PageRank , Personalized PageRank

Huy Init

Ngày 6 tháng 11 năm 2024

Tóm tắt nội dung

Bài viết này trình bày về thuật toán PageRank, một phương pháp xếp hạng quan trọng được sử dụng để đánh giá tầm quan trọng của các node trong đồ thị. Chúng tôi so sánh kết quả của thuật toán PageRank triển khai tùy chỉnh với mô hình PageRank có sẵn trong thư viện scikit-network. Kết quả cho thấy sự tương đồng đáng kể giữa hai phương pháp, mặc dù có sự khác biệt đối với node không có liên kết ra (sink node). Phân tích cho thấy cách tiếp cận khác biệt của mỗi mô hình trong việc xử lý các node loại này.

Keywords: PageRank, Đồ thị, Xếp hạng node, scikit-network, Node sink

1 Kiến thức cơ bản PageRank

[1]

1.1 Giới thiệu

PageRank là một thuật toán xếp hạng trang web, được Larry Page và Sergey Brin phát triển tại Đại học Stanford, với mục đích đo lường mức độ quan trọng của các trang web dựa trên cấu trúc liên kết giữa chúng. Dưới đây là các nội dung cơ bản về sự hình thành và ý nghĩa của thuật toán PageRank:

Sự hình thành của PageRank:

1. **Nền tảng lý thuyết**: Thuật toán PageRank dựa trên ý tưởng rằng các liên kết (links) từ một trang web đến một trang khác có thể được coi là một "phiếu bầu"cho tầm quan trọng của trang nhận được liên kết đó. Tuy nhiên, không phải tất cả các phiếu bầu đều có giá trị như nhau. Một liên kết từ một trang web có thứ hạng cao sẽ có giá trị hơn so với liên kết từ một trang có thứ hạng thấp.

2. **Quy trình lặp lại**: PageRank hoạt động theo quy trình lặp đi lặp lại. Ban đầu, mỗi trang web được gán một giá trị rank (thứ hạng) khởi đầu, sau đó thứ hạng của mỗi trang được cập nhật dựa trên thứ hạng của các trang liên kết đến nó. Thứ hạng của một trang sẽ tỷ lệ thuận với tổng thứ hạng của các trang liên kết đến nó, chia cho số lượng liên kết mà các trang đó có.

3. **Mô hình "random surfer"**: PageRank mô phỏng một người dùng ngẫu nhiên di chuyển trên web bằng cách nhấp vào các liên kết. Các trang có nhiều liên kết đến sẽ có nhiều khả năng được người dùng ngẫu nhiên truy cập, do đó có thứ hạng cao hơn.

4. **Hệ số giảm dần (Damping factor)**: Thuật toán có thêm một hệ số giảm dần để giả định rằng người dùng có thể nhảy đến bất kỳ trang web nào một cách ngẫu nhiên, thay vì chỉ di chuyển theo các liên kết. Điều này giúp đảm bảo rằng ngay cả những trang ít liên kết trực tiếp vẫn có thể nhận được một lượng nhỏ thứ hạng.

Ý nghĩa của PageRank:

- **Xếp hạng website**: Thuật toán PageRank cung cấp một phương pháp để xếp hạng các trang web dựa trên mức độ quan trọng và sự liên quan của chúng trong mạng lưới liên kết trên internet. Điều này giúp công cụ tìm kiếm, như Google, quyết định trang nào nên xuất hiện ở đầu kết quả tìm kiếm.

- **Biểu thị uy tín**: PageRank không chỉ xem xét số lượng liên kết đến một trang mà còn chú trọng đến chất lượng của các liên kết. Những liên kết từ các trang web có uy tín sẽ có trọng lượng cao hơn, làm tăng thứ hạng cho các trang web nhận được liên kết.

Tóm lại, **PageRank** mang lại ý nghĩa quan trọng trong việc sắp xếp và tạo thứ tự cho các trang web trên internet, giúp người dùng dễ dàng tìm thấy những trang thông tin có giá trị nhất dựa trên các liên kết giữa chúng.

1.2 Một số ma trận được dùng trong thuật toán

Ma trận PageRank $P_r(t)$:

$$P_r(t) = \begin{bmatrix} P_r(p_0;t) \\ P_r(p_1;t) \\ \vdots \\ P_r(p_{N_p};t) \end{bmatrix}$$

Giải thích: Đây là ma trận cột chứa các giá trị PageRank của tất cả các nút tại bước tính toán t.

