PageRank 的算法研究

一、引言

PageRank 是 Google 搜索中的核心算法，对于按用户提交的关键词搜索得到的网页，基于这些网页在互联网中相互之间的浩瀚链接关系，PageRank 算法对网页进行等级评分（称为 PR 值）排序，其主要思想是同时考虑各网页被链接数量以及各网页本身的重要性，据此，算法较 TF-IDF 而言，具有颠覆性的意义。[1]

二、数学原理

（一）矩阵

1 矩阵的定义

由 m×n 个数a_{ij} (i= 1，2，…，m；j= 1，2，…，n)排成的 m 行 n 列的数表

$$\begin{matrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{matrix}$$

称为 m 行 n 列矩阵，简称$m \times n$ 矩阵。

为表示它是一个整体，总是加一个括弧，并用大写黑体字母表示它，记作

$$\mathrm{A}=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

这$m \times n$个数称为矩阵 A 的元素，简称为元。

数a_{ij}位于矩阵 A 的第i行 第j列，称为矩阵 A 的(i,j)元。以数 a_{ij}为(i,j)元的矩阵可简记作(a_{ij})或$(a_{ij})m \times n$，$m \times n$矩阵 A 也记作 $Am \times n$。

2 矩阵的计算

（1）加法

定义：设有两个$m \times n$阵 A=(a_{ij})和 B=(b_{ij})，那么矩阵 A 与 B 的和记作$A+B$

规定：只有当两个矩阵是同型矩阵时，这两个矩阵才能进行加法运算

规律：矩阵加法满足下列运算规律（设 A，B，C 都是$m \times n$矩阵）：

（i） A +B = B +A；

（ii）（A +B）+ C = A +（B +C）

设矩阵 A = (a_{ij})，记 -A = $(-a_{ij})$， -A 称为矩阵 A 的负矩阵

显然有 A +（-A）= O， 由此规定矩阵的减法为 A - B = A +（- B）

（2）数与矩阵相乘

定义：数 λ 与矩阵 A 的乘积记作A^{λ}

规定：

$$\lambda A = A\lambda = \begin{pmatrix} \lambda a_{11} & \lambda a_{12} & \cdots & \lambda a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \cdots & \lambda a_{2n} \\ \vdots & & & \vdots \\ \lambda a_{m1} & \lambda a_{m2} & \cdots & \lambda a_{mn} \end{pmatrix}$$

规律：数乘矩阵满足下列运算规律（设 A、B 为 $m \times n$ 矩阵， λ 、 μ 为数）：

（i）（ λ μ ）A = λ （ μ A）

（ii）（ λ + μ ）A = λ A + μ A

（iii） λ （A +B）= λ A + λ B

矩阵加法与数乘矩阵统称为矩阵的线性运算。

（3）矩阵与矩阵的乘法

定义：只有当第一个矩阵（左矩阵）的列数等于第二个矩阵（右矩阵）的 行数时，两个矩阵才能相乘。

设 A=$(a_i)_j$是一个$m \times s$矩阵，B=$(b_i)_j$是一个$s \times n$矩阵，那么规定矩阵 A 与矩阵 B 的乘积是一个$m \times n$矩阵 C =(b_{ij})，其中

$$C_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{is}b_{sj} = \sum a_{ik}b_{kj}\ (i = 1,2,\ldots,m; j = 1,2,\ldots,n)$$

并把此乘积记作 C = A B. 按此定义，一个 1×s 行矩阵与一个 s×1 列矩阵的乘

积是一个 1 阶方阵，也就 是一个数。

由此表明乘积矩阵 A B = C 的(i,j)元c_{ij}就是 A 的第i行与 B 的第 j 列的乘积。

规律：矩阵的乘法不满足交换律，即在一般情形下，A B≠BA。对于两个 n 阶方阵 A、B，若 A B = BA，则称方阵 A 与 B 是可交换的。

矩阵的乘法虽不满足交换律，但仍满足下列结合律和分配律（假设运算都 是可行的）：

（i）（A B）C = A（B C）

（ii）λ（A B）=（λA）B = A（λB）（其中λ为数）

（iii） A（B + C）= A B +A C，（B + C）A = BA + CA

（4）矩阵的转置

定义：把矩阵 A 的行换成同序数的列 得到一个新矩阵，叫做 A 的转置矩阵，记作A^T。

规律：矩阵的转置也是一种运算，满足下述运算规律（假设运算都是可行的）：

（i）$(A^T)^T$ = A

（ii）$(A+B)^T = A^T + B^T$

（iii） $(\lambda A)^T = \lambda A^T$

（iv）$(AB)^T = A^T B^T$

（二）一阶马尔可夫链

随机过程就是一些统计模型，利用这些统计模型可以对自然界的一些事物进行预测和处理。

考虑一个随机变量的序列 X ={$X_0, X_1, \cdots, X_t, \cdots$}，这里的 X_t 表示时刻 t 的随机变量，t=0，1，2，…。每个随机变量X_t (t=0，1，2，…)的取值集合相同，称为状态空间，表示为 S。随机变量可以是离散的，也可以是连续的。以上随机变量的序列构成随机过程（stochastic process）。

