图大作业，OS:Windows10，Python:3.7.4

张配天-2018202180

2020 年 6 月 28 日

目录

1 基本 PageRank 实现

1.1 读取文件

1. **联想 genism 中的 vocabulary，**也定义两个字典：node_to_index 以顶点为键，顶点对应的序号为值，用来快速存储和查找顶点对应的序号；index_to_node 以顶点的序号为键，顶点名称为值，用来在最后输出的时候查找对应的顶点字符串。
2. 用正则表达式处理输入的文件，利用分隔符将头顶点和尾顶点分开。
3. 顶点的序号从 0 开始递增，每一个顶点对应唯一的序号，对于一个输入的头顶点，在 node_to_index 中查找该头顶点，若有则记录其序号，若没有则在 node_to_index 中加入该顶点，保存其对应的序号，并相应的更新 index_to_node 字典，最后将序号加一。获取到头顶点和尾顶点的序号后，将两者打包为元组，存入 edge_list。
4. 读取文件函数 readFile 仅返回 edge_list 和 index_to_node 字典。

1.2 pagerank 迭代

1. 利用 scipy.sparse.lil_matrix 存储原始的邻接稀疏矩阵。
2. 利用 sum 函数对每行求和，计算各顶点出度，得一维出度向量 out。
3. 将出度向量中不为 0 的元素转换成它的倒数，据此利用 sparse.diags 构建对角矩阵，此对角矩阵**等价于出度对角矩阵求逆。**
4. 利用 numpy 的传播，初始化用户向量。
5. 创建一个初值全为 0 的同维分数向量，记录前序用户向量。
6. 在迭代中以节点分数向量和自身转置相乘的结果判断是否收敛。
7. 利用矩阵间乘法和公式 $p^{k+1} = \alpha * p^k \circledcirc L + (1-\alpha) \circledcirc \mathbf{1_s}$ 进行迭代计算，得到 p^{k+1}。
8. **将节点分数向量和全为 1 的矩阵相乘相乘转化为节点分数向量和所有元素均为节点分数向量各元素总和的同维向量求和。**
9. **处理悬挂点要求把悬挂顶即出度为 0 的顶和所有顶之间连边，由于大幅度修改邻接稀疏矩阵代价高昂，于是将此问题转化为将用户向量中出度为 0 的顶的对应位置的元素乘 $\frac{1}{n}$，之后加到 p^{k+1} 中，**说明如 图 *1*。因此：
   - 记录出度向量中等于 0 的元素，得到相应的布尔索引。
   - 利用该索引构建对应位置为 $\frac{1}{n}$，其余位置为 0 的调整向量，将其与节点分数向量 (p^k) 相乘，得到的积与 p^{k+1} 相加。
10. 最终得到的节点分数向量使用 numpy.argsort 进行排序，得到从大到小的索引列表，再根据 index_to_node 字典将其转化为对应顶点的名称。