Ma trận Outdegree D:

$$D = \begin{bmatrix} \frac{1}{D_{0,0}} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{D_{1,1}} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{D_{N_p,N_p}} \end{bmatrix}$$

Giải thích: Đây là ma trận đường chéo với các giá trị nghịch đảo của số lượng liên kết ra từ mỗi nút. Nếu nút p_i có $D_{i,i}$ liên kết ra, giá trị tại vị trí (i,i) là $\frac{1}{D_{i,i}}$.

Ma trận Kề O:

$$O = \begin{bmatrix} O_{0,0} & O_{0,1} & \cdots & O_{0,N_p} \\ O_{1,0} & O_{1,1} & \cdots & O_{1,N_p} \\ \vdots & \vdots & \ddots & \vdots \\ O_{N_p,0} & O_{N_p,1} & \cdots & O_{N_p,N_p} \end{bmatrix}$$

Giải thích: Đây là ma trận kề, trong đó mỗi phần tử $O_{j,i}$ đại diện cho số lượng liên kết từ nút p_j đến nút p_i.

Ma trận Kề Điều Chỉnh $\hat{O}$:

$$\mathbf{\hat{O} = O \cdot D}$$

Giải thích: Đây là ma trận kề đã được chuẩn hóa theo số lượng liên kết ra từ mỗi nút, tạo ra bằng cách nhân ma trận kề O với ma trận đường chéo D.

Việc sử dụng ba ma trận trong quá trình tính toán PageRank hoặc các thuật toán liên quan đến đồ thị là cần thiết vì mỗi ma trận đại diện cho một khía cạnh khác nhau của đồ thị. Cụ thể:

Ma trận kề (Adjacency Matrix):

Chức năng: Biểu diễn mối quan hệ giữa các nút (nodes) trong đồ thị. Ma trận này xác định nút nào có liên kết với nút nào.

Ý nghĩa: Mỗi hàng và cột tương ứng với một nút, và giá trị tại vị trí [i, j] là số lượng cạnh từ nút j đến nút i. Điều này giúp nắm bắt cấu trúc của đồ thị và cách các nút kết nối với nhau.

Ma trận bậc ra (Outdegree Matrix):

Chức năng: Là một ma trận chéo, trong đó giá trị tại mỗi nút là nghịch đảo số lượng cạnh ra (outdegree) từ nút đó.

Ý nghĩa: Khi tính toán PageRank, việc cân nhắc bậc ra rất quan trọng vì nó cho thấy mức độ ảnh hưởng của một nút dựa trên số liên kết mà nó tạo ra. Nếu một nút có nhiều cạnh ra, điểm PageRank của nó sẽ được chia đều cho các nút khác, và nghịch đảo bậc ra giúp chuẩn hóa điều này.

Ma trận kề đã điều chỉnh (Modified Adjacency Matrix):

Chức năng: Được tạo ra bằng cách nhân ma trận kề với ma trận bậc ra, ma trận này điều chỉnh sự phân phối ảnh hưởng giữa các nút.

Ý nghĩa: Ma trận này giúp phân phối ảnh hưởng của các nút trong đồ thị theo cách mà một nút có nhiều cạnh ra sẽ phân phối điểm số của nó cho các nút liên quan theo tỷ lệ thích hợp. Đây là yếu tố quan trọng trong thuật toán PageRank để xác định tầm quan trọng của các nút.

Tại sao cần cả ba ma trận?

Ma trận kề giúp xác định các mối quan hệ giữa các nút.

Ma trận bậc ra điều chỉnh sự phân phối điểm PageRank dựa trên số lượng liên kết đi từ một nút.

Ma trận kề đã điều chỉnh kết hợp cả hai thông tin trên để tính toán chính xác ảnh hưởng của các nút đến nhau, đặc biệt là trong các thuật toán đánh giá độ quan trọng như PageRank.

Ba ma trận này kết hợp với nhau để đảm bảo tính chính xác và khả năng phản ánh mức độ ảnh hưởng của mỗi nút trong đồ thị.

1.3 Các bước Thuật toán

Thuật toán PageRank là một quy trình lặp đi lặp lại được sử dụng để xác định mức độ quan trọng của từng trang trong một đồ thị. Dưới đây là mô tả chi tiết:

- **Bước 1:Khởi Tạo (tại $t = 0$)** Đặt giá trị xếp hạng ban đầu cho tất cả các trang (nút) trong đồ thị:

$$P_r(p_i; 0) = \frac{1}{N_p}$$

  Tại thời điểm bắt đầu (vòng lặp $t = 0$), mỗi trang được gán một giá trị xếp hạng ban đầu bằng nhau, là tỷ lệ nghịch của tổng số trang N_p. Điều này có nghĩa là mỗi trang bắt đầu với một xếp hạng bằng nhau.