假设在时刻 0 的随机变量 X0 遵循概率分布 P（X0）=π_0，称为初始状态分布。在某个时刻 t≥1 的随机变量 Xi 只依赖与 Xt-1，而不依赖于过去的随机变量 X={X_0，X_1，…，X_{t-2}} ，这一性质称为马尔可夫性，即

$$P(X_t|X_0, X_1, \cdots, X_{t-1}) = P(X_{t+1}|X_t) t = 0,1,2,\cdots$$

具有马尔可夫性的随机序列 X={$X_0, X_1, \cdots, X_t, \cdots$} ，称为马尔可夫链（Markov chain），或马尔可夫过程（Markov process）。条件概率分布 P(Xt|Xt-1)称为马尔可夫链的转移概率分布。转移概率分布决定了马尔可夫链的特性。

（三）马尔可夫模型

假设我们的序列状态是

$$\cdots X_{t-2}, X_{t-1}, X_t, X_{t+1} \cdots$$

，那么我们的在时刻 Xt+1 的状态的条件概率仅仅依赖于时刻X_t ，即：

$$P(X_{t+1}|\cdots, X_{t-2}, X_{t-1}, X_t) = P(X_{t+1}|X_t)$$

既然某一时刻状态转移的概率只依赖于它的前一个状态，那么只要能求出系统中任意两个状态之间的转换概率，这个马尔科夫链的模型就确定了。

三、PageRank 定义

在实际应用中许多数据都以图（graph)的形式存在，比如，互联网、社交网络都可以看作是一个图，图数据上的机器学习具有理论与应用上的重要意义。

PageRank 算法是图的链接分析（link analysis）的代表性算法，属于图数据上的无监督学习方法。

PageRank 可以定义在任意有向图上，后来被应用到社会影响力分析、文本摘要等多个问题。

（一） 基本思想

历史上，PageRank 算法作为计算互联网网页重要度的算法被提出。PageRank 是定义在网页集合上的一个函数，它对每个网页给出一个正实数，表示网页的重要程度，整体构成一个向量，PageRank 值越高，网页就越重要，在互联网搜索的排序中可能就被排在前面。

假设互联网是一个有向图，在其基础上定义随机游走模型，即一阶马尔可夫链，表示网页浏览者在互联网上随机浏览网页的过程。假设浏览者在每个网页依照连接出去的超链接以等概率跳转到下一个网页，并在网上持续不断进行这样的随机跳转，这个过程形成一阶马尔可夫链。PageRank 表示这个马尔可夫链的平稳分布。每个网页的 PageRank 值就是平稳概率。

图 1 表示一个有向图，假设是简化的互联网实例，节点 A,B,C,D 表示网页，有向边结点之间的有向边表示网页之间的超链接，边的权值表示网页之间随机跳转的概率。

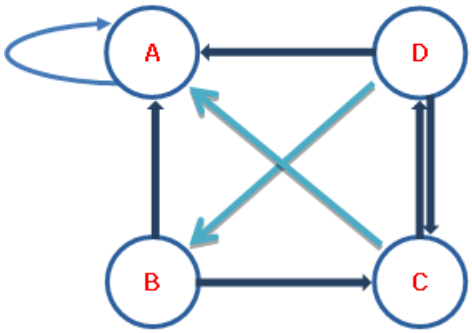


图 1

假设有一个浏览者，在网上随机游走，如果浏览者在网页 B，则下一步以 1/2 的概率前往 A，1/2 的概率前往 B。

一个网页，如果指向该网页的超链接越多，随机跳转到该网页的概率也就越高，该网页的 PageRank 值就越高，这个网页也就越重要。

一个网页，如果指向该网页的 PageRank 值越高，随机跳转到该网页的概率也就越高，该网页的 PageRank 值就越高，这个网页也就越重要。

PageRank 值依赖于网络的拓扑结构，一旦网络的拓扑(连接关系)确定，PageRank 值就确定。

（二）有向图和随机游走模型

1 有向图

有向图（directed graph）记作 G=（V，E），其中 V 和 E 分别表示结点和有向边的集合。

比如，互联网就可以看作是一个有向图，每个网页是有向图的一个结点，网页之间的每一条超链接是有向图的一条边。

强连通图：从一个结点出发到达另一个结点，所经过的边的一个序列称为一条路径 （path），路径上边的个数称为路径的长度。如果一个有向图从其中任何一个结点出发可以到达其他任何一个结点，就称这个有向图是强连通图（strongly connected graph）。

