PageRank

1 PageRank

1.1 模型定义

PageRank 算法是计算互联网网页重要程度的算法，它对每一个网页给出一个正实数（即 PageRank 值），表示网页的重要程度。PageRank 值越高，网页就越重要。PageRank 值依赖网络的拓扑结构，当网络结构确定后，PageRank 值就确定。PageRank 基本定义是在一个**有向图**上的**随机游走模型**。

定义（PageRank 的基本定义）：给定一个包含 n 个节点 $v_1, v_2, ..., v_n$ 的强连通且非周期性的有向图，在有向图上定义随机游走模型（相当于一阶马尔可夫链）。随机游走的特点是从一个节点到有有向边连出的所有节点的转移概率相等，转移矩阵为 M。该马尔可夫链具有平稳分布R，满足 $MR = R$。平稳分布 R 称为这个有向图的 PageRank。

记 $R = [PR(v_1), PR(v_2), ..., PR(v_n)]^T$，易于发现：

$$PR(v_i) \geq 0, \quad i = 1, 2, ..., n$$

$$\sum_{i=1}^{n} PR(v_i) = 1$$

$$PR(v_i) = \sum_{v_j \in M(v_i)} \frac{PR(v_j)}{L(v_j)}, \quad i = 1, 2, ..., n$$

其中，$M(v_i)$ 表示指向 v_i 的节点集合，$L(v_j)$ 表示节点 v_j 连出的有向边的个数。

按以上定义方法定义的 PageRank 必须满足**强连通和非周期性的条件**。但实际情况中，大部分有向图可能存在孤立点等情况，不满足这一条件，致使其平稳分布也不存在。可使用**平滑方法**解决这一问题，可假设存在另一个完全随机游走，其转移矩阵的元素全部为 $\frac{1}{n}$。将两个转移矩阵的线性组合构成一个新的转移矩阵，按新的转移矩阵定义一个新的马尔可夫链。该马尔可夫链的转移矩阵为 $M' = dM + \frac{1-d}{n}1_{n\times n}$，其中 $0 \leq d \leq 1$ 是阻尼因子，$1_{n\times n}$ 是元素均为 1 的 $n \times n$ 的矩阵。可以证明，该马尔可夫链一定具有平稳分布，且平稳分布 R 满足

$$R = dMR + \frac{1-d}{n}1_n$$

R 即为有向图的一般 PageRank，同样可以得到：

$$PR(v_i) > 0, \quad i = 1, 2, ..., n$$

$$\sum_{i=1}^{n} PR(v_i) = 1$$

$$PR(v_i) = d\left(\sum_{v_j \in M(v_i)} \frac{PR(v_j)}{L(v_j)}\right) + \frac{1-d}{n}, \quad i = 1, 2, ..., n$$

注意，由于加入了平滑项，因此所有节点的 PageRank 值都不会为 0。

1.2 PageRank 的计算

PageRank 的计算主要有代数算法，迭代算法和幂法。

1.2.1 代数算法

由于 $R = dMR + \frac{1-d}{n}1_n$，于是

$$(I - dM)R = \frac{1-d}{n}1_n$$

$$R = (I - dM)^{-1}\frac{1-d}{n}1_n$$

当 $0 < d < 1$ 时，代数算法可以求得唯一解。但对于节点数较多的网络，计算逆矩阵的计算量极大，因此该方法不常使用。

1.2.2 迭代算法

迭代算法使用递推式 $R_{t+1} = dMR_t + \frac{1-d}{n}1_n$ 进行迭代，直至收敛。其算法如下：

输入：含有 n 个节点的有向图，转移矩阵 M，阻尼因子 d。

1. 初始化向量 R_0
2. 不断计算 $R_{t+1} = dMR_t + \frac{1-d}{n}1_n$
3. 如果 R_{t+1} 和 R_t 充分接近，停止迭代，令 $R = R_{t+1}$；否则回到 2

