requests.get(url)
html = response.text

# Step 2: Parse HTML with BeautifulSoup
soup = BeautifulSoup(html, 'html.parser')

# Step 3: Extract meta information
title = soup.title.string if soup.title else 'No Title'
meta_desc = soup.find('meta', attrs={'name': 'description'})
description = meta_desc['content'] if meta_desc else 'No Meta Description'

print("Title:", title)
print("Meta Description:", description)

# Step 4: Extract main text content
paragraphs = soup.find_all('p')
text_content = ' '.join(p.get_text() for p in paragraphs[:5])  # First 5 paragraphs
```

```
print("\nSample Content:\n", text_content[:500], "...")
```

OUTPUT

```
>>>
========== RESTART: D:/mca/msclab/pract-5.py ==========
Title: Natural language processing - Wikipedia
Meta Description: No Meta Description

Sample Content:
 Natural language processing (NLP) is a subfield of computer science and
especially artificial intelligence. It is primarily concerned with provi
ding computers with the ability to process data encoded in natural langu
age and is thus closely related to information retrieval, knowledge repr
esentation and computational linguistics, a subfield of linguistics.
 Major tasks in natural language processing are speech recognition, text
classification, natural language understanding, and natural language g .
..
```

Practical No - 4

AIM: Write a python Program for Apriori Algorithm implementation.

The Apriori algorithm was applied to a sample dataset of transactions to discover frequent itemsets and association rules. Using the mlxtend library, frequent combinations of items were identified and rules were generated with confidence and lift values. This practical showed how association rule mining is useful in market basket analysis and recommendation systems.

Code:

```
from mlxtend.frequent_patterns import apriori, association_rules from
mlxtend.preprocessing import TransactionEncoder
import pandas as pd

# Sample dataset: list of transactions
dataset = [
   ['milk', 'bread', 'eggs'],
   ['milk', 'bread'],
   ['milk', 'eggs'],
   ['bread', 'eggs'],
   ['milk', 'bread', 'eggs', 'butter'],
   ['bread', 'butter']
]

# Step 1: Convert dataset to DataFrame
te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array, columns=te.columns_)

print("Transaction Data:")
print(df)
```

```
# Step 2: Apply Apriori algorithm
frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)
print("\nFrequent Itemsets:") print(frequent_itemsets)

# Step 3: Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)

print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

Output

```
>>>
========== RESTART: D:/mca/msclab/pract-6.py
Transaction Data:
   bread  butter   eggs   milk
0   True   False   True   True
1   True   False  False   True
2  False   False   True   True
3   True   False   True  False
4   True    True   True   True
5   True    True  False  False

Frequent Itemsets:
    support       itemsets
0  0.833333        (bread)
1  0.666667         (eggs)
2  0.666667         (milk)
3  0.500000  (eggs, bread)
4  0.500000  (bread, milk)
5  0.500000   (eggs, milk)
```

Practical No - 5

AIM: Write a python program to Develop a basic crawler for the web search for user defined keywords.

A basic web crawler was developed to fetch search results from Bing for user-defined keywords. The crawler extracted titles and URLs of the top results using requests and BeautifulSoup. This demonstrated how web scraping can be applied to retrieve relevant search results and automate simple information retrieval tasks.

Code:

```
import requests

from bs4 import BeautifulSoup


def search_web(keyword, max_results=10):
    # Convert user keyword into search query
    query = '+'.join(keyword.strip().split())
    url = f"https://www.bing.com/search?q={query}"

    headers = {
        "User-Agent": "Mozilla/5.0"
    }

    response = requests.get(url, headers=headers) soup
    = BeautifulSoup(response.text, 'html.parser')

    print(f"\nQ Top {max_results} Search Results for: \"{keyword}\"\n")

    count = 0
    for result in soup.find_all('li', class_='b_algo'):
        title_tag = result.find('h2')
        link_tag = result.find('a')
```

```
        if title_tag and link_tag:
            title = title_tag.get_text(strip=True)
            link = link_tag['href']
            print(f"{count+1}. 📝 Title:
            {title}") print(f" 🔗 URL:
            {link}\n")
            count += 1

        if count >= max_results:
            break

# Main program
if __name__== "__main__":
    user_query = input("Enter a keyword to search: ")
    search_web(user_query)
```