- **Bước 2: Lặp Lại (cho $t = 1, 2, 3, \ldots$)** Cập nhật giá trị xếp hạng cho mỗi trang theo công thức sau:

$$\mathbf{P_r(t+1) = d \cdot \hat{O} \cdot P_r(t) + \frac{1-d}{N_p} \cdot 1}$$

  Giải thích:

- d: Hệ số giảm tốc (damping factor), thường có giá trị khoảng 0.85. Nó kiểm soát mức độ mà thuật toán dựa vào các liên kết giữa các nút so với việc nhảy ngẫu nhiên.
- $\hat{O}$: Ma trận kề đã chuẩn hóa.
- $P_r(t)$: Vector PageRank tại bước t.
- $\frac{1-d}{N_p}$: Một phần của giá trị được cộng thêm vào để đại diện cho khả năng nhảy ngẫu nhiên đến bất kỳ trang nào.
- 1: Vector cột với tất cả phần tử bằng 1, đại diện cho sự đồng đều của xác suất nhảy ngẫu nhiên.

Lặp lại Bước 2 cho đến khi sự thay đổi giữa hai bước lặp $P_r(t+1)$ và $P_r(t)$ nhỏ hơn một giá trị ϵ (độ sai số chấp nhận), nghĩa là:

$$|P_r(t+1) - P_r(t)| < \epsilon$$

Khi điều kiện này thỏa mãn, thuật toán dừng lại và kết quả PageRank được coi là hội tụ.

1.4 Giải thích cách tính PageRank bằng phương pháp Power Iteration (lặp công suất)

Công thức cập nhật giá trị PageRank dựa trên lý thuyết về ngẫu nhiên và đồ thị, đặc biệt là mô hình "random surfer" (người lướt web ngẫu nhiên). Dưới đây là các cơ sở để xây dựng công thức:

1. Ý tưởng "Random Surfer":

Giả sử một người dùng (người lướt web ngẫu nhiên) đang truy cập một trang web bất kỳ. Người này có thể:

- Chọn một liên kết trên trang hiện tại và truy cập một trang mới theo liên kết đó (theo ma trận kề $\hat{O}$).
- Nhảy ngẫu nhiên đến bất kỳ trang nào trên web mà không theo liên kết nào (được điều khiển bởi hệ số giảm tốc $1 - d$).

Mô hình này giúp mô phỏng hành vi thực tế của người dùng khi duyệt web.

2. Tính chất của PageRank:

PageRank của một trang được xác định bởi số lượng và chất lượng của các liên kết đến nó. Chất lượng của liên kết được đánh giá dựa trên PageRank của trang gốc, cùng với số lượng liên kết ra từ trang gốc đó.

Nếu một trang nhận được liên kết từ nhiều trang khác có PageRank cao, giá trị PageRank của nó sẽ tăng lên. Tương tự, nếu trang gốc có nhiều liên kết ra (out-links), PageRank cho mỗi liên kết ra sẽ bị chia nhỏ.

Thuật toán PageRank sử dụng hệ số giảm tốc d để kiểm soát sự cân bằng giữa hai khả năng:

- Lướt web qua liên kết: Người dùng sẽ theo liên kết trên trang web với xác suất d, và điều này được mô phỏng bởi ma trận kề chuẩn hóa $\hat{O}$.
- Nhảy ngẫu nhiên: Người dùng có thể nhảy đến một trang ngẫu nhiên với xác suất $1 - d$. Điều này được thêm vào để đảm bảo tính hội tụ của thuật toán ngay cả trong các trường hợp đồ thị có các thành phần cô lập (như không có liên kết ra từ một trang cụ thể).

Nội dung này giải thích cách tính PageRank bằng phương pháp Power Iteration (lặp công suất), một kỹ thuật từ đại số tuyến tính.

Tóm tắt ý chính:

1. $P_r(t)$: Đây là một vector cột chứa tất cả giá trị PageRank của các trang (node) tại bước lặp thứ t. Mỗi phần tử của vector đại diện cho giá trị PageRank của một trang cụ thể tại thời điểm đó.

2. D: Là ma trận chéo vuông, trong đó mỗi phần tử trên đường chéo chính là nghịch đảo của số cạnh ra (out-degree) của mỗi node. Nếu một node không có cạnh ra (outgoing edges), phần tử tương ứng sẽ là 0.

3. O: Là ma trận kề (adjacency matrix) của đồ thị, thể hiện các liên kết giữa các trang (node) trong đồ thị.

4. $\hat{O}$: Đây là phiên bản điều chỉnh của ma trận kề O, trong đó mỗi phần tử $\hat{O}_{i,j}$ thể hiện tỷ lệ số cạnh từ trang j đến trang i chia cho tổng số cạnh ra từ trang j. Nói cách khác, cột tương ứng với node j trong ma trận $\hat{O}$ được chuẩn hóa sao cho tổng của cột đó bằng 1.

