

**课 程 实 验 报 告**

**课程名称： 大数据分析**

**专业班级： ACM1901**

**学 号： U201914965**

**姓 名： 卫云泽**

**指导教师： 崔金华**

**报告日期： 2021年12月13日**

**计算机科学与技术学院**

**目录**

[实验二 PageRank算法及其实现 1](#_Toc58252332)

[**2.1实验目的** 1](#_Toc58252333)

[**2.2 实验内容** 1](#_Toc58252334)

[**2.3 实验过程** 1](#_Toc58252335)

[2.3.1 编程思路 1](#_Toc58252336)

[2.3.2 遇到的问题及解决方式 3](#_Toc58252337)

[2.3.3 实验测试与结果分析 3](#_Toc58252338)

[**2.4 实验总结** 4](#_Toc58252339)

# 实验二 PageRank算法及其实现

## **2.1实验目的**

1、学习pagerank算法并熟悉其推导过程；

2、实现pagerank算法[[1]](#footnote-1)；（可选进阶版）理解阻尼系数[[2]](#footnote-2)的作用；

3、将pagerank算法运用于实际，并对结果进行分析。

## **2.2 实验内容**

提供的数据集包含邮件内容（emails.csv），人名与id映射（persons.csv），别名信息（aliases.csv），emails文件中只考虑MetadataTo和MetadataFrom两列，分别表示收件人和寄件人姓名，但这些姓名包含许多别名，思考如何对邮件中人名进行统一并映射到唯一id？（提供预处理代码preprocess.py以供参考）。

完成这些后，即可由寄件人和收件人为节点构造有向图，不考虑重复边，编写pagerank算法的代码，根据每个节点的入度计算其pagerank值，迭代直到误差小于10-8

实验进阶版考虑加入teleport β，用以对概率转移矩阵进行修正，解决dead ends和spider trap的问题。

输出人名id及其对应的pagerank值。

## **2.3 实验过程**

### 2.3.1 编程思路

**数据的读入与预处理：**

实验所用数据集是经过预处理的数据，即发送方和接收方均使用id表示，但初步观察可发现数据集中id并不连续，故需要对读入的id重新编号，方可用于后续PageRank过程。数据的读入与id的重编号过程如下。

1. sr = pd.read\_csv("sent\_receive.csv")
2. # 统计出现的ID #
3. ids = set()
4. **for** id **in** sr['sent\_id']:
5. ids.add(id)
6. **for** id **in** sr['receive\_id']:
7. ids.add(id)
8. # 给ID再编号 #
9. id2num = {}
10. num = 0
11. **for** id **in** ids:
12. id2num[id] = num
13. num += 1

**初始化PageRank转移矩阵：**

基于预处理的数据，可构建状态转移矩阵，首先对存在收/发关系的用户id对进行标记（如使用1进行标记），全部标记完成后再根据各列标记总数计算出各个标记处对应的转移概率值，即保证了每一列的转移概率之和为1。使用标记方法的好处是同一组收/发关系只会被考虑一次，即自动实现了**重复边的过滤**，避免个别节点pagerank值过大的结果。PageRank状态转移矩阵的构建过程如下。

1. # 初始化矩阵M #
2. id\_sum = len(ids)
3. M = np.zeros([id\_sum, id\_sum])
4. edges = size(sr['sent\_id'])
5. **for** i **in** range(0, edges):
6. M[id2num[ sr['receive\_id'][i]], id2num[sr['sent\_id'][i] ]] = 1
7. **for** j **in** range(0, id\_sum):
8. sum\_j = sum(M[..., j])
9. **if** sum\_j == 0:
10. **continue**
11. **for** i **in** range(0, id\_sum):
12. M[i, j] /= sum\_j

**初始化rank向量并设置阻尼系数：**

在开始运行PageRank之前，需要初始化rank向量并设置阻尼系数。初始化的rank向量是包含项、每项值均为的列向量（其中是id总数），阻尼系数按照经验值设置为0.85。此部分代码实现较为简单，不再单独展示。

**进行PageRank迭代：**

基于公式r执行PageRank迭代过程，迭代需要设置出口条件，实验中设置为本次迭代结果相比上次rank值的最大差值不大于1e-8，则说明PageRank已达到收敛，可以停止迭代。迭代过程代码如下。

1. # 开始迭代 #
2. e = 10
3. **while** e > 1e-8:
4. p\_n\_next = beta \* np.dot(M, p\_n) + (1 - beta) \* p\_init
5. e = max(map(abs, p\_n\_next - p\_n))
6. p\_n = p\_n\_next

### 2.3.2 遇到的问题及解决方式

**PageRank结果之和不为1的问题：**

经测试，实验结果确实为不同id计算了响应的PageRank值，但观察发现用户的PageRank值普遍较小，尝试计算所有用户PageRank结果之和，发现求得的所有PageRank值之和小于1。图2-1展示了部分id的PageRank结果及所有用户的PageRank结果之和（Rank\_sum值）。

文本

低可信度描述已自动生成

图2-1 PageRank总和小于1的错误结果

**解决方式：**

经过仔细分析、回顾课堂内容，考虑到图中可能存在“黑洞节点”，即某些节点只有入度没有出度，这样的节点会“吸收”图中的PageRank总量，表现为结果的rank值之和不为1。一种解决策略是在迭代过程中执行归一化处理，即每一次产生新的Rank值向量后，对向量中的每一个rank值执行除以当前rank值之和的操作，即保证每次迭代结果的Rank值之和为1。加入归一化处理的迭代代码如下，其中代码第5行为归一化过程。

1. # 开始迭代 #
2. e = 10
3. **while** e > 1e-8:
4. p\_n\_next = beta \* np.dot(M, p\_n) + (1 - beta) \* p\_init
5. p\_n\_next /= sum(p\_n\_next) # 归一化
6. e = max(map(abs, p\_n\_next - p\_n))
7. p\_n = p\_n\_next

### 2.3.3 实验测试与结果分析

PageRank获得的结果最终输出到文件中，结果部分id对应Rank值如图2-2所示，图中冒号前的数字为用户id，最左侧的数字为行号。

文本

描述已自动生成

图2-2 PageRank程序运行结果（部分）

经验证，实验结果的所有Rank值之和为1，即归一化问题得到解决。输出结果能够正确反应各个用户的相对排名，具有一定的现实意义。

## **2.4 实验总结**

本实验基于寄件人-收件人数据集实现了PageRank算法，并加入了阻尼系数β，修正了概率转移矩阵，解决了dead ends和spider trap的问题。实验结果展示了不同id对应的pagerank值。

本次实验虽然较为简单，但也让我有了很大的收获。首先是亲自动手实现了PageRank过程，对理论化的知识有了实践上的认知；在实验中我也尝试再现了dead ends和spider trap等经典问题，领略了使用阻尼系数修正转移矩阵的精妙之处。此外，我还遇到了PageRank结果之和不为1的问题，最初认为是阻尼系数的加入对结果带来影响，后发现修正的转移矩阵同样具有归一化特性，rank结果不为1的问题是图中存在某些“黑洞节点”，采用迭代过程中的归一化操作可以解决此问题。

1. 基本pagerank公式r=Mr [↑](#footnote-ref-1)
2. 进阶版pagerank公式：r，其中为阻尼系数，常见值为0.85 [↑](#footnote-ref-2)