输出：有向图的 PageRank 向量 R。

1.2.3 幂法

Perron-Frobenius 定理证明了一般 PageRank 向量 R 是转移矩阵 $A = dM + \frac{1-d}{n}I_{n\times n}$ 的主特征向量。幂法通过近似计算矩阵的主特征值和主特征向量求得有向图的一般 PageRank 值。

n 阶矩阵 A 的主特征值和主特征向量可采用如下的步骤求得。首先，任取一个 n 维向量 x_0，构造如下序列

$$x_0, \quad x_1 = Ax_0, \quad x_2 = Ax_1, \quad ..., \quad x_k = Ax_{k-1}$$

记 A 的 n 个特征值按绝对值大小排列有 $|\lambda_1| \geq |\lambda_2| \geq ... \geq |\lambda_n|$，对应的特征向量为 $u_1, u_2, ..., u_n$。将初始向量 x_0 表示为 $u_1, u_2, ..., u_n$ 的线性组合：$x_0 = a_1u_1 + a_2u_2 + ... + a_nu_n$。因此有

$$\begin{aligned} x_k = A^kx_0 &= a_1A^ku_1 + a_2A^ku_2 + ... + a_nA^ku_k \\ &= a_1\lambda_1^ku_1 + a_2\lambda_2^ku_2 + ... + a_n\lambda_n^ku_n \\ &= a_1\lambda_1^k\left[u_1 + \frac{a_2}{a_1}\left(\frac{\lambda_2}{\lambda_1}\right)^k u_2 + ... + \frac{a_n}{a_1}\left(\frac{\lambda_n}{\lambda_1}\right)^k u_n\right] \end{aligned}$$

当 $k \to +\infty, x_k \to a_1\lambda_1^ku_1$，因此

$$\begin{aligned} x_k &\approx a_1\lambda_1^ku_1 \\ x_{k+1} &\approx a_1\lambda_1^{k+1}u_1 \end{aligned}$$

因此，$\lambda_1 \approx \frac{x_{k+1,j}}{x_{k,j}}$。此外，在每一次的迭代过程中，需要进行规范化操作，即：

$$\begin{aligned} y_{t+1} &= Ax_t \\ x_{t+1} &= \frac{y_{t+1}}{||y_{t+1}||_\infty} \end{aligned}$$

其中，$||y_{t+1}||_\infty = \max\{|x_1|, |x_2|, ..., |x_n|\}$。因此，PageRank 的幂法算法如下：

输入：含有 n 个节点的有向图，转移矩阵 M，阻尼因子 d。

1. 初始化向量 x_0
2. 计算转移矩阵 $A = dM + \frac{1-d}{n}I_{n\times n}$
3. 计算并规范化

$$\begin{aligned} y_{t+1} &= Ax_t \\ x_{t+1} &= \frac{y_{t+1}}{||y_{t+1}||_\infty} \end{aligned}$$

4. 当 $||x_{t+1} - x_t|| < \epsilon$ 时停止迭代，令 $R = x_{t+1}$；否则，回到 3
5. 对 R 进行规范化操作，使其变为概率分布

输出：有向图的 PageRank 向量 R。

2 代码实现

此次考虑的有向图的转移矩阵为：

$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

PageRank 算法在 sklearn 中没有接口，但在 scikit-network 中有接口。此处我们不调用接口，直接对迭代算法和幂法进行实现。

准备数据：

```
import numpy as np
M = np.array([[0,1/2,0,0],
              [1/3,0,0,1/2],
              [1/3,0,1,1/2],
              [1/3,1/2,0,0]])
```

2.1 迭代算法

```
class IterPageRank:

    def __init__(self,d=0.8,init=None):
        self.d = d
        self.init = init

    def fit(self,M):
        n = M.shape[0]
        if self.init is None:
            r = np.ones((n,1))/n
        else:
```

```
            r = self.init

        err = 1
        while err>1e-3:
            r_new = self.d*M.dot(r)+(1-self.d)/n*np.ones((n,1))
            err = np.mean(np.abs(r_new-r))
            r = r_new

        self.pr = r
```

结果为：

```
clf = IterPageRank()
clf.fit(M)
print(clf.pr)
```

```
## [[0.10180032]
##  [0.12903271]
##  [0.64013426]
##  [0.12903271]]
```

2.2 幂法

```
class PowerPageRank:

    def __init__(self,d=0.8,init=None):
        self.d = d
        self.init = init

    def fit(self,M):
        n = M.shape[0]
        if self.init is None:
            r = np.ones((n,1))/n
        else:
            r = self.init
        A = self.d*M+(1-self.d)/n*np.ones((n,1))

        err = 1
        while err>1e-3:
```

```
            r_new = A.dot(r)
            r_new = r_new/np.max(r_new)
            err = np.mean(np.abs(r_new-r))
            r = r_new

        self.pr = r/np.sum(r)
```

结果为：

```
clf = PowerPageRank()
clf.fit(M)
print(clf.pr)
```

```
## [[0.10180032]
##  [0.12903271]
##  [0.64013426]
##  [0.12903271]]
```