CL-II 5 IR

July 22, 2025

[39]:

```
'''NAME:Aher Swami Sandip
ROLL NO.01
COURSE: AI&DS
CLASS: BE
SUB:Computer Laboratory-II (Information Retrival)'''
```

[1]:

```
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin
import numpy as np

# Function to get all the links from a webpage
def get_links(url):
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.content, 'html.parser')
        links = set()

        for link in soup.find_all('a', href=True):
            absolute_url = urljoin(url, link['href'])
            if absolute_url.startswith('http'):
                links.add(absolute_url)
        return links
    except Exception as e:
        print(f"Error fetching {url}: {e}")
        return set()

# Function to build the link graph
def build_graph(start_url, depth=2):
    pages = {start_url}  # Initialize the set of pages with the start URL
    graph = {}

    # Crawl pages up to the given depth
    for _ in range(depth):
        new_pages = set()
        for page in pages:
            if page not in graph:  # Only process pages that haven't been␣
 ↪processed
```

```python
                links = get_links(page)  # Get links from the current page
                graph[page] = links      # Store the links in the graph
                new_pages.update(links)  # Add newly discovered links to␣
 ↪new_pages set
        pages.update(new_pages)  # Update pages to include newly found pages

    return graph

# Example: Starting from a single URL
start_url = "https://example.com"
depth = 2  # Define the depth here

link_graph = build_graph(start_url, depth)

# PageRank implementation
def page_rank(graph, iterations=100, d=0.85):
    pages = list(graph.keys())
    n = len(pages)

    # Initialize PageRank values
    ranks = np.ones(n) / n

    # Create adjacency matrix
    adjacency_matrix = np.zeros((n, n))

    for i, page in enumerate(pages):
        for link in graph[page]:
            if link in pages:
                j = pages.index(link)
                adjacency_matrix[j, i] = 1.0 / len(graph[page])

    # PageRank iterative process
    for _ in range(iterations):
        ranks = (1 - d) / n + d * adjacency_matrix.dot(ranks)

    # Mapping pages back to their PageRank values
    page_rank_dict = {pages[i]: ranks[i] for i in range(n)}
    return page_rank_dict

# Compute PageRank
ranks = page_rank(link_graph)
for page, rank in ranks.items():
    print(f"{page}: {rank:.4f}")
```

```
https://example.com: 0.0750
https://www.iana.org/domains/example: 0.1388
```

[ ]: