

**课 程 实 验 报 告**

**课程名称： 大数据分析实验**

**专业班级： 大数据2101班**

**学 号： U202115578**

**姓 名： 罗理恒**

**指导教师： 王蔚**

**报告日期： 2023年5月11日**

**计算机科学与技术学院**

**目录**

[实验一 wordCount算法及其实现 1](#_Toc59559190)

[1.1实验目的 1](#_Toc283926828)

[1.2 实验内容 1](#_Toc249534562)

[1.3 实验过程 1](#_Toc2039304590)

[1.3.1 系统整体框架 1](#_Toc753238010)

[1.3.2 编程思路与实现 2](#_Toc255135005)

[1.3.2实验测试 7](#_Toc1676669623)

[1.3.3 对比实验及结果分析 9](#_Toc505937827)

[1.4 实验总结 10](#_Toc1409299916)

[1.4.1 实验中遇到的问题及解决 10](#_Toc1506545449)

[1.4.2 实验总结 11](#_Toc1677163213)

[实验一 参考文献 11](#_Toc211770369)

[实验二 PageRank算法及其实现 12](#_Toc844188704)

[1.1实验目的 12](#_Toc2002576046)

[1.2 实验内容 12](#_Toc1931889338)

[1.3 实验过程 12](#_Toc1458844773)

[1.3.1 系统整体框架 12](#_Toc983302012)

[1.3.2 编程思路与实现 13](#_Toc1470252019)

[1.3.3 实验测试 16](#_Toc1578840951)

[1.3.3 对比实验及结果分析 17](#_Toc1271921125)

[1.4 实验总结 18](#_Toc1126125637)

[1.4.1 实验遇到的问题及解决 18](#_Toc1020200048)

[1.4.2 实验总结 19](#_Toc992769088)

[实验二 参考文献 19](#_Toc1669608473)

[实验三 关系挖掘实验 20](#_Toc2088274009)

[3.1 实验目的 20](#_Toc1296026342)

[3.2 实验内容 20](#_Toc388098473)

[3.3 实验过程 21](#_Toc863199772)

[3.3.1 系统整体框架 21](#_Toc1546532519)

[3.3.2 编程思路与实现 22](#_Toc1577467192)

[3.3.3 实验测试 27](#_Toc1805473729)

[3.3.4 对比实验与结果分析 29](#_Toc653031193)

[3.4 实验总结 29](#_Toc1853824581)

[3.4.1 实验中遇到的问题及解决 29](#_Toc1536982191)

[3.4.2 实验总结 30](#_Toc2126378021)

[实验三参考文献 30](#_Toc1760029220)

[实验四 kmeans算法及其实现 31](#_Toc1371346762)

[4.1实验目的 31](#_Toc1430529330)

[4.2 实验内容 31](#_Toc1827021145)

[4.3 实验过程 32](#_Toc2023199209)

[4.3.1 系统整体框架 32](#_Toc653039065)

[4.3.2 编程思路及实现 32](#_Toc1986129285)

[4.3.3 实验测试 35](#_Toc389084027)

[4.3.4 对比试验及结果分析 36](#_Toc247536674)

[4.4 实验总结 37](#_Toc673055679)

[4.3.2 实验中遇到的问题及解决方式 37](#_Toc1250428204)

[4.3.3 实验总结 38](#_Toc671855086)

[实验四 参考文献 38](#_Toc399414476)

# 实验一 wordCount算法及其实现

## **1.1实验目的**

1、理解map-reduce算法思想与流程，了解其在大规模数据处理中的优势和不足，在实验中模拟数据的并行处理。

2、应用map-reduce思想解决wordCount问题；

3、掌握并应用combine与shuffle过程，在原有map-reduce的基础上增加combine，shuffle过程以增加数据处理效率。

## **1.2 实验内容**

提供9个预处理过的源文件（source01-09）模拟9个分布式节点，每个源文件中包含一百万个由英文、数字和字符（不包括逗号）构成的单词，单词由逗号与换行符分割。