Phương trình cập nhật PageRank: Phương trình cập nhật giá trị PageRank tại bước $t+1$ được diễn giải như sau:

$$\mathbf{P_r(t+1) = d \cdot \hat{O} \cdot P_r(t) + \frac{1-d}{N_p} \cdot 1}$$

- d: hệ số giảm tốc (damping factor), thường có giá trị khoảng 0.85. Nó kiểm soát mức độ mà thuật toán dựa vào các liên kết giữa các node so với việc nhảy ngẫu nhiên.

- $\hat{O}$: ma trận kề đã chuẩn hóa.

- $P_r(t)$: vector PageRank tại bước t.

- $\frac{1-d}{N_p}$: một phần của giá trị được cộng thêm vào để đại diện cho khả năng nhảy ngẫu nhiên đến bất kỳ trang nào.

- 1: vector cột với tất cả phần tử bằng 1, đại diện cho sự đồng đều của xác suất nhảy ngẫu nhiên.

Chú ý :

- $d \cdot \hat{O} \cdot P_r(t)$: Đây là phần mô phỏng việc người dùng nhấp vào các liên kết theo ma trận kề chuẩn hóa $\hat{O}$. Nó cho thấy PageRank của trang ở bước tiếp theo phụ thuộc vào tổng PageRank từ các trang liên kết đến.

- $\frac{1-d}{N_p} \cdot 1$: Đây là phần mô phỏng khả năng nhảy ngẫu nhiên đến bất kỳ trang nào trên đồ thị. Mỗi trang có xác suất được nhảy đến bằng nhau, vì vậy phần tử này phân bố đều giá trị PageRank cho tất cả các trang.

Điều kiện hội tụ: Thuật toán tiếp tục lặp qua các bước cho đến khi sự thay đổi giữa hai bước lặp $P_r(t+1)$ và $P_r(t)$ nhỏ hơn một giá trị ϵ (độ sai số chấp nhận), nghĩa là:

$$\mathbf{|P_r(t+1) - P_r(t)| < \epsilon}$$

Khi điều kiện này thỏa mãn, thuật toán dừng lại và kết quả PageRank được coi là hội tụ.

Kết luận: Thuật toán sử dụng phương pháp Power Iteration để tính PageRank, trong đó các giá trị PageRank được lặp đi lặp lại cho đến khi chúng hội tụ, tức là không còn thay đổi nhiều qua các bước lặp.

1.5 Code

```
1 class Graph(object):
2
```

```
    def __init__(self, edges: List[Tuple]) -> None:
        # initialize objects
        nodes = set()
        indegrees = {}
        outdegrees = {}
        # determine the unique set of nodes in the graph, and count number
            ↪ of outbound edges per source node
        for edge in edges:
            nodes.update(list(edge))
            src, dst = edge
            try:
                outdegrees[src] += 1
            except:
                outdegrees[src] = 1
            try:
                indegrees[dst] += 1
            except:
                indegrees[dst] = 1
        nodes = list(nodes)
        nodes.sort()
        # store graph data
        self.edges = edges
        self.nodes = nodes
        self.number_nodes = len(nodes)
        self.number_edges = len(edges)
        self.indegrees = indegrees
        self.outdegrees = outdegrees

    def _build_adjacency_matrix(self) -> np.array:
        # work out adjacency matrix
        O = np.zeros((self.number_nodes,self.number_nodes))
        for edge in self.edges:
            src, dst = edge
            O[self.nodes.index(dst),self.nodes.index(src)] += 1
        return O

    def _build_outdegree_matrix(self) -> np.array:
        D = np.zeros((self.number_nodes,self.number_nodes))
        for node in self.nodes:
            try:
                D[self.nodes.index(node),self.nodes.index(node)] =
                    ↪ 1/self.outdegrees[node]
            except:
                D[self.nodes.index(node),self.nodes.index(node)] = 0
        return D

    def get_modified_adjacency_matrix(self) -> np.array:
        return
            ↪ np.matmul(self._build_adjacency_matrix(),self._build_outdegree_matrix
```