周期性图：假设 k 是一个大于 1 的自然数，如果从有向图的一个结点出发返回到这个结点的路径的长度都是[公式]的倍数，那么称这个结点为周期性结点。如果一个有向图不含 有周期性结点，则称这个有向图为非周期性图 （aperiodic graph），否则为周期性图。

图 2 是一个周期性有向图的例子。从结点 A 出发返回到 A，必须经过路径 A-B-C-A ，所有可能的路径的长度都是 3 的倍数，所以结点 A 是周期性结点。 这个有向图是周期性图。A 是周期性结点。 这个有向图是周期性图。

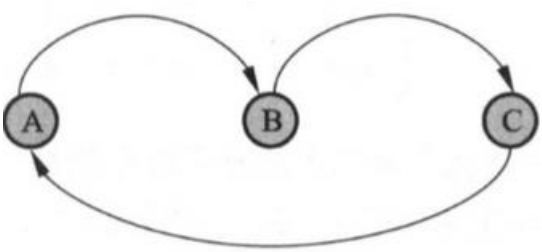


图 2

2 随机游走模型

(1) 定义

给定一个含有 n 个结点的有向图，在有向图上定义随机游走（random walk）模型，即一阶马尔可夫链，其中结点表示状态，有向边 表示状态之间的转移，假设一个结点到通过有向边相连的所有结点的转移概率相等。

具体来说，转移矩阵是一个 n 阶矩阵 M

$$M = [m_{ij}]_{m\times n}$$

第i行 第 j 列的元素m_{ij}取值规则如下：如果结点 j 有k条有向边连出，并且结点i是其连出的一个结点，则$m_{ij} = 1/k$；否则$m_{ij} = 0, i, j = 1,2,\cdots,n$

(2) 转移矩阵的性质

$$m_{ij} \geq 0$$

$$\sum_{i=1}^{n} m_{ij}$$

即每个元素非负，每列元素之和为 1，即矩阵 M 为随机矩阵（stochastic matrix)。

(3) 随机游走

随机游走者每经一个单位时间转移一个状态，如果当前时刻在第 j 个结点（状

态），那么下一个时刻在第i个结点（状态）的概率是m_{ij}，这一概率只依赖于当前的状态，与过去无关，具有马尔可夫性。

根据图 1 和上述描述，可以得到每个单位时间内的转移概率表。

	A	B	C	D
A	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$
B	0	0	0	$\frac{1}{3}$
C	0	$\frac{1}{2}$	0	$\frac{1}{3}$
D	0	0	$\frac{1}{2}$	0

从而，写出该随机游走的状态转移矩阵

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

随机游走在某个时刻 t 访问各个结点的概率分布就是马尔可夫链在时刻 t 的状态分布，可以用一个 n 维列向量 Rt 表示，那么在时刻 t+1 访问各个结点的概率分布 R_{t+1} 满足

$$R_{t+1} = MR_t$$

给定一个包含 n 个结点的强连通且非周期性的有向图，在其基础上定义随机游走模型：

假设转移矩阵为 M，在时刻 0,1,2,… ,t,⋯访问各个结点的概率分布为

$$R_0, MR_0, M^2R_0, \cdots, M^tR_0$$

则极限$\lim_{t\to\infty} M^tR_0 = R$存在，极限向量 R 表示马尔可夫链的平稳分布，满足 MR=R。

（三）PageRank 的基本定义

1 一阶马尔可夫链

不可约 且非周期的有限状态马尔可夫链，有唯一平稳分布存在，并且当时

间趋于无穷时状态分布收敛于 唯一的平稳分布。

根据马尔可夫链平稳分布定理，强连通且非周期的有向图上定义的随机游走模型（马尔可夫链），在图上的随机游走当时间趋近于无穷时状态分布收敛于 唯一的平稳分布。

2 具体定义

给定一个包含 n 个结点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，即一阶马尔可夫链。随机游走模型的特点是从一个结点到 有向边给出的所有结点的转移概率相等，转移矩阵为 M。