要求应用map-reduce思想，模拟9个map节点与3个reduce节点实现wordCount功能，输出对应的map文件和最终的reduce结果文件。由于源文件较大，要求使用多线程来模拟分布式节点。

学有余力的同学可以在map-reduce的基础上添加combine与shuffle过程，并可以计算线程运行时间来考察这些过程对算法整体的影响。

提示：实现shuffle过程时应保证每个reduce节点的工作量尽量相当，来减少整体运行时间。

## **1.3 实验过程**

### 1.3.1 系统整体框架

Map-Reduce框架是处理大规模数据任务中使用的编程模型和计算框架，其优势在于高并行性，高扩展性，低成本。[1]

本实验采用分布式处理方式，模拟多个计算节点，解决数词问题（WordCount）。从架构出发，本实验的map-reduce系统由Map，Combine，Shuffle，Reduce四个部分组成，其整体的运行逻辑如图1-1所示：

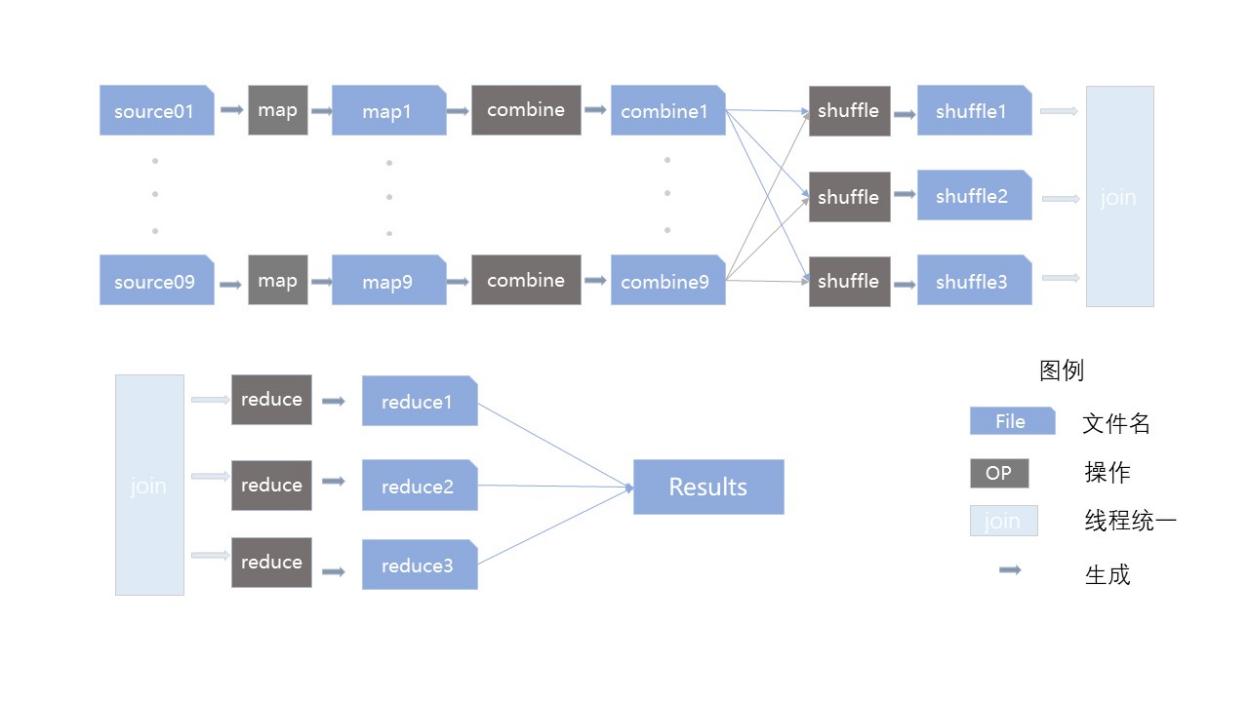
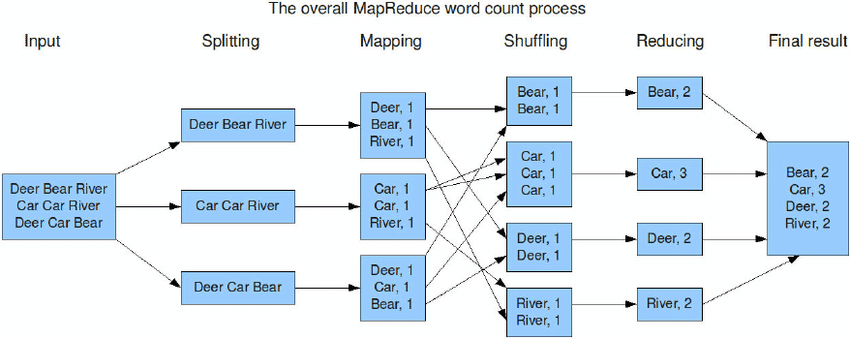