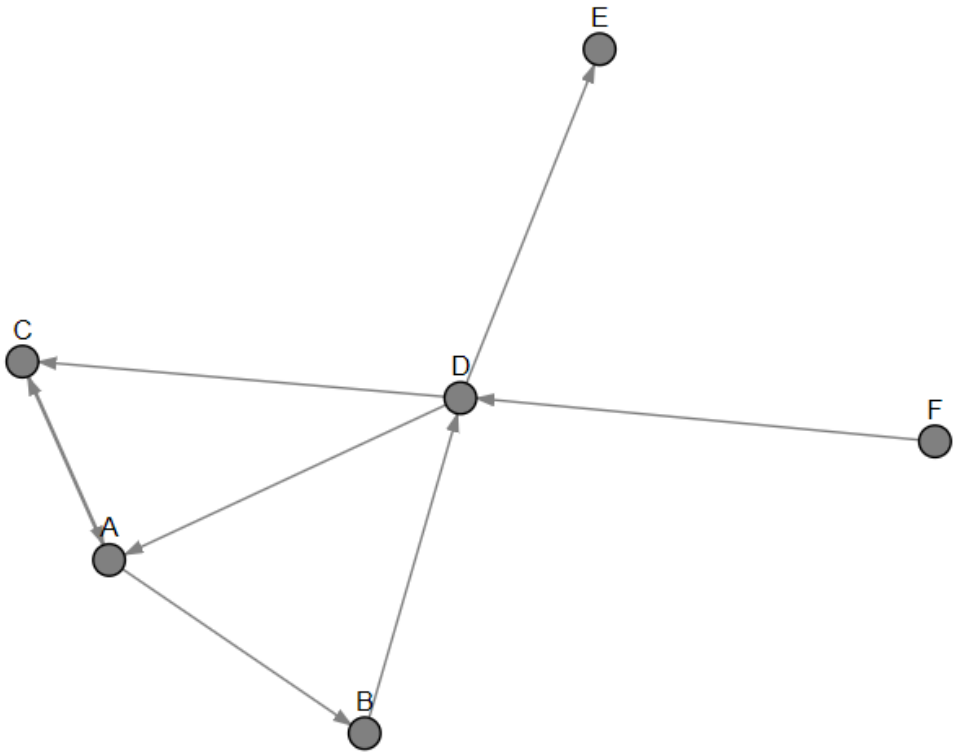
```
49
50      def get_edges(self) -> List[Tuple]:
51          return self.edges
52
53      def get_nodes(self) -> List:
54          return self.nodes
55
56      def get_number_edges(self) -> int:
57          return self.number_edges
58
59      def get_number_nodes(self) -> int:
60          return self.number_nodes
61
62      def get_indegrees(self) -> dict:
63          return self.indegrees
64
65      def get_outdegrees(self) -> dict:
66          return self.outdegrees
```

Listing 1: Class Graph in Python

```
1  class PageRank(object):
2
3      def __init__(self, damping_factor: float=0.85, epsilon: float=1e-8) ->
           ↪ None:
4          self.damping_factor = damping_factor
5          self.epsilon = epsilon
6
7      def _initalize_pagerank(self, graph: Graph) -> np.array:
8          return
               ↪ (1/graph.get_number_nodes())*np.ones((graph.get_number_nodes(),1))
9
10     def _identity_vector(self, graph: Graph) -> np.array:
11         return np.ones((graph.get_number_nodes(),1))
12
13     def _step(self, P1: np.array, I: np.array, graph: Graph) -> np.array:
14         P2 = (
15             self.damping_factor*np.matmul(graph.get_modified_adjacency_matrix(),P1)
16             + (1 - self.damping_factor)*I/graph.get_number_nodes()
17         )
18         return(P2/np.sum(P2))
19
20     def evaluate(self, graph: Graph) -> dict:
21         # obtain nodes from graph
22         nodes = graph.get_nodes()
23         # setup initial pagerank steps
24         P1 = self._initalize_pagerank(graph)
25         I  = self._identity_vector(graph)
26         P2 = self._step(P1, I, graph)
```

```
        # step through the algorithm, updating our pageranks
        while(np.linalg.norm(P1 - P2) >= self.epsilon):
            P1 = P2
            P2 = self._step(P1, I, graph)
        # package results and return
        pageranks = {}
        for node, rank in zip(nodes,P2.flatten()):
            pageranks[node] = rank
        return pageranks
```

Listing 2: Class PageRank in Python



Hình 1: Đồ thị ví dụ với các node và cạnh.

Trong đồ thị được xét, danh sách các cạnh bao gồm: `[("A", "B"), ("B", "D"), ("D", "A"), ("D", "C"), ("A", "C"), ("C", "A"), ("D", "E"), ("F", "D")]`.
Danh sách các node sau khi sắp xếp là: `['A', 'B', 'C', 'D', 'E', 'F']`.
Danh sách các node bao gồm: `['A', 'B', 'C', 'D', 'E', 'F']`. Số lượng node trong đồ thị là 6 và số lượng cạnh là 8.
Số lượng cạnh vào của các node (indegrees) được xác định như sau: `'B': 1, 'D': 2, 'A': 2, 'C': 2, 'E': 1`.
Số lượng cạnh ra của các node (outdegrees) được xác định như sau: `'A': 2, 'B': 1, 'D': 3, 'C': 1, 'F': 1`.