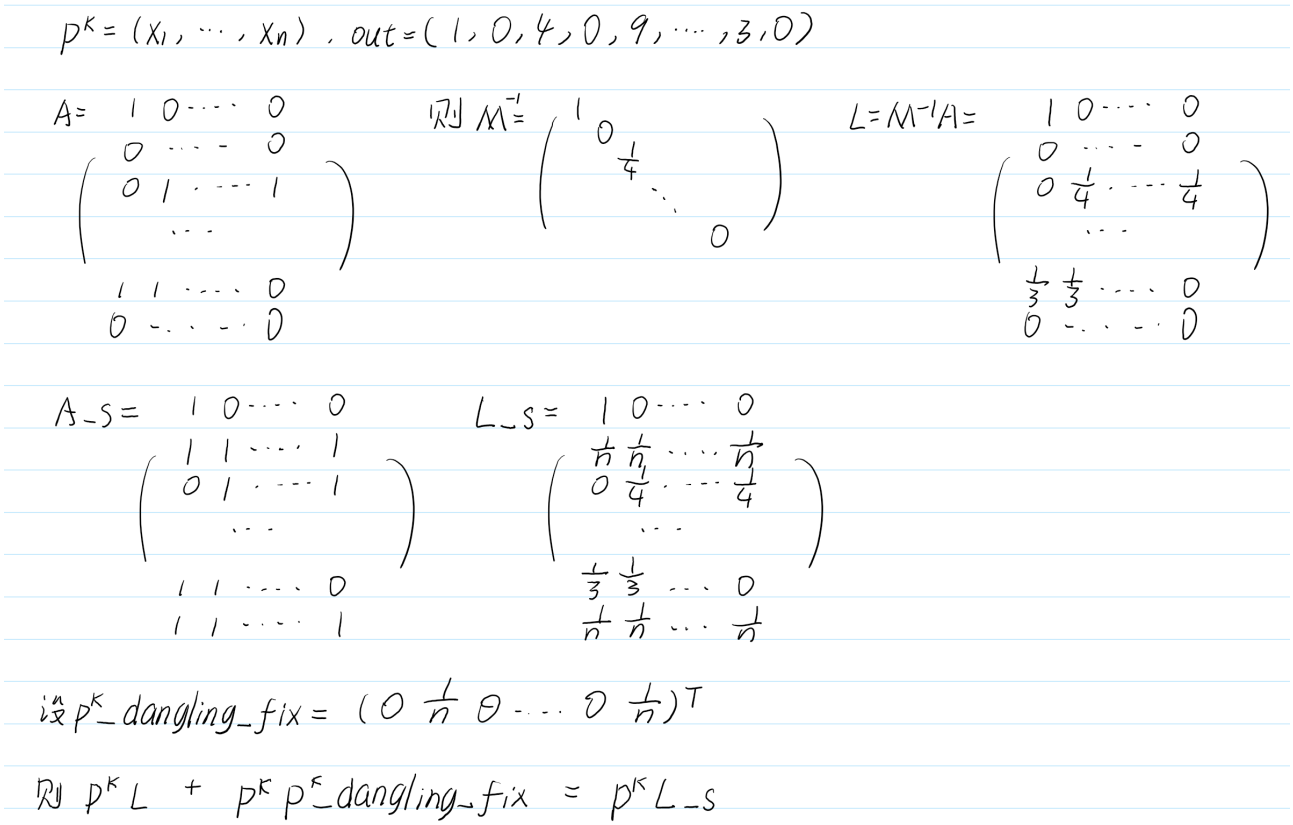

图 1: 算法说明

```
(base) PS C:\Pt_Python> & C:/apps/Anaconda/python.exe c:/Pt_Python/projects/PageRank/pagerank.py
time:3.7394771575927734
top10:['18', '737', '1719', '118', '790', '136', '143', '40', '1619', '4415']
score:[0.00399962 0.00263328 0.00187174 0.00184109 0.00180275 0.00177061
 0.0017256  0.00149914 0.00135318 0.00123522]
(base) PS C:\Pt_Python>
```

图 2: 运行结果

1.3 运行结果

程序运行 Epinions 数据集结果如 图 *2*，**耗时 3.7 秒**

2 Personalized PageRank 实现

2.1 实验思路

与 1 相似，在此不赘述。唯一不同是改变了节点分数向量的初值。

2.2 运行结果

程序运行 Epinions 数据集结果如 图 *3*，**耗时 3 秒**

```
(base) PS C:\Pt_Python> & C:/apps/Anaconda/python.exe c:/Pt_Python/projects/PageRank/pagerank.py
time:3.001681327819824
top10:['18', '31', '27', '34', '40', '30', '0', '12', '28', '29']
score:[0.00733736 0.00619428 0.00605359 0.00584595 0.00577025 0.0057507
 0.00566045 0.00544317 0.00510656 0.00486766]
(base) PS C:\Pt_Python>
```

图 3: 运行结果

3 关于存储大规模图数据的思考

图有多种存储方式：**邻接矩阵、邻接表、十字链表**等等，但在存储图数据时必须先考虑**数据规模和图上要进行的运算**。在 PageRank 的实现中，由于需要用邻接矩阵进行计算，且需要求各顶点出度，因此自然地想到使用邻接矩阵进行存储。