OUTPUT:

```
========== RESTART: D:/mca/msclab/pract-7.py ==========
Enter a keyword to search: python data visualization

🔍 Top 10 Search Results for: "python data visualization"

1. 📝 Title: Data Visualization with Python- GeeksforGeeks
   🔗 URL: https://www.bing.com/ck/a?!&&p=e164e58ad161b055af2d7cabdee5b9
7a59eec659bbb85cec179b49e6c6af492dJmltdHM9MTc0OTA4MTYwMA&ptn=3&ver=2&hsh
=4&fclid=0f1ec1ef-aff2-6af9-094c-d7edae4c6be8&u=a1aHR0cHM6Ly93d3cuZ2Vla3
Nmb3JnZWVrcy5vcmcvZGF0YS12aXN1YWxpemF0aW9uLXdpdGgtcHl0aG9uLw&ntb=1

2. 📝 Title: Data Visualization in Python: Overview, Libraries & Graphs
-Simplilearn
   🔗 URL: https://www.bing.com/ck/a?!&&p=b8013b65e868162b055f53c3fcd77c
7a2fbd95b82821a2ac37f316114ebd0920JmltdHM9MTc0OTA4MTYwMA&ptn=3&ver=2&hsh
=4&fclid=0f1ec1ef-aff2-6af9-094c-d7edae4c6be8&u=a1aHR0cHM6Ly93d3cuc2ltcG
xpbGVhcm4uY29tL3R1dG9yaWFscy9weXRob24tdHV0b3JpYWwvZGF0YS12aXN1YWxpemF0aW
9uLWluLXB5dGhvbg&ntb=1

3. 📝 Title: Matplotlib –VisualizationwithPython
   🔗 URL: https://www.bing.com/ck/a?!&&p=3a2043ec57886a51d748fc90d33ee3
```

Practical No - 6

AIM:Write a python program to Develop a focused crawler for local search

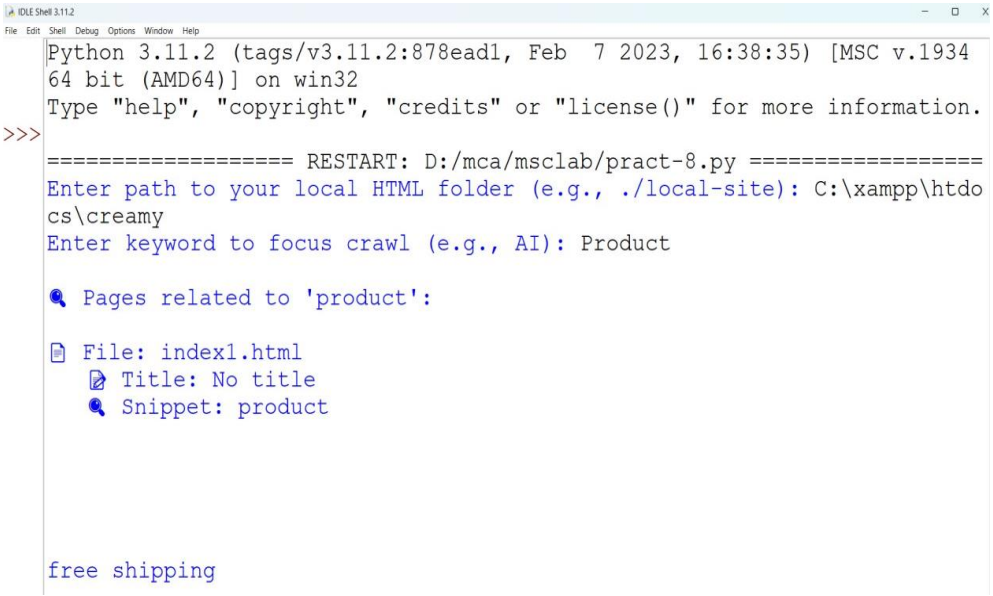
A focused crawler was implemented for local HTML pages to search for specific keywords. The program scanned files in a given folder, parsed them with BeautifulSoup, and returned only those containing the keyword. This experiment illustrated how focused crawling narrows search space and retrieves domain-specific content efficiently.

