一. PageRank相关的定义

1. term spam

指页面中隐藏大量与网页内容无关的词语, 只是为了在各种排序中排名优先. 因此, 仅仅依赖网页中关键词的统计来为网页进行排序是容易被误导的.   
2. spam farm

指的是作弊者用很多的作弊网页指向作弊者自己的某个核心网页, 来提高作弊网页的入度. 因此, 只依赖入度来为网页也是排序不靠谱的.

3. PageRank算法

是Google创始人拉里·佩奇和谢尔盖·布林于1997年构建早期的搜索系统原型时提出的链接分析算法. 该算法大大改进了当时网页排序的可靠性.

在接下去的讨论中, PageRank代表了一个算法, 也可以理解成一个函数. 也就是说我们有一个网站A, 通过PageRank我们可以给出一个得分, 即score = PageRank(A) . 同时, PageRank函数给出的得分也常常被称为PageRank值.

二. PageRank算法

1. PageRank算法核心思想

一个网页的重要性和价值, 应该由其他网页对它的评价决定, 而不是由网页自身所含有的信息来决定.   
因此, 算法把一个结点指向另外一个结点的有向边视作是投票. 而且, 不是所有的投票都是等价值的, 来自那些得分较高的重要结点的投票价值更高.

PageRank算法是一个需要多轮迭代直到收敛的算法.

2. 基本的算法描述

1）在初始阶段：网页通过链接关系构建起Web图，每个页面设置相同的PageRank值，通过若干轮的计算，会得到每个页面所获得的最终PageRank值。随着每一轮的计算进行，网页当前的PageRank值会不断得到更新。

2）在一轮中更新页面PageRank得分的计算方法：在一轮更新页面PageRank得分的计算中，每个页面将其当前的PageRank值平均分配到本页面包含的出链上，这样每个链接即获得了相应的得分。而每个页面将所有指向本页面的入链所传入的得分求和，即可得到新的PageRank得分。当每个页面都获得了更新后的PageRank值，就完成了一轮PageRank计算。

3. 计算表达式

我们假设有如下一个网页链接关系, 用邻接链表的表示方式是

A: B, C, D

B: A, D

C: A

D: B, C

我们假设每个结点初始的得分值都是一样的, 因此A=B=C=D=0.25, 我们用一个列向量v = (1/4, 1/4, 1/4, 1/4)'来表示所有结点目前的PageRank值.

我们根据邻接链表的链接关系, 可以画出如下的一个转移矩阵M.

其中, 第一列代表A会把它的得分均分给B, C, D三个结点, 因此每个结点拥有0.33的权重. 第一行代表着A结点能够从B, C结点得到的得分权重是0.5和1, 也就是说B会把自己一半的得分投给A, 而C会把自己所有的得分都投给A.



在第一轮迭代之后, 生成的v\* = M • v, 其中v代表v = (1/4, 1/4, 1/4, 1/4)'初始的列向量, v\*代表第一轮结束以后所有结点的得分列向量, 并且将作为算法下一轮的输入.

4. 改进: 解决流量旋涡(终止点)问题

问题: 如果图中存在着一个没有出度, 但是有入度的结点. 那么由于它不断地吸收别的结点传入的得分, 而不把得分投给别人, 最终会使得整个图结构中, 最后其他结点的得分都趋于0 , 而这个结点趋近于1.  
还有一种情况是, 称作"采集器陷阱"的结构, 即有入度的一个子图没有出度. 进来的PageRank值都在这个子图内循环和积累, 导致这个子图如果被看成结点, 那么它会有接近1的重要性.

我们使用一个叫抽税法(taxation) 的技巧来改进PageRank.

v' = βMv + (1-β)e/n 

这里, β可以取0.2左右的数值. e代表的是和v相同维度数的单位向量.

内涵: 把1 - β的得分作为抽税, 平均分配给所有结点. 这样能够缓解流量旋涡的问题, 让其他结点能获得一定的得分.

5. 在搜索引擎中的实际使用

PageRank得出的重要性值是网页排序中的一个重要属性, 但是不是唯一的.   
首先, 网页至少要包含查询中一个keyword, 一般要能够排在前10的话, 必须包含所有的搜索词项. 同时, 如果关键词出现在网页的标题, head meta标签中, 那么也会提高排名.

三. 计算问题

1. 一般情形下PageRank算法开销分析

1)时间开销: 算法主要时间开销将会是每轮迭代中做Mv的矩阵乘法上, 这是O(V2)的时间开销(V代表图中结点个数, 下同), 再乘上算法需要迭代k轮完成收敛, 因此PageRank的时间开销是O(kV2). 不过, 一般来说, 这个收敛需要次数k会是在10~100之间的数值, 不会特别大.

2)空间开销: 算法最大的空间开销来自于存储整个M矩阵到内存中, 这是O(n2)的空间开销. 因此如果假设有10^6个结点的图, 需要的M矩阵大小是10^12, 按照int型4byte来存储, 这相当于4TB的内存开销, 这是任何单机都无法承受的空间开销.

2. 对空间开销的优化办法

由于M矩阵的空间开销过大, 必须考虑对其的优化存储. 已知大多数情况下, M矩阵十分稀疏, 那么我们可以使用邻接链表或者类似形式, 只存储非零元素的值.

比如, 在Python中, 可以通过构造字典数据类型来实现.

G = {1: [2, 3, 4], 2: [1, 4], 3: [1], 4: [2, 3]}

G[1] = [2, 3, 4]表示结点1和2, 3, 4是有一条有向边.

实际运算中, 为了提高运算速度, 我们会以邻接链表形式存储两个linkIn, linkOut两个图, 方便运算中的快速调用.

linkOut = {1: [2, 3, 4], 2: [1, 4], 3: [1], 4: [2, 3]}

linkIn = {1: [2,3], 2: [1,4], 3: [1,4], 4: [1, 2]}

那么这种情形下, 空间开销就是O(V+E)的, 因为邻接链表存储了所有的点和有向边. 在稀疏图中, O(V+E)往往远小于O(V^2).

3. 对时间开销的优化办法

为了实现从O(V^2)下降到O(V+E)的优化, 我们需要重新定义一般PageRank中的Mv矩阵乘法操作.

我们知道, Mv乘法实际上完成的目的是算出v\*列向量, 也就是每个结点新的PageRank值. 按照之前所述, 我们有如下观察:

Observation : 结点的新PageRank值 = Σ (来源结点的PageRank值 • 本结点所分享到的权重)

因此, 做常规矩阵乘法中遍历所有元素的做法是非常浪费的行为. 我们可以直接利用linkIn字典找到指向本结点的所有结点, 并用linkOut字典获取本结点所分享到的权重值.

v\* += 1/len(linkOut[fromNode]) \* v[fromNode]

说明: v\*是本结点新的PageRank值, 1/len(linkOut[fromNode])代表获取本结点所分享到的权重值, v[fromNode]获取来源结点的PageRank值.

由此, 我们的时间开销变成了O(V + E), 在稀疏图中, 这样的时间开销远比O(V^2)小.

备注: 项目代码https://github.com/imcheney/NetworkMining/blob/master/core/MyPageRank.py