这个马尔可夫链具有平稳分布 R：MR = R

平稳分布 R 称为这个有向图的 PageRank。R 的各个分量称为各个结点的 PageRank 值。

$$R = \begin{bmatrix} PR(v1) \\ PR(v2) \\ \vdots \\ PR(vn) \end{bmatrix}$$

其中，$PR(v_i), i = 1,2,\cdots,n$表示结点 v_i 的 PageRank 值。

那么：

$$PR(v_i) \geq 0, \qquad i = 1,2,\cdots,n$$

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

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

M(v_i)表示指向结点v_i的结点集合。

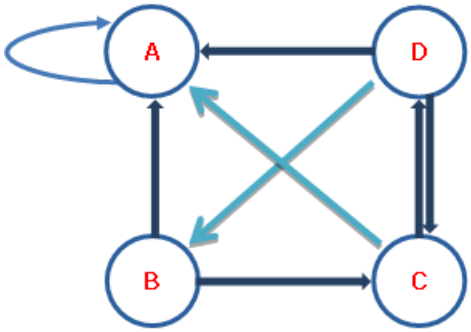
L(v_j)表示结点v_j连出的有向边的个数。

PageRank 的基本定义是理想化的，在这中情况下，PageRank 存在，而且可以通过不断迭代求得 PageRank 值。

四、案例分析

（一）案例描述

根据以下有向图，进行 PageRank 的数学分析和程序设计。



（二）状态转移矩阵和初始分布向量

根据上文的分析，状态转移矩阵为

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

根据随机游走模型，用户访问每个网页的可能性相同，故取初始分布向量 R_0 为

$$R0 = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}$$

（三）进行迭代运算

设计矩阵相乘的 python 程序进行理论验证。

1 单步矩阵相乘

```
import numpy as np
a=np.array([[1,1/2,1/2,1/3],[0,0,0,1/3],[0,1/2,0,1/3],[0,0,1/2,0]])
b=np.array([1/4,1/4,1/4,1/4])
print(np.dot(a,b))
```

以转移矩阵 M 乘初始向量 R0 得

```
[0.58333333 0.08333333 0.20833333 0.125      ]
```

2 设计迭代相乘

```
import numpy as np
a=np.array([[1,1/2,1/2,1/3],[0,0,0,1/3],[0,1/2,0,1/3],[0,0,1/2,0]])
b=np.array([1/4,1/4,1/4,1/4])
while 1:
    PR=np.dot(a,b)
    print(PR)
    b=PR
```

以转移矩阵 M 连乘初始向量 R_0 得到向量序列：

$$R0 = [1\ \ 0\ \ 0\ \ 0]$$

```
[1.0000000e+000 1.7584849e-317 3.3247757e-317 2.9613939e-317]
[1.0000000e+000 9.8713130e-318 1.8663735e-317 1.6623876e-317]
[1.000000e+000 5.541292e-318 1.047695e-317 9.331867e-318]
[1.000000e+000 3.110622e-318 5.881269e-318 5.238474e-318]
[1.000000e+000 1.746156e-318 3.301465e-318 2.940634e-318]
[1.000000e+000 9.802114e-319 1.853290e-318 1.650733e-318]
[1.00000e+000 5.50246e-319 1.04035e-318 9.26645e-319]
[1.00000e+000 3.08880e-319 5.84005e-319 5.20172e-319]
[1.00000e+000 1.73392e-319 3.27832e-319 2.92003e-319]
[1.00000e+000 9.73359e-320 1.84035e-319 1.63916e-319]
[1.00000e+000 5.46387e-320 1.03304e-319 9.20148e-320]
[1.0000e+000 3.0672e-320 5.7993e-320 5.1650e-320]
[1.0000e+000 1.7218e-320 3.2554e-320 2.8997e-320]
[1.0000e+000 9.6639e-321 1.8271e-320 1.6275e-320]
[1.0000e+000 5.4248e-321 1.0257e-320 9.1353e-321]
[1.000e+000 3.043e-321 5.756e-321 5.128e-321]
[1.000e+000 1.709e-321 3.231e-321 2.875e-321]
[1.000e+000 9.585e-322 1.813e-321 1.616e-321]
[1.00e+000 5.39e-322 1.02e-321 9.09e-322]
[1.0e+000 3.0e-322 5.7e-322 5.1e-322]
[1.00e+000 1.68e-322 3.16e-322 2.87e-322]
[1.0e+000 9.4e-323 1.8e-322 1.6e-322]
[1.00e+000 5.43e-323 1.04e-322 8.89e-323]
[1.e+000 3.e-323 6.e-323 5.e-323]
[1.0e+000 1.5e-323 3.0e-323 3.0e-323]
[1.0e+000 9.9e-324 2.0e-323 1.5e-323]
[1.e+000 5.e-324 1.e-323 1.e-323]
[1.e+000 5.e-324 5.e-324 5.e-324]
[1. 0. 0. 0.]
```