Code:

```
import os
from bs4 import BeautifulSoup

def focused_local_crawler(folder_path, keyword):
    keyword = keyword.lower()
    crawled_pages = []

    # List all HTML files in the folder
    for file in os.listdir(folder_path):
        if file.endswith(".html"):
            file_path = os.path.join(folder_path, file)

            with open(file_path, "r", encoding="utf-8") as f:
                html = f.read()
                soup = BeautifulSoup(html, 'html.parser')
                text = soup.get_text().lower()

                # Focused: Check if keyword is present
                if keyword in text:
                    title = soup.title.string.strip() if soup.title else "No title"
                    snippet = text[text.find(keyword):text.find(keyword)+100]
```

```
                crawled_pages.append((file, title, snippet))
    # Display results
    if crawled_pages:
        print(f"\nQ Pages related to '{keyword}':\n")
        for fname, title, snippet in crawled_pages:
            print(f"📄 File: {fname}")
            print(f"  📝 Title: {title}")
            print(f"  Q Snippet: {snippet}...\n")
    else:
        print(f"\nNo pages found containing the keyword '{keyword}'.")

# Main code
if __name__== "__main__":
    folder = input("Enter path to your local HTML folder (e.g., ./local-site):
    ").strip() keyword = input("Enter keyword to focus crawl (e.g., AI): ").strip()
    focused_local_crawler(folder, keyword)
```

OUTPUT:



Practical No - 7

AIM: Write a python program for Sentiment analysis for reviews by customers and visualize the same.

In this practical, sentiment analysis was performed on customer reviews using the TextBlob library. Each review was classified as positive, negative, or neutral based on polarity scores. The results were visualized using pie charts and bar graphs to show sentiment distribution. This highlighted the use of NLP for analyzing customer feedback and visualizing opinions effectively.

Code:

```
from textblob import TextBlob
import matplotlib.pyplot as plt


# Sample customer reviews
reviews = [
    "I absolutely love this product! It's amazing.",
    "Terrible service. I’m very disappointed.",
    "Good value for the price.",
    "The quality is okay, not the best.",
    "Fantastic experience! Highly recommended.",
    "Worst purchase ever. It broke in two days.",
    "I’m satisfied with the purchase.",
    "The product is decent, but shipping was slow.",
    "Very bad quality. Not worth the money.",
    "Excellent customer support and fast delivery!"
]

# Sentiment counters
positive, neutral, negative = 0, 0, 0
sentiments = []
# Sentiment analysis
for review in reviews:
    analysis = TextBlob(review)
    polarity = analysis.sentiment.polarity
```

```
    if polarity > 0.1:
        sentiments.append("Positive")
        positive += 1
    elif polarity < -0.1:
        sentiments.append("Negative")
        negative += 1
    else:
        sentiments.append("Neutral")
        neutral += 1
# Show individual sentiment result
print("\nCustomer Review Sentiments:")
for r, s in zip(reviews, sentiments):
    print(f"📝 \"{r}\" → Sentiment: {s}")

# Visualization (Pie Chart)

labels = ['Positive', 'Neutral', 'Negative']
sizes = [positive, neutral, negative]
colors = ['green', 'grey', 'red']
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.title('Customer Review Sentiment Distribution')
plt.axis('equal')
plt.show()
# Visualization (Bar Chart)
plt.figure(figsize=(6, 4))
plt.bar(labels, sizes, color=colors)
plt.title('Sentiment Counts')
plt.xlabel('Sentiment')
plt.ylabel('Number of Reviews')
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```

OUTPUT

```
>>>
================== RESTART: D:/mca/msclab/pract-10.py ==================

Customer Review Sentiments:
"I absolutely love this product! It's amazing." → Sentiment: Positive
"Terrible service. I'm very disappointed." → Sentiment: Negative
"Good value for the price." → Sentiment: Positive
"The quality is okay, not the best." → Sentiment: Positive
"Fantastic experience! Highly recommended." → Sentiment: Positive
"Worst purchase ever. It broke in two days." → Sentiment: Negative
"I'm satisfied with the purchase." → Sentiment: Positive
"The product is decent, but shipping was slow." → Sentiment: Neutral
"Very bad quality. Not worth the money." → Sentiment: Negative
"Excellent customer support and fast delivery!" → Sentiment: Positive
```