Department of Computer Science and Engineering (Data Science)

Machine Learning – IV

Experiment 5

Name : Jhanvi Parekh
Sap : 60009210033

Aim:

To implement and gain a comprehensive understanding of the PageRank algorithm, a fundamental algorithm for ranking web pages based on their importance.

Theory:

Introduction to PageRank Algorithm:

- Developed by Larry Page and Sergey Brin at Google, PageRank is a link analysis algorithm used to assign a numerical weight to each element of a hyperlinked set of web pages.
- The algorithm assumes that more important pages are likely to receive more links from other pages.

Algorithm Overview: •

Random Surfer Model:

  - o PageRank models a user who starts on a random page and follows links with a certain probability.

- Link Matrix and Transition Probability:
  - o Represent the web as a matrix where each element (i, j) corresponds to the link from page i to page j. o Normalize the matrix to obtain the transition probability matrix.
- PageRank Calculation:
  - o Iteratively compute the PageRank vector un l convergence using the formula:
    - PR(A)=(1−d)+d(L(B)PR(B)+L(C)PR(C)+…)
    - where PR(A) is the PageRank of page A, PR(B) is the PageRank of page B, L(B) is the number of outbound links on page B, and d is the damping factor (typically set to 0.85).

Step-by-Step Implementation:

- Step 1: Graph Representation: Represent the web pages and their links using a graph structure.
- Step 2: Transition Probability Matrix: Create a matrix representing the transition probabilities between pages.
- Step 3: Iterative PageRank Calculation: Implement the iterative algorithm to compute PageRank un l convergence.
- Step 4: Damping Factor: Integrate the damping factor into the algorithm.
- Step 5: Convergence Criteria: Define a convergence criterion to stop the iteration when PageRank values stabilize.

Implementation Tips:

- Use efficient data structures for the graph representation.
- Experiment with different damping factors and observe their impact on results.

Lab Experiments to be Performed in This Session:

Execute the PageRank algorithm on a dataset to gain insights into its functionality and operation.

```python
import numpy as np
```

```python
class PageRank:
    def __init__(self):
        self.__adj_list = {}
        self.__parent_list = {}
        self.__n = 0

    def addNode(self, node):
        self.__n += 1
        self.__adj_list[node] = self.__adj_list.get(node, [])
        self.__parent_list[node] = self.__parent_list.get(node, [])

    def addPath(self, u, v):
        self.__adj_list[u].append(v)
        self.__parent_list[v].append(u)

    def printAdjList(self):
        print("ADJACENCY LIST")
        for key, value in self.__adj_list.items():
            print(f"{key} -> {value}")
```

```python
    def __getM(self):
        M = [[0 for i in range(self.__n)] for j in range(self.__n)]
        for u in self.__adj_list:
            n = len(self.__adj_list[u])
            if n > 0:
                for v in self.__adj_list[u]:
                    M[v][u] = 1 / n
        return np.array(M)

    def printMatrix(self, matrix, name):
        print(f"\n{name} Matrix:")
        print(np.array(matrix))
```

```python
    def rankPages(self, iterations):
        R = [[1 / self.__n] for i in range(self.__n)]
        R = np.array(R)

        M = self.__getM()
        self.printMatrix(M, "Transition Matrix")

        for iteration in range(iterations):
            R = np.matmul(M, R)
            R = np.round(R, 3)
            print(f"\nRank Vector after iteration {iteration + 1}:")
            self.printMatrix(R, "Rank Vector")

        print(f"\nFINAL R VECTOR: ")
        print(*list(map(list, enumerate(R))), sep="\n")
        print(f"PAGE WITH MAXIMUM PAGERANK SCORE: {np.argmax(R)}")
```

```python
    def rankPagesWithBeta(self, iterations, beta):
        R = [1 / self.__n for _ in range(self.__n)]
        R = np.array(R)

        M = self.__getM()
        self.printMatrix(M, "Transition Matrix with Beta")

        for iteration in range(iterations):
            new_R = beta * np.matmul(M, R) + (1 - beta) / self.__n
            R = np.round(new_R, 3)
            print(f"\nRank Vector after iteration {iteration + 1} with Beta:")
            self.printMatrix(R, "Rank Vector with Beta")

        print(f"\nFINAL R VECTOR with Beta: ")
        print(*list(map(list, enumerate(R))), sep="\n")
        print(f"WIHT B = {beta}, PAGE WITH MAXIMUM PAGERANK SCORE: {np.argmax(R)}")
```

```python
PR = PageRank()
PR.addNode(0)
PR.addNode(1)
PR.addNode(2)

PR.addPath(0, 0)
PR.addPath(0, 1)
PR.addPath(0, 2)

PR.addPath(1, 0)
PR.addPath(1, 2)

PR.addPath(2, 1)
PR.addPath(2, 2)

PR.printAdjList()
print()
PR.rankPages(3)
```

```
ADJACENCY LIST
0 -> [0, 1, 2]
1 -> [0, 2]
2 -> [1, 2]


Transition Matrix Matrix:
[[0.33333333 0.5        0.        ]
 [0.33333333 0.         0.5       ]
 [0.33333333 0.5        0.5       ]]

Rank Vector after iteration 1:

Rank Vector Matrix:
[[0.278]
 [0.278]
 [0.444]]

Rank Vector after iteration 2:

Rank Vector Matrix:
[[0.232]
 [0.315]
 [0.454]]

Rank Vector after iteration 3:

Rank Vector Matrix:
[[0.235]
 [0.304]
 [0.462]]

FINAL R VECTOR:
[0, array([0.235])]
[1, array([0.304])]
[2, array([0.462])]
PAGE WITH MAXIMUM PAGERANK SCORE: 2
```

```python
PR.printAdjList()
print()
PR.rankPages(3)
print()
PR.rankPagesWithBeta(3, 0.85)
```

```
ADJACENCY LIST
0 -> [0, 1, 2]
1 -> [0, 2]
2 -> [1, 2]


Transition Matrix Matrix:
[[0.33333333 0.5        0.        ]
 [0.33333333 0.         0.5       ]
 [0.33333333 0.5        0.5       ]]

Rank Vector after iteration 1:

Rank Vector Matrix:
[[0.278]
 [0.278]
 [0.444]]

Rank Vector after iteration 2:

Rank Vector Matrix:
[[0.232]
 [0.315]
 [0.454]]

Rank Vector after iteration 3:

Rank Vector Matrix:
[[0.235]
 [0.304]
 [0.462]]

FINAL R VECTOR:
[0, array([0.235])]
[1, array([0.304])]
[2, array([0.462])]
PAGE WITH MAXIMUM PAGERANK SCORE: 2


Transition Matrix with Beta Matrix:
[[0.33333333 0.5        0.        ]
 [0.33333333 0.         0.5       ]
 [0.33333333 0.5        0.5       ]]
```

```
Rank Vector after iteration 1 with Beta:

Rank Vector with Beta Matrix:
[0.286 0.286 0.428]

Rank Vector after iteration 2 with Beta:

Rank Vector with Beta Matrix:
[0.253 0.313 0.434]

Rank Vector after iteration 3 with Beta:

Rank Vector with Beta Matrix:
[0.255 0.306 0.439]
```