Data Mining Project 3

Link Analysis

F74102030 資訊114 練智剛

Fina a way

Revision Graph

```
rev_graph_1.txt
1,2
2,3
3,4
4,5
5,6
1,7
7,8
9,1
=====rev_graph_1=====
Pagerank:
[0.079 0.077 0.111 0.141 0.169 0.193 0.077 0.111 0.042]

Authority:
[1.526e-05 5.000e-01 1.526e-05 1.526e-05 1.526e-05 1.526e
 1.526e-05 0.000e+00]

Hub:
[9.998e-01 3.051e-05 3.051e-05 3.051e-05 3.051e-05 0.000e
 0.000e+00 3.051e-05]
```

```
rev_graph_2.txt
1,2
2,3
3,4
4,5
5,1
2,1
1,3
=====rev_graph_2=====
Pagerank:
[0.262 0.138 0.2   0.2   0.2  ]
```

```
Authority:
[3.569e-01 1.981e-01 4.450e-01 7.757e-09 7.757e-09]

Hub:
[3.569e-01 4.450e-01 7.757e-09 7.757e-09 1.981e-01]
```

```
rev_graph_3.txt
1,2
2,1
2,3
3,2
3,4
4,3
3,1
1,3
=====rev_graph_3=====
Pagerank:
[0.247 0.247 0.37  0.136]

Authority:
[0.27  0.27  0.315 0.145]

Hub:
[0.27  0.27  0.315 0.145]
```

Algorithm description

1. PageRank

```
def PageRank(G:Graph, max_iters:int, damping_factor:float):
    N = G.N #總共有幾個pages
    d = damping_factor
    PageRanks = np.full(N, 1/N)   #將所有節點的權重初始化為1/N
    for iter in range(max_iters):
        newPageRanks = np.zeros(N) #建立一個長度為N的一維矩陣
        for i in range(N):#計算所有指向節點i的節點的PageRank之和
            for n in G.in_neighbors[i]:
                newPageRanks[i] += PageRanks[n] / len(G.out_neighbors[n])
                #根據PageRank的演算法公式更新每個節點的權重
        PageRanks =  d/N + (1-d) * newPageRanks
        #正規化PageRanks以確保總和為1
    PageRanks = PageRanks / (PageRanks.sum())
    return PageRanks
```

2. HITS

```
def HITS(G:Graph, max_iters:int)-> Tuple[np.array, np.array]:
    auths = np.ones(G.N) #初始時將所有節點的authority設置為1
    hubs = np.ones(G.N)  #初始時將所有節點的hub設置為1

    for _ in range(max_iters):
        new_auths = np.zeros_like(auths) # 用於存儲新的authority分數
        new_hubs = np.zeros_like(hubs) # 用於存儲新的hub分數
        for n in range(G.N):
              new_auths[n] = hubs[G.in_neighbors[n]].sum()
                        #計算新的authority分數，總和為指向節點n的所有節點的hub分數之和
            new_hubs[n] = auths[G.out_neighbors[n]].sum()
                        #計算新的hub分數，總和為節點n指向的所有節點的authority分數之和
                #正規化新的authority和hub分數以確保總和為1
        auths = new_auths / np.sum(new_auths)
        hubs = new_hubs / np.sum(new_hubs)
    return auths, hubs
```

3. SimRank

```
def SimRank(G: Graph,
            max_iters:int,
            decay_factor:float):
    C = decay_factor
    def update_simrank(a:int, b:int, simRank: np.array):
                if a == b: #如果a和b相等代表是同一個點
            return 1
                # 取得節點a和節點b的in_neightbors
        a_in_neighbors = G.in_neighbors[a]
        b_in_neighbors = G.in_neighbors[b]
        a_in_size, b_in_size = len(a_in_neighbors), len(b_in_neighbors)
        if not a_in_size or not b_in_size: # 如果沒有 in_neighbors
            return 0
        temp = 0
                #計算兩個節點的simRank(根據講義的公式)
        for i in a_in_neighbors:
            for j in b_in_neighbors:
                temp += simRank[i, j]
        return C * temp / (a_in_size * b_in_size)
    # 初始化SimRank相似性矩陣
    simRank = np.zeros((G.N, G.N))
    for iter in range(max_iters):
        newSimRank = np.zeros_like(simRank)#創建新的相似性矩陣
        for a in range(G.N):
            for b in range(a, G.N):
                                #對每一對節點更新相似性分數
                newSimRank[b, a] = update_simrank(a, b, simRank)
```

```
                                newSimRank[a, b] = newSimRank[b, a]
        #將新的相似性矩陣複製到原始相似性矩陣
                simRank = newSimRank.copy()
    return simRank
```

Result analysis and discussion

Graph1

```
Pagerank:
[0.056 0.107 0.152 0.193 0.23  0.263]

Authority:
[0.  0.2 0.2 0.2 0.2 0.2]

Hub:
[0.2 0.2 0.2 0.2 0.2 0. ]

SimRank:
[[1. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0.]
[0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 1.]]
```