图1-1 Map-Reduce架构流程图

具体到WordCount问题上，图1-2展示了一个简单的分词示例。

图1-2 WordCount示例（图片来源于网络）[2]

### 1.3.2 编程思路与实现

**1）数据处理**

在进行map操作之前，通常需要对数据进行预处理。幸运的是，本实验给出的源文件（source0x）已经用逗号与换行符分隔，无需预处理，可以直接进行map操作。

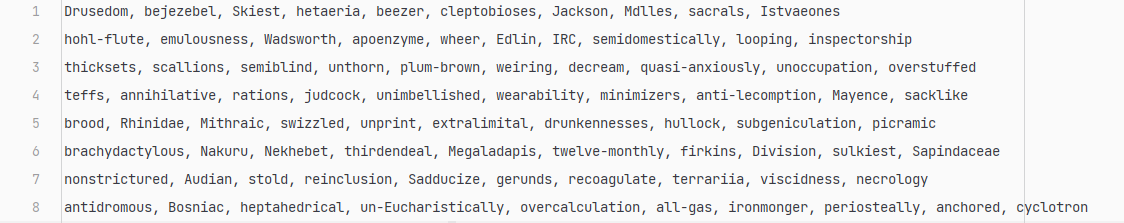


图1-3 source01文件格式

**2）Map**

map过程用于将源数据source0x文件中的信息提取出来，形成<word, count>的键值对。由于源文件中的数据较为规整，故此处的count恒为1。在实际操作中，我根据逗号进行分词，将每个词以<word,1>的格式写入了map文件。由于源文件有9个，我选择建立9个线程对数据进行并行处理，分别进行map操作。



图1-4 map过程代码展示



图1-5 创建线程并执行map操作

**3）Combine**

在Map-Reduce框架中，combine属于可选的中间阶段，负责对map过程中产生的结果（map-x文件）进行局部规约，在本文件内先对重复单词进行合并，这样可以显著减少数据传输量。

具体地，combine过程对map文件中处理好的<word, count>键值对进行统计，累加相同的关键字。在实现上，我设置了一个字典方便累加，遍历map文件中的每一行，若字典中已存在该词，则count累加；若未出现这个单词，则在字典中新增该词。在处理完成后，将字典按照键值的大小进行排序，这是为了方便后续的Shuffle和Reduce操作。最后，将结果写入对应combine文件中。由于map文件为map1-map9，相应地会生成combine1-combine9文件。由于建立线程的过程和map操作基本相似，此处不再赘述。



图1-6 Combine过程代码展示

**4）Shuffle**

在Map-Reduce框架中，shuffle主要负责“传递”中间键值对。由于reduce的节点较map（或map-combine）较少，shuffle操作按照一定的规则将map中的中间结果分区至更少的结果中（数量一般与reduce节点数量相同）。[3]