*Ma trận kề (Adjacency Matrix)

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

*Ma trận bậc ra (Outdegree Matrix)

$$\begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.33333333 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

*Ma trận kề đã điều chỉnh (Modified Adjacency Matrix)

$$\begin{bmatrix} 0 & 0 & 1 & 0.33333333 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0.33333333 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.33333333 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

*Khởi tạo PageRank

$$\begin{bmatrix} 0.16666667 \\ 0.16666667 \\ 0.16666667 \\ 0.16666667 \\ 0.16666667 \\ 0.16666667 \end{bmatrix}$$

*Khởi tạo Vector Đơn Vị

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Dưới đây là kết quả của các lần lặp trong thuật toán PageRank:shape 6x1

$$\text{Iteration 1:} \quad P2 = \begin{bmatrix} 0.28912513 \\ 0.14098989 \\ 0.25060997 \\ 0.15580342 \\ 0.13654583 \\ 0.02692576 \end{bmatrix}$$

$$\text{Iteration 2:} \quad P2 = \begin{bmatrix} 0.31921175 \\ 0.16729511 \\ 0.21723572 \\ 0.18975163 \\ 0.07822319 \\ 0.02828259 \end{bmatrix}$$

$$\text{Iteration 3:} \quad P2 = \begin{bmatrix} 0.28217506 \\ 0.17210844 \\ 0.22970069 \\ 0.20486228 \\ 0.08437289 \\ 0.02678064 \end{bmatrix}$$

$$\text{Iteration 45:} \quad P2 = \begin{bmatrix} 0.29526337 \\ 0.16277503 \\ 0.22454694 \\ 0.20155998 \\ 0.08881329 \\ 0.02704139 \end{bmatrix}$$

Kết quả của thuật toán PageRank là:

$$\text{ranks} = \begin{Bmatrix} A & : 0.29526336887933935 \\ B & : 0.16277503210453523 \\ C & : 0.22454693557427846 \\ D & : 0.20155998078146667 \\ E & : 0.08881329306506174 \\ F & : 0.027041389595318478 \end{Bmatrix}$$

1.6 Kiểm nghiệm bằng thư viện

Đoạn mã Python dưới đây sử dụng thư viện `sknetwork` để tính toán PageRank:

```
from sknetwork.ranking import PageRank
from sknetwork.data import from_edge_list
```

```
edges = [("A", "B"), ("B", "D"), ("D", "A"), ("D", "C"), ("A", "C"), ("C",
    ↪ "A"), ("D", "E"), ("F", "D")]
graph = from_edge_list(edges, directed=True)
adjacency = graph.adjacency
pagerank = PageRank()
scores = pagerank.fit_predict(adjacency)
print(scores)
```

Kết quả của PageRank là:

```
[0.24685275 0.1374625  0.18928852 0.17249258 0.22754571 0.02635795]
```

*Nhận xét về Kết quả PageRank

Sự tương đồng giữa các phương pháp Các xếp hạng của các node trong đồ thị rất giống nhau giữa cài đặt tùy chỉnh và mô hình `scikit-network`. Cụ thể, các node A, C, và D đều đứng đầu bảng xếp hạng. Điều này là hợp lý vì chúng có số lượng liên kết lớn (degree cao).

Sự khác biệt đối với Node E Sự khác biệt lớn nhất giữa hai bộ kết quả là giá trị PageRank của Node E. Trong mô hình `scikit-network`, Node E nhận được trọng số cao hơn. Điều này cho thấy mô hình `scikit-network` xử lý các node "sink"(hay còn gọi là node bị treo, không có liên kết ra) một cách khác biệt so với cài đặt tùy chỉnh.

Vấn đề với Node E Node E là một node "sink", nghĩa là nó không có liên kết ra. Thuật toán PageRank cổ điển gặp khó khăn trong việc xử lý chính xác các node loại này. Sự khác biệt giữa các kết quả là minh chứng rõ ràng cho vấn đề này. Để cải thiện thuật toán PageRank, cần có những điều chỉnh để xử lý các node như Node E, đảm bảo rằng tất cả các node, kể cả những node không có liên kết ra, đều được xử lý hợp lý trong tính toán PageRank.