3 非理想结果

通过 R_0 的最终迭代情况可以发现，PR 值陷入了一种 A 网站的权威值最高（为 1），其他网站的权威值为 0 的情况，在这里我把它称为“采集器陷阱”。

五、两大常见问题和解决方法

实际上，案例所给模型仍旧欠缺了另一种问题的考虑，下面针对出现的问题进行分析与修改。

（一）常见问题

1 终止点问题

问题描述：某一些网站没有出链。

修改的思路：

方法一：修改 M 矩阵；

方法二：给终止点增加一个链向其他所有外联网的连接

方法三：删除终止点

2 采集器陷阱问题

问题描述：一组网页不 链向他们之外的其他网页

修改的思路：抽税法：给每一个网页增加指向所有页面的链接，并给每个链接一个参数（B-阻尼系数）控制的转移概率。

（二）解决方法—— PageRank 的一般定义

给定一个含有 n 个结点的任意有向图，在有向图上定义一个一般的随机游走模型，即一阶马尔可夫链。一般的随机游走模型的转移矩阵由两部分的线性组合组成，一部分是有向图的基本转移矩阵 M，表示从一个结点到其连出的所有结点的转移概率相等，另一部分是完全随机的转移矩阵，表示从任意一个结点到任意一个结点的转移概率都是 1/n，线性组合系数为阻尼因子 d(0≤d≤1)。这个一般随机游走的马尔可夫链存在平稳分布，记作 R。定义平稳分布向量 R 为这个有向图的一般 PageRank。R 由公式

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

决定，其中 1 是所有分量为 1 的 n 维向量。

一般 PageRank 的定义意味着互联网浏览者，按照以下方法在网上随机游走：

在任意一个网页上，浏览者

1 要么以概率 d 决定按照超链接随机跳转，这时所有浏览者以该网页中存在

的超链接为基础，等概率从连接出去的超链接跳转到下一个网页。

2 要么以概率（1-d)决定完全随机跳转，这时以等概率 1/n 跳转到任意一个网页。

这个机制保证从没有连接出去的超链接的网页也可以跳转出，从而解决了终止点问题和采集器陷阱问题。这样即可以保证平稳分布，又实现了模拟真实用户的浏览情况。

一般 PageRank 的存在因此适用于任何结构的网络。

（三）案例再分析

根据以上模型，预设 d=0.85。

1 单步矩阵相乘

```
import numpy as np
d=0.85
a=np.array([[1,1/2,1/2,1/3],[0,0,0,1/3],[0,1/2,0,1/3],[0,0,1/2,0]])
b=np.array([1/4,1/4,1/4,1/4])
R=d*np.dot(a,b)+(1-d)/4*np.ones(4)#增加以 d 为基础的完全随机概率的矩阵
print(R)
```

以转移矩阵 M 乘初始向量 R0 并增加阻尼系数得

```
[0.53333333 0.10833333 0.21458333 0.14375   ]
```

2 设计迭代相乘

```
import numpy as np
d=0.85
a=np.array([[1,1/2,1/2,1/3],[0,0,0,1/3],[0,1/2,0,1/3],[0,0,1/2,0]])
b=np.array([1/4,1/4,1/4,1/4])
while 1:
    R=d*np.dot(a,b)+(1-d)/4*np.ones(4)#增加以 d 为基础的完全随机概率的矩阵
    print(R)
    b=R
```

以转移矩阵 M 连乘初始向量 R0 并增加阻尼系数得

```
[0.53333333 0.10833333 0.21458333 0.14375   ]
[0.66880208 0.07822917 0.12427083 0.12869792]
[0.72850868 0.07396441 0.10721181 0.0903151 ]
[0.75932155 0.06308928 0.09452415 0.08306502]
[0.77344411 0.06103509 0.08784803 0.07767277]
[0.78021011 0.05950728 0.0854472  0.07483541]
[0.78348761 0.05870337 0.08399396 0.07381506]
[0.7850251  0.05841427 0.0833632  0.07319743]
[0.78576603 0.05823927 0.08306534 0.07292936]
[0.7861189  0.05816332 0.08291501 0.07280277]
[0.78628681 0.05812745 0.08284686 0.07273888]
[0.78636722 0.05810935 0.08281352 0.07270992]
[0.7864055  0.05810114 0.08279762 0.07269574]
[0.78642377 0.05809713 0.08279011 0.07268899]
[0.7864325  0.05809521 0.08278649 0.0726858 ]
[0.78643666 0.05809431 0.08278477 0.07268426]
[0.78643864 0.05809387 0.08278395 0.07268353]
[0.78643959 0.05809367 0.08278356 0.07268318]
[0.78644004 0.05809357 0.08278338 0.07268301]
[0.78644026 0.05809352 0.08278329 0.07268293]
[0.78644036 0.0580935  0.08278324 0.0726829 ]
[0.78644041 0.05809349 0.08278322 0.07268288]
[0.78644043 0.05809348 0.08278321 0.07268287]
[0.78644044 0.05809348 0.08278321 0.07268287]
[0.78644045 0.05809348 0.08278321 0.07268286]
[0.78644045 0.05809348 0.08278321 0.07268286]
[0.78644045 0.05809348 0.08278321 0.07268286]
```