对于如何“分区”，一般的策略有两种。一是按照单词字母顺序继续划分，在本实验中可分为A-I，J-R，others三个类。另一种策略为使用hash模m的方法，将不同的单词哈希进3个shuffle文件中。在具体的实践中，我选择后一种方法，在shuffle过程中，相对于前一种方法，各个线程的负载更加均衡，耗时也更少。



图1-7 使用单词首字母顺序策略进行shuffle

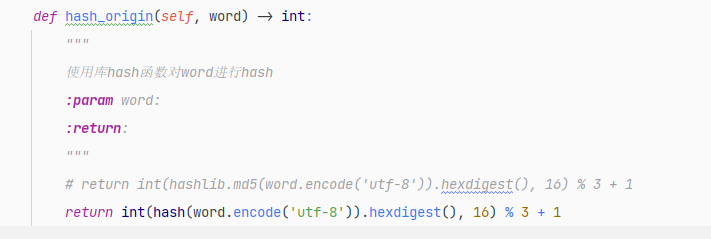


图1-8 使用哈希策略进行shuffle

具体地，shuffle过程对combine文件中的word进行分类，按照哈希结果写入shuffle文件。由于实验并未规定reduce节点的数量，所以我分别尝试了设置3个，4个shuffle节点（在实现上仅仅是mod3或者mod4的微小改动）并创建对应的线程（同map类似）。最后将9个combine文件转换成3个（或4个）shuffle文件（shuffle1-shuffle4）。



图1-9 Shuffle过程代码展示

5）**Reduce**

在Map-Reduce框架中，reduce将shuffle完成的文件再次合并，输出最终的结果。由于仅需要完成计数工作，无需单独设置Reduce函数。

具体地，reduce过程对shuffle文件内的键值对进行统计，其操作与前文的combine操作非常的类似。设置了一个字典，遍历map文件中的每一行，若字典中已存在该词，则count累加；若未出现这个单词，则在字典中新增该词。最后将结果键值对写入对应的reduce文件中即可。其创建线程的过程同combine类似，最后生成了3个reduce文件（reduce1-3）。



图1-10 Reduce过程代码展示

**6）结果生成**

最后生成的结果仅仅需要将3个reduce文件中的内容写入result.csv文件中即可，也可先读入一个字典，先排序后写入。

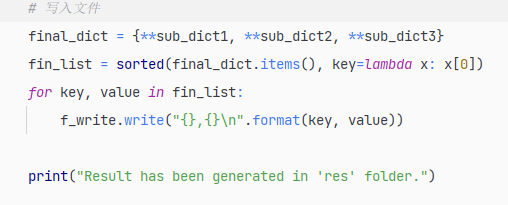


图1-11 生成结果代码展示

**7）并行处理数据**

Map-Reduce的一大优势就在于其并行性，事实上，在map，combine，shuffle操作的过程中，各个文件之间并不涉及交互，因此可以并行处理。在map操作时创建9个线程thread，第i个线程的执行顺序为：[4]