2 Kiến thức cơ bản Personalized PageRank

Nội dung bạn đưa ra mô tả cách cập nhật giá trị PageRank và một khái niệm quan trọng trong thuật toán PageRank cá nhân hóa (Personalized PageRank). Dưới đây là giải thích chi tiết:

2.1 Quy tắc Cập Nhật PageRank

Quy tắc cập nhật PageRank được mô tả bằng công thức sau:

$$\mathbf{P_r(t+1) = d \cdot O \cdot P_r(t) + (1-d) \cdot \frac{1}{N_p} \cdot 1}$$

Trong đó:

- 1: Là vector cột gồm N_p phần tử, mỗi phần tử đều bằng 1. Đây là vector xác suất đồng đều cho các node trong mô hình PageRank không cá nhân hóa.
- $1 - d$: Được gọi là "fly-out probability"(xác suất bay ra). Đây là xác suất mà chúng ta sẽ chuyển đến một node nào đó không phụ thuộc vào liên kết trực tiếp từ node hiện tại.
  - Trong thuật toán PageRank không cá nhân hóa, vector 1 chứa toàn bộ các giá trị 1, nghĩa là mỗi node có cơ hội chuyển đến tất cả các node khác với xác suất đồng đều.

– Trong PageRank cá nhân hóa, chúng ta điều chỉnh xác suất bay ra để chỉ tập trung vào một hoặc một số node cụ thể. Do đó, chỉ những phần tử tương ứng với các node mà chúng ta quan tâm mới được gán giá trị 1.0, còn lại là 0.0.

2.2 PageRank Cá Nhân Hóa

Trong PageRank cá nhân hóa, chúng ta điều chỉnh thuật toán để tập trung vào các node cụ thể, thay vì phân phối xác suất đồng đều cho tất cả các node. Điều này có thể được thực hiện bằng cách:

- Chỉnh sửa vector 1 sao cho nó chỉ chứa giá trị 1.0 cho các node mà chúng ta quan tâm, và các phần tử khác là 0.0.

Khi xây dựng PageRank cá nhân hóa, bạn sẽ cần một lớp mới để xử lý thuật toán này. Lớp này sẽ:

- Thay đổi vector xác suất bay ra để chỉ tập trung vào các node mục tiêu.
- Cập nhật giá trị PageRank theo công thức đã điều chỉnh.

Bằng cách này, PageRank cá nhân hóa cho phép bạn điều chỉnh thuật toán để phản ánh tầm quan trọng hoặc sự chú ý đặc biệt đối với một số node cụ thể trong đồ thị.

2.3 Code

```
class PersonalizedPageRank(PageRank):

    def __init__(self, selected_nodes: List=[], damping_factor: float=0.85,
       ↪ epsilon: float=1e-8) -> None:
        super().__init__(damping_factor, epsilon)
        self.selected_nodes = selected_nodes

    def _identity_vector(self, graph: Graph) -> np.array:
        I = np.zeros((graph.get_number_nodes(),1))
        idx_selected = [graph.get_nodes().index(n) for n in
           ↪ graph.get_nodes() if n in self.selected_nodes]
        I[idx_selected] = 1.0
        print(I)
        return I
```

Listing 3: Personalized PageRank Class

```
# create a pagerank object
pr = PersonalizedPageRank(selected_nodes=["D"])
# compute pageranks
ranks = pr.evaluate(graph)
```

Listing 4: Personalized PageRank Calculation

Kết quả của quá trình tính toán Personalized PageRank với node "D"là node được chọn như sau:

$$\text{identity\_vector} = \left[[0.], [0.], [0.], [1.], [0.], [0.]\right]$$

$$\text{ranks} = \begin{Bmatrix} A & : 0.3195 \\ B & : 0.1667 \\ C & : 0.2379 \\ D & : 0.2047 \\ E & : 0.0712 \\ F & : 0.0000 \end{Bmatrix}$$

2.4 Kiểm nghiệm bằng thư viện

```
from sknetwork.data import karate_club, painters, movie_actor
from sknetwork.ranking import PageRank
from sknetwork.visualization import visualize_graph, visualize_bigraph

weights = {3: 1}
pagerank = PageRank()
scores = pagerank.fit_predict(adjacency, weights)
scores
```

Input: Graph

graph='names': array(['A', 'B', 'C', 'D', 'E', 'F'], dtype='<U1'), 'adjacency': <6x6 sparse matrix of type '<class 'numpy.int64'>' with 8 stored elements in Compressed Sparse Row format>

Output: PageRank scores

$$\text{array}([0.28638012, 0.12978779, 0.21526388, 0.28309211, 0.08547609, 0])$$