Graph1是一個線性的圖形，從node1連到node6。node1並沒有parent nodes，因此pagerank的值最低，而node6匯集了前面node的值，而有最高的pagerank。由於是一個單向圖形，所以除了node1的authority是0，其他的值都是一樣的，反之只有node6的hub值是0，其他都是一樣的。SimRank 矩陣顯示所有節點對之間的相似性，對角線上的值為1，表示每個節點與自己相似性為最高

Graph2

```
Pagerank:
[0.2 0.2 0.2 0.2 0.2]

Authority:
[0.2 0.2 0.2 0.2 0.2]

Hub:
[0.2 0.2 0.2 0.2 0.2]

SimRank:
[[1. 0. 0. 0. 0.]
[0. 1. 0. 0. 0.]
[0. 0. 1. 0. 0.]
```

```
[0. 0. 0. 1. 0.]
[0. 0. 0. 0. 1.]]
```

graph2是一個circular graph，所有點的進出數量都是相同的，所以在計算pagerank和hits時每一個node的值都是相等的。SimRank 的結果呈對角線為1，表示每個節點相對於自身的相似度為1，其他節點的相似度則相對較低。

Graph3

```
Pagerank:
[0.172 0.328 0.328 0.172]

Authority:
[0.191 0.309 0.309 0.191]

Hub:
[0.191 0.309 0.309 0.191]

SimRank:
[[1.    0.    0.538 0.   ]
 [0.    1.    0.    0.538]
 [0.538 0.    1.    0.   ]
 [0.    0.538 0.    1.   ]]
```

graph3是一線性的雙向圖，node2和3在圖的中間，所以pagerank的值較高，同樣的authority和hub也因為有較多的指向，因此有比較高的值。node1和node4在SimRank 中的相似度較高，這是因為它們分別指向和被指向的節點相似，而node2和node3之間的相似度也相對較高。

Damping factor in PageRank

用graph3進行討論

```
d = 0.1
[0.172 0.328 0.328 0.172]

d = 0.3
[0.185 0.315 0.315 0.185]

d = 0.5
[0.2 0.3 0.3 0.2]

d = 0.7
[0.217 0.283 0.283 0.217]

d = 1.0
[0.25 0.25 0.25 0.25]
```

從上面的結果可以看出來，較高的 damping factor 會使權重更加均勻，而較低的 damping factor 則可能使得一些node的權重明顯增加。

Decay factor in SimRank

用graph3進行討論

```
c = 0.1
[[1.    0.    0.053 0.   ]
 [0.    1.    0.    0.053]
 [0.053 0.    1.    0.   ]
 [0.    0.053 0.    1.   ]]

c = 0.3
[[1.    0.    0.176 0.   ]
 [0.    1.    0.    0.176]
 [0.176 0.    1.    0.   ]
 [0.    0.176 0.    1.   ]]

c = 0.5
[[1.    0.    0.333 0.   ]
 [0.    1.    0.    0.333]
 [0.333 0.    1.    0.   ]
 [0.    0.333 0.    1.   ]]

c = 0.7
[[1.    0.    0.538 0.   ]
 [0.    1.    0.    0.538]
 [0.538 0.    1.    0.   ]
 [0.    0.538 0.    1.   ]]

c = 1.0
[[1. 0. 1. 0.]
 [0. 1. 0. 1.]
 [1. 0. 1. 0.]
 [0. 1. 0. 1.]]
```

decay factor控制了相似度的下降速度，較高的decay factorc下降的比較快，只有關係很接近的nodes會有高相似度，所以node1,3以及node2,4的相似度上升

Effectiveness analysis

```
graph_1 - PageRank: 0.00100 seconds
graph_1 - HITS: 0.00200 seconds
graph_1 - SimRank: 0.00000 seconds

graph_2 - PageRank: 0.00352 seconds
graph_2 - HITS: 0.00200 seconds
graph_2 - SimRank: 0.00100 seconds
```

```
graph_3 - PageRank: 0.00100 seconds
graph_3 - HITS: 0.00200 seconds
graph_3 - SimRank: 0.00100 seconds

graph_4 - PageRank: 0.00000 seconds
graph_4 - HITS: 0.00298 seconds
graph_4 - SimRank: 0.00200 seconds

graph_5 - PageRank: 0.01798 seconds
graph_5 - HITS: 0.17653 seconds
graph_5 - SimRank: 6.13191 seconds

graph_6 - PageRank: 0.07659 seconds
graph_6 - HITS: 0.28237 seconds

ibm-5000 - PageRank: 0.07333 seconds
ibm-5000 - HITS: 0.20692 seconds
```

graph1相對簡單，節點和邊的數量較少，因此各算法的計算時間都相對較短。graph2中的計算時間相對較長，可能是因為graph2是一個環形結構，導致PageRank的計算時間增加。graph3的計算時間相對較短，可能是因為graph3的結構較為簡單，節點之間的相互影響較少。從上面的統計可以看出，在node數量比較多的graph_5，SimRank的執行時間遠大於兩外兩個演算法，因為在計算每個node的SimRank時，需要兩個兩個node都計算過一次，所以當node數量一多計算時間就會很長。總體來看，圖的複雜度和結構對各算法的計算時間都有影響，並且不同的演算法對於不同結構的圖的計算效率也有所不同。