最终，PR 值稳定在[0.78644045 0.05809348 0.08278321 0.07268286]。

至此，以上终止点问题和采集器陷阱问题得到了有效的解决。

六、算法设计与优化

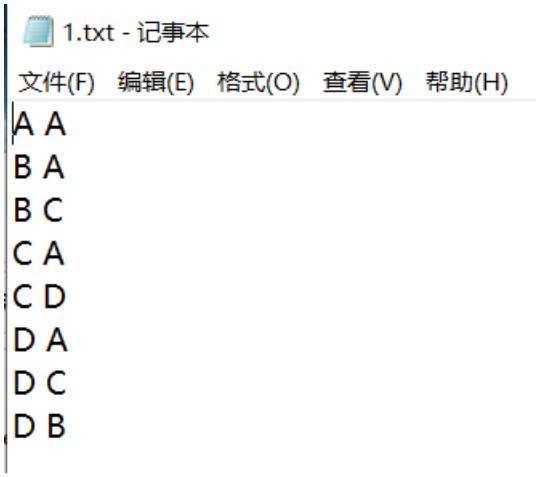
完成了案例分析后，PageRank 的核心模块就完成了。接下来只需要设计读入有向图的模块就可以了。

（一）初始算法设计

1 有向图和状态转移矩阵

(1) 需求分析

根据案例的有向图，下面为四个网站之间的超链接关系。



根据超链接关系生成，状态转移矩阵。

(2) 算法设计

读入有向图，存储边
根据边获取节点集合
获取结点的出链个数
序号转化
循环遍历初始矩阵并归一化

(3) 代码设计

```
import numpy as np
# 读入有向图，存储边
edges=[]
f = open('1.txt', 'r')
for line in f:
```

```
    edges.append(line.strip('\n').split(' '))
print(edges)
# 根据边获取节点的集合
nodes = []
for edge in edges:
    if edge[0] not in nodes:
        nodes.append(edge[0])
    if edge[1] not in nodes:
        nodes.append(edge[1])
print(nodes)
# 获取节点出链的个数
sides=dict()
for edge in edges:
    if edge[0] not in sides:
        sides[edge[0]]=1
    elif edge[0] in sides:
        sides[edge[0]] += 1
print(sides)

#序号转化
edges_num=[]
for edge in edges:
    line=[]
    for i in edge:
        if i=='A':
            i=0
        elif i=='B':
            i=1
        elif i=='C':
            i=2
        elif i=='D':
            i=3
        line.append(i)
    edges_num.append(line)
print(edges_num)
M = np.zeros([len(nodes),len(nodes)])#生成空矩阵
#循环遍历归一化
for i in edges_num:
    M[i[1],i[0]]=i[0]+1
    if M[i[1],i[0]]==1:
        M[i[1], i[0]]=1/sides['A']
    elif M[i[1],i[0]]==2:
        M[i[1], i[0]]=1/sides['B']
    elif M[i[1],i[0]]==3:
        M[i[1], i[0]]=1/sides['C']
    elif M[i[1],i[0]]==4:
        M[i[1], i[0]]=1/sides['D']
print(M)
```

（4）输出

```
[['A', 'A'], ['B', 'A'], ['B', 'C'], ['C', 'A'], ['C', 'D'], ['D', 'A'], ['D', 'C'], ['D', 'B']]
['A', 'B', 'C', 'D']
{'A': 1, 'B': 2, 'C': 2, 'D': 3}
[[0, 0], [1, 0], [1, 2], [2, 0], [2, 3], [3, 0], [3, 2], [3, 1]]
[[1.         0.5        0.5        0.33333333]
 [0.         0.         0.         0.33333333]
 [0.         0.5        0.         0.33333333]
 [0.         0.         0.5        0.        ]]
```

2 随机游走计算 PageRank 值

使用案例中，改进后的 PageRank 算法即可。

```
import numpy as np
d=0.85
a=np.array([[1,1/2,1/2,1/3],[0,0,0,1/3],[0,1/2,0,1/3],[0,0,1/2,0]])
b=np.array([1/4,1/4,1/4,1/4])
while 1:
    R=d*np.dot(a,b)+(1-d)/4*np.ones(4)#增加以 d 为基础的完全随机概率的矩
阵
    print(R)
    b=R
```

3 预期误差达到平稳分布

（1）需求分析

误差即精度，需要计算到什么情况下停止。按照一阶马尔可夫链的原理，最终的迭代结果将会趋近平稳分布，那么就会出现三种情况：设置的精度迭代结束后未平稳分布；设置的精度迭代后接近平稳分布；设置的精度迭代结束前就平稳分布。