2.5 PageRank,Personalized PageRank, và Topic-Specific PageRank

PageRank, **Personalized PageRank**, và **Topic-Specific PageRank** là ba phiên bản của thuật toán PageRank được điều chỉnh để phục vụ các mục đích khác nhau. Dưới đây là phân biệt giữa chúng:

1. PageRank

PageRank là thuật toán gốc được phát triển bởi Larry Page và Sergey Brin để xếp hạng các trang web trên internet. Nó đánh giá tầm quan trọng của các trang dựa trên số lượng và chất lượng của liên kết đến trang đó. Các đặc điểm chính của PageRank là:

- **Công thức**:

$$P_r(t+1) = d \cdot O_r^P(t) + (1-d) \cdot \frac{1}{N_p} \cdot \mathbf{1}$$

Trong đó:

- $P_r(t)$ là vector PageRank tại bước t.
- O là ma trận kề đã được điều chỉnh.
- d là hệ số làm mờ (thường là 0.85).

- $\mathbf{1}$ là vector có tất cả các phần tử đều là 1.
- $\frac{1}{N_p}$ là xác suất rơi vào bất kỳ trang nào trong tổng số N_p trang.

- **Ứng dụng**: Dùng để xếp hạng các trang web hoặc các nút trong đồ thị dựa trên số lượng liên kết và tầm quan trọng của các liên kết đến chúng.

2. Personalized PageRank

Personalized PageRank là phiên bản của PageRank cho phép cá nhân hóa kết quả dựa trên sự quan tâm đặc biệt đến một số nút cụ thể trong đồ thị. Các đặc điểm chính của Personalized PageRank là:

- **Công thức**:

$$P_r(t+1) = d \cdot O_r^P(t) + (1-d) \cdot \mathbf{p}$$

Trong đó:

- $\mathbf{p}$ là vector cá nhân hóa, chỉ ra sự quan tâm đến các nút cụ thể. Các giá trị trong vector này thường là 0 hoặc 1, và tổng của chúng bằng 1.

- **Ứng dụng**: Khi bạn muốn tập trung vào một số nút cụ thể trong đồ thị. Ví dụ, trong một mạng xã hội, bạn có thể muốn tìm các nút quan trọng liên quan đến một người dùng cụ thể.

3. Topic-Specific PageRank

Topic-Specific PageRank là phiên bản của PageRank được điều chỉnh để tập trung vào các chủ đề hoặc loại nội dung cụ thể trong đồ thị. Đây là sự kết hợp giữa PageRank và việc điều chỉnh dựa trên các chủ đề hoặc lĩnh vực quan tâm.

- **Công thức**:

$$P_r(t+1) = d \cdot O_r^P(t) + (1-d) \cdot \mathbf{p}_t$$

Trong đó:

- $\mathbf{p}_t$ là vector chủ đề cụ thể, chứa các xác suất phân phối cho các nút dựa trên chủ đề hoặc lĩnh vực cụ thể.

- **Ứng dụng**: Khi bạn muốn xếp hạng các nút trong một đồ thị dựa trên sự quan tâm đến một chủ đề cụ thể. Ví dụ, trong một mạng lưới tài liệu, bạn có thể muốn xếp hạng các tài liệu dựa trên sự liên quan đến một chủ đề nghiên cứu cụ thể.

Tóm Tắt

- **PageRank**: Đánh giá tầm quan trọng chung của các nút dựa trên liên kết đến chúng.
- **Personalized PageRank**: Tập trung vào sự quan tâm cá nhân hóa đến một số nút cụ thể.
- **Topic-Specific PageRank**: Tập trung vào sự quan tâm đến các chủ đề hoặc lĩnh vực cụ thể trong đồ thị.

Mỗi phiên bản của PageRank phục vụ các mục đích khác nhau và có các ứng dụng phù hợp tùy theo yêu cầu của bài toán cụ thể.

3 Tài liệu tham khảo

Tài liệu

[1] *Learn the PageRank Algorithm from Scratch*. Retrieved from here.

[2] 1998-01, The PageRank Citation Ranking: Bringing Order to the Web. Retrieved from here.

[3] 2010, Pagerank-Based Collaborative Filtering Recommendation. In Lecture Notes in Computer Science. Retrieved from here.

[4] 2014-07-18, PageRank beyond the Web. Retrieved from here.

[5] 2017-04,A PageRank-Based Collaborative Filtering Recommendation Approach in Digital Libraries. Retrieved from here.

[6] 2019-11, A Survey on Personalized PageRank Computation Algorithms. Retrieved from here.

[7] 2021, Context-aware graph-based recommendations exploiting Personalized PageRank. Retrieved from here.