WEEK-7

7. To parse XML text, generate Web graph and compute topic specific page rank

Parse XML Text

Parsing XML means reading the XML data, understanding its structure, and extracting the relevant information from it.

- Purpose: Often, XML files store web crawl data — URLs, hyperlinks, anchor texts, and metadata.
- Process:
    1. Load XML file into memory.
    2. Use an XML parser (like Python's xml.etree.ElementTree or lxml) to navigate the tree structure.
    3. Extract required fields — e.g., <page>, <link>, <title>, <content>.
    4. Store these in data structures like dictionaries or adjacency lists for further graph processing.

Generate Web Graph

The web graph is a directed graph where:

- Nodes (vertices) = web pages
- Edges = hyperlinks from one page to another
- Construction:
    1. Create a mapping from URL → node ID.
    2. For each <page> in the XML, list its outgoing <link>s.
    3. Build an adjacency list or adjacency matrix.
- Purpose: This graph becomes the foundation for PageRank calculations.

Compute Topic-Specific PageRank

Topic-specific PageRank modifies the standard PageRank by biasing the random surfer towards a set of topic-related pages.

- Idea: Instead of teleporting uniformly to any page, the algorithm teleports more often to pages relevant to a given topic (e.g., "sports" or "technology").
- Steps:
    1. Identify a topic seed set — pages known to be about the topic.
    2. Initialize teleportation vector v where:
        - Higher values for topic-relevant pages

- Lower (or zero) for irrelevant pages

3. Use the PageRank equation:

$$PR=\alpha MPR+(1-\alpha)v$$

- M: transition matrix from the web graph
- α (alpha): damping factor (usually 0.85)
- v: topic-biased teleportation vector

4. Iterate until convergence (small changes between iterations).

PYTHON CODE

```
import xml.etree.ElementTree as ET
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
# Step 1: Parse XML
def parse_xml(xml_text):
    tree = ET.fromstring(xml_text)
    graph = {}
    topics_map = {}
    for page in tree.findall('page'):
        title = page.find('title').text.strip()
        links = [link.text.strip() for link in page.findall('link')]
        topics = page.find('topics').text.strip().split(",") if page.find('topics') is not None else []
        graph[title] = links
        topics_map[title] = [t.strip() for t in topics]
    return graph, topics_map
# Step 2: Build Adjacency Matrix
def build_adj_matrix(graph):
    pages = list(graph.keys())
    idx = {page: i for i, page in enumerate(pages)}
    n = len(pages)
    M = np.zeros((n, n))
```

```
    for page, links in graph.items():
        if links:
            for link in links:
                if link in idx:
                    M[idx[link]][idx[page]] = 1 / len(links)
        else:
            M[:, idx[page]] = 1 / n  # dangling node handling
    return M, pages
# Step 3: Compute Topic-Specific PageRank
def topic_specific_pagerank(M, pages, topics_map, topic, d=0.85, tol=1e-6, max_iter=100):
    n = len(pages)
    teleport = np.array([1.0 if topic in topics_map[p] else 0.0 for p in pages])
    if teleport.sum() == 0:
        teleport = np.ones(n)
    teleport = teleport / teleport.sum()  # normalize
    r = np.ones(n) / n  # initial rank
    for i in range(max_iter):
        r_new = d * M @ r + (1 - d) * teleport
        if np.linalg.norm(r_new - r, 1) < tol:
            break
        r = r_new
    return dict(zip(pages, r))
# Step 4: Visualize the Web Graph with Topic Highlight
def draw_web_graph(graph, topics_map, topic):
    G = nx.DiGraph()
    for page, links in graph.items():
        for link in links:
            G.add_edge(page, link)
    # Node colors: highlight pages having the topic
    node_colors = []
```

```
    for page in G.nodes():
        if topic in topics_map.get(page, []):
            node_colors.append("lightgreen")  # highlight topic pages
        else:
            node_colors.append("skyblue")     # normal pages
    plt.figure(figsize=(6, 4))
    pos = nx.spring_layout(G, seed=42)
    nx.draw(G, pos, with_labels=True, node_color=node_colors,
            node_size=1500, font_size=10, arrowsize=15, edge_color="gray")
    plt.title(f"Web Graph (Highlighted Topic: {topic})")
    plt.show()
```

 # Step 5: Input and Execute

```
xml_text = '''<web>
    <page>
        <title>PageA</title>
        <link>PageB</link>
        <link>PageC</link>
        <topics>science,education</topics>
    </page>
    <page>
        <title>PageB</title>
        <link>PageC</link>
        <topics>science</topics>
    </page>
    <page>
        <title>PageC</title>
        <topics>sports</topics>
    </page>
</web>'''
```

```
graph, topics_map = parse_xml(xml_text)
M, pages = build_adj_matrix(graph)
# Draw the web graph with topic highlighting
topic = "science"
draw_web_graph(graph, topics_map, topic)
# Compute topic-specific PageRank
ranks = topic_specific_pagerank(M, pages, topics_map, topic)
print("\nTopic-Specific PageRank (Topic: science):")
for page, score in sorted(ranks.items(), key=lambda x: -x[1]):
    print(f"{page}: {score:.4f}")
```