正常情况为“设置的精度迭代后接近平稳分布”。

就“设置的精度迭代结束后未平稳分布”而言，既然是设置了精度本身就是估计，即“在误差范围内”可以接受的，无需在考虑该情况。

就“设置的精度迭代结束前就平稳分布”而言，只需要在设计算法的时候予以考虑即可。

（2）算法设计

设计误差为循环结束条件
误差为当前矩阵同下一次迭代后的矩阵差的绝对值
设置误差值为 float 的精准度即可

（3）代码设计

```
import numpy as np
d=0.85
a=np.array([[1,1/2,1/2,1/3],[0,0,0,1/3],[0,1/2,0,1/3],[0,0,1/2,0]])
b=np.array([1/4,1/4,1/4,1/4])
e=1
while e > 0.00000001:
    R=d*np.dot(a,b)+(1-d)/4*np.ones(4)#增加以 d 为基础的完全随机概率的矩
阵
    print(R)
    e=R-b
    e= max(map(abs, e))
    b=R
```

（二）算法优化设计

1 优化角度

提高代码适应性：使得任意数量的网站都可以进行计算

降低代码的冗余程度：将相同的分支结构合并

2 算法重设计

序号转化重设计：使用 ASCII 转化为 DEC 数字的 chr 和 ord 函数直接映射

分支结构重设计：将“获取边结点的集合”同“获取边的数量”一同计算

3 代码重设计

```
import numpy as np
# 读入有向图，存储边
edges=[]
f = open('1.txt', 'r')
for line in f:
```

```
    edges.append(line.strip('\n').split(' '))
print(edges)


nodes = []
sides=dict()
for edge in edges:
    edge[0] = ord(edge[0]) - 65
    edge[1] = ord(edge[1]) - 65
    # 根据边获取结点的集合
    if edge[0] not in nodes:
        nodes.append(edge[0])
    if edge[1] not in nodes:
        nodes.append(edge[1])
    # 根据边获取边的字典
    if edge[0] not in sides:
        sides[edge[0]] = 1
    elif edge[0] in sides:
        sides[edge[0]] += 1
print(nodes)
print(edges)
print(sides)

M = np.zeros([len(nodes),len(nodes)])#生成空矩阵
#循环遍历归一化
for i in edges:
    M[i[1],i[0]]=i[0]+1
    for k in range(len(nodes)):
        if M[i[1],i[0]]==k+1:
            M[i[1], i[0]]=1/sides[k]
print(M)
```

```
[['A', 'A'], ['B', 'A'], ['B', 'C'], ['C', 'A'], ['C', 'D'], ['D', 'A'], ['D', 'C'], ['D', 'B']]
[0, 1, 2, 3]
[[0, 0], [1, 0], [1, 2], [2, 0], [2, 3], [3, 0], [3, 2], [3, 1]]
{0: 1, 1: 2, 2: 2, 3: 3}
[[1.         0.5        0.5        0.33333333]
 [0.         0.         0.         0.33333333]
 [0.         0.5        0.         0.33333333]
 [0.         0.         0.5        0.        ]]
```

（三）最终成果

1 最终代码

```
import numpy as np
# 读入有向图，存储边
edges=[]
f = open('1.txt', 'r')
for line in f:
    edges.append(line.strip('\n').split(' '))
```

```
print(edges)


nodes = []
sides=dict()
for edge in edges:
    edge[0] = ord(edge[0]) - 65
    edge[1] = ord(edge[1]) - 65
    # 根据边获取结点的集合
    if edge[0] not in nodes:
        nodes.append(edge[0])
    if edge[1] not in nodes:
        nodes.append(edge[1])
    # 根据边获取边的字典
    if edge[0] not in sides:
        sides[edge[0]] = 1
    elif edge[0] in sides:
        sides[edge[0]] += 1
print(nodes)
print(edges)
print(sides)

M = np.zeros([len(nodes),len(nodes)])#生成空矩阵
#循环遍历归一化
for i in edges:
    M[i[1],i[0]]=i[0]+1
    for k in range(len(nodes)):
        if M[i[1],i[0]]==k+1:
            M[i[1], i[0]]=1/sides[k]
print(M)

print("开始计算 PR 值......")
d=0.85
page_num = len(nodes)
#计算初始分布向量 R
R0 = np.ones(page_num)
for i in range(page_num):
    R0[i]=1/page_num
e=1
while e > 0.00000001:
    R=d*np.dot(M,R0)+(1-d)/4*np.ones(4)#增加以 d 为基础的完全随机概率的
矩阵
    print(R)
    e=R-R0
    e= max(map(abs, e))
    R0=R
```