我首先尝试了使用 sparse 中 lil_matrix 进行存储，但由于稀疏矩阵的特性，不能批量地修改矩阵某一行的元素，因此我想自己实现一个数据结构 (代码中注释掉了)，使用字典链接邻接矩阵每一列不为 1 的元素，但在实现过程中突然灵光一闪，想到可以不修改矩阵中的元素，而用实验思路中的方法优化运算，最终达成目的。在加入 damping_factor 后的全 1 矩阵运算我也采用了相似的方法，其实是有些取巧的，但是在 pagerank 功能的范围内其效果蛮好。

因此，在存储图数据时，先要考虑到数据的规模，最好直接用 numpy 存储，毕竟其方法多、用着方便；但当内存受限时，必须考虑使用更巧妙的方法；且**必须考虑如何方便的进行后续的运算**，我自己想定义的数据结构的漏洞在于在此结构没有区分顶和顶之间的“**关联关系**”和“**值**”。

总之，sparse 还是很好用的类。

4 Personalized PageRank 线性可加证明题

PPR 公式如下：

$p = \alpha pL + (1-\alpha)p^0$

不妨设 $L = \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_n \end{bmatrix}$，其中，$l_i = \begin{bmatrix} l_{i1} & l_{i2} & \cdots & l_{in} \end{bmatrix}$

下面用数学归纳法证明 $p^k = \sum_i^n \lambda_i p_i^k$：

设 $p^m = \begin{bmatrix} \lambda_1^m & \lambda_2^m & \cdots & \lambda_n^m \end{bmatrix}$，$p_i^m = \begin{bmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{bmatrix}$

显然，$m = 1$ 时，满足 $p^1 = \sum_i^n \lambda_i p_i^1$，若 $m = k$ 时；有 $p^k = \sum_i^n \lambda_i p_i^k$ 成立。

则有：

$$\begin{bmatrix} \lambda_1^k & \lambda_2^k & \cdots & \lambda_n^k \end{bmatrix} = \sum_i^n \begin{bmatrix} \lambda_i a_{i1} & \lambda_i a_{i2} & \cdots & \lambda_i a_{in} \end{bmatrix} \tag{1}$$

那么 $m = k + 1$ 时：

$$p^{k+1} = \sum_{i}^{n} \alpha \lambda_i^k l_i + (1-\alpha)p^0$$

$$\Longrightarrow \sum_{i}^{n} \lambda_i p_i^{k+1} = \sum_{i}^{n} (\sum_{j}^{n} \alpha \lambda_i a_{ij} l_j + (1-\alpha)\lambda_i p_i^0)$$

$$\Longrightarrow \sum_{i}^{n}\sum_{j}^{n} \alpha \lambda_i a_{ij} l_j + \sum_{j}^{n} (1-\alpha)\lambda_i p_i^0 = \sum_{j}^{n}\sum_{i}^{n} \alpha \lambda_i a_{ij} l_j + \sum_{i}^{n} (1-\alpha)\lambda_i p_i^0 = \sum_{j}^{n}\sum_{i}^{n} \alpha \lambda_i a_{ij} l_j + (1-\alpha)\lambda_i p^0$$

由 (1)，上式

$$= \sum_{i}^{n} \alpha \lambda_i^k l_i + (1-\alpha)p^0$$

即 $p^{k+1} = \sum_i^n \lambda_i p_i^{k+1}$ 成立，$p^k = \sum_i^n \lambda_i p_i^k$ 得证。

综上，原题中，$p^* = \sum_i^n \lambda_i p_i^*$ 成立

5 实验小结

通过本次实验，我温习了 numpy 的各种知识，并学习了 sparse 中各种稀疏矩阵的用法，并且通过学会优化运算来达成目标。同时，自己在实现的过程中也遇到了这样那样的问题，但是最终都得以解决，我觉得 python 是一个很实用的工具，其中各种类都尤其独到之处，有空可以静下心研究研究其源码，这样才能知道究竟怎样才能最节省运行时间。

值得一提的是，我的 pagerank 代码的跑 Epinions 数据集的结果和 networkx 中跑出来的结果有一点点不同，我认为是收敛条件的原因。以上。