2 结果截图

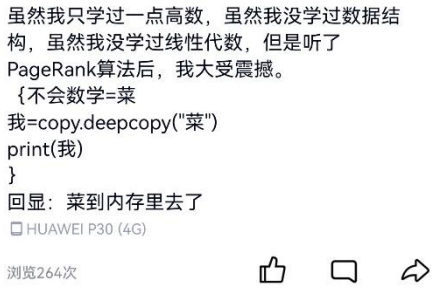
```
[['A', 'A'], ['B', 'A'], ['B', 'C'], ['C', 'A'], ['C', 'D'], ['D', 'A'], ['D', 'C'], ['D', 'B']]
[0, 1, 2, 3]
[[0, 0], [1, 0], [1, 2], [2, 0], [2, 3], [3, 0], [3, 2], [3, 1]]
{0: 1, 1: 2, 2: 2, 3: 3}
[[1.         0.5        0.5        0.33333333]
 [0.         0.         0.         0.33333333]
 [0.         0.5        0.         0.33333333]
 [0.         0.         0.5        0.        ]]
开始计算PR值
[0.53333333 0.10833333 0.21458333 0.14375   ]
[0.66880208 0.07822917 0.12427083 0.12869792]
[0.72850868 0.07396441 0.10721181 0.0903151 ]
[0.75932155 0.06308928 0.09452415 0.08306502]
[0.77344411 0.06103509 0.08784803 0.07767277]
[0.78021011 0.05950728 0.0854472  0.07483541]
[0.78348761 0.05870337 0.08399396 0.07381506]
[0.7850251  0.05841427 0.0833632  0.07319743]
[0.78576603 0.05823927 0.08306534 0.07292936]
[0.7861189  0.05816332 0.08291501 0.07280277]
[0.78628681 0.05812745 0.08284686 0.07273888]
[0.78636722 0.05810935 0.08281352 0.07270992]
[0.7864055  0.05810114 0.08279762 0.07269574]
[0.78642377 0.05809713 0.08279011 0.07268899]
[0.7864325  0.05809521 0.08278649 0.0726858 ]
[0.78643666 0.05809431 0.08278477 0.07268426]
[0.78643864 0.05809387 0.08278395 0.07268353]
[0.78643959 0.05809367 0.08278356 0.07268318]
[0.78644004 0.05809357 0.08278338 0.07268301]
[0.78644026 0.05809352 0.08278329 0.07268293]
[0.78644036 0.0580935  0.08278324 0.0726829 ]
[0.78644041 0.05809349 0.08278322 0.07268288]
[0.78644043 0.05809348 0.08278321 0.07268287]
[0.78644044 0.05809348 0.08278321 0.07268287]
[0.78644045 0.05809348 0.08278321 0.07268286]
```

3 总结与体会

经历了此次的学习，不得不说兴趣的主导性是非常强悍的。从一个月前开始对 PageRank 算法产生浓厚的兴趣开始，一直到现在自学了部分的随机过程和概率论，丝毫没有学习大学数学的压力，也许是所有的计算都可以通过程序来完成吧（偷笑）。最后，直到完成 PageRank 的全部学习和编程，我成长了许多。

每每就着清晨的黑夜开始进行程序设计的那一刻，我的内心都得到了极大的满足，似乎我就可以掌控指尖的世界一样。程序称成了我的数学伙伴，更确切点，它让我不用在费时费力的去学习什么矩阵如何计算，不用去学什么幂法和代数算法，更不用去考虑什么逆矩阵——就简简单单的几行代码和基础数学知识就能走进数学的世界。

还记得 1 个月前发过一次说说：



那时的我恰巧听完了“网络情报收集与分析”的 PageRank 算法课程，当时老师就比较“细致”的讲解了一下该算法。由于我既没学过线性代数，又没学过概率论，因此听矩阵相乘是懵逼的，听一阶马尔可夫链是懵逼的，听随机游走模型还是懵逼的.....心酸的往事在现在回想起来都有点好笑，明明“那么简单的知识”还好意思说“大受震撼”（微笑）——好吧，其实一切的原因都来自于知识上的不足。

知识是学不完的，唯有“学习能力”才是关键，最后，一句简单的话送给自己——“生活很忙，那就对了。”

参考文献

[1] 郑华,孙宇锋. PageRank 思想在若干数学课程教学中的渗透[J]. 教育教学论坛, 2020, 476(30): 294-295.

[2] 同济大学数学系编. 工程数学 线性代数 第 6 版[M]. 北京：高等教育出版社, 2014.06.

[3] 李航著. 统计学习方法 第 2 版[M]. 北京：清华大学出版社, 2019.05.

[4] 清华大学《统计学习方法》第 2 版课件

[5] 原理参考：https://blog.csdn.net/gamer_gyt/article/details/47443877

[6] 初始算法参考：https://blog.csdn.net/ten_sory/article/details/80927738

[7] PageRank 算法研究现状与展望
https://wenku.baidu.com/view/ebb4f694b9f67c1cfad6195f312b3169a451ea8d.html

[8] PageRank 算法详解：https://zhuanlan.zhihu.com/p/137561088