QianJianTec1684567729097

在9个线程全部完成后，再进行reduce操作。

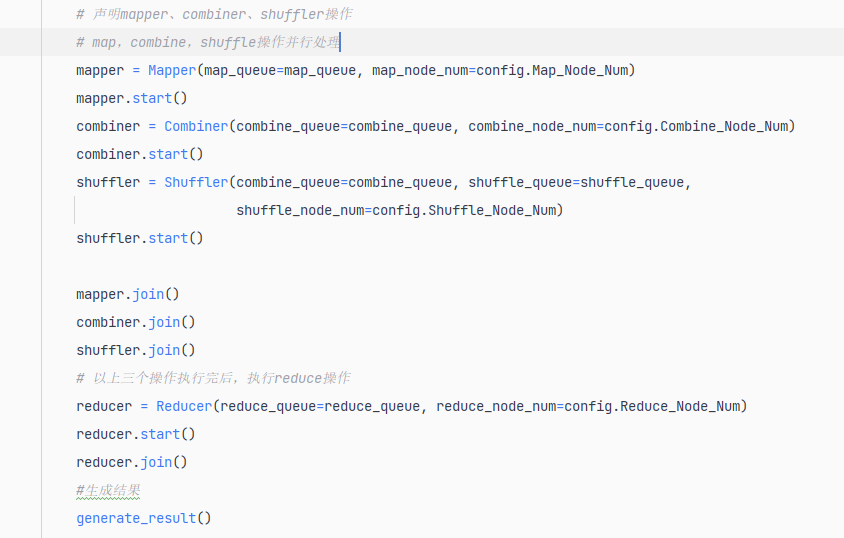


图1-12 并行处理数据代码展示

### 1.3.2实验测试

此部分主要展示Map-Reduce框架中各个环节运行的情况，以及最后的结果。

1. **实验环境**

本实验使用python进行开发，版本为3.11.2。使用的库为threading，os，time等。

实验的目录设置和基本参数如图1-13所示：



图1-13 实验基本参数设置

1. **Map测试**



图1-14 map文件截图展示

1. **Combine测试**

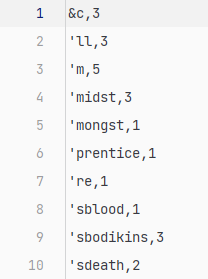


图1-15 combine文件截图展示

1. **Shuffle测试**

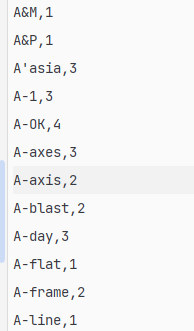


图1-16 shuffle文件截图展示

1. **Reduce测试**

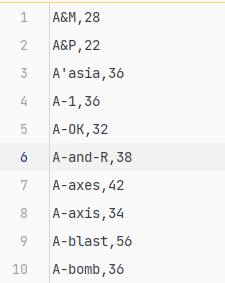


图1-17 shuffle文件截图展示

1. **Map-Reduce系统测试**

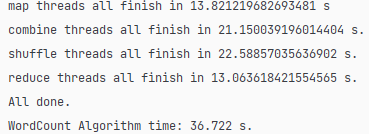


图1-18 总体运行时间展示

总体来说，系统运行正常，各个部分工作正常，生成结果正确且有序。

### 1.3.3 对比实验及结果分析

在设计程序中，我也在关注程序性能的优化问题，希望在保持正确性的基础上，最大程度的提升性能。我关注了以下几个问题：

1. 使用Combine操作是否能提升系统的运行效率？
2. 在Shuffle部分选择不同的分区策略会对该部分的运行时间产生什么影响？
3. 在Reduce过程中选择不同的计算节点数量对系统的性能产生什么影响？
4. 多线程并行能否提升系统运行的效率？

针对这四个问题，我做了四组对比实验，其整合的结果如图1-18所示。对比结果证明，这些操作都对Map-Reduce系统的性能产生了较大的影响。



图1-19 对比实验结果图

下面我将逐一分析这些结果的原因：

1. **Shuffle过程是Map-Reduce系统的性能的瓶颈**

由于shuffle过程需要完成map至reduce的中间数据传输，其中的数据映射，数据排序，甚至是网络传输，磁盘I/O都可能造成长时间耗时，成为map-reduce系统的性能瓶颈。因此，前三个问题均围绕着降低shuffle过程的耗时来提升系统的整体效率。

对于问题一，通过combine操作，能显著减少键值对的数量，方便shuffle的映射，同时combine操作还会对键值对进行排序，这极大的减轻了shuffle过程的工作负担。

对于问题二，使用Hash函数策略进行shuffle操作会使得每个线程的工作量大致一致（而且这个效果会随着数据集规模的增大表现得更加优秀），各个线程负载更加均衡；而首字母划分则可能导致一个线程工作量太大而其他线程“无事可做”，影响整个系统的效率。

对于问题三，reduce节点的数量应该适应数据集的规模和特征，本实验的数据集相对较小，过多的reduce节点并不一定意味着性能的提升。一方面，shuffle过程的映射可能造成负载不均，更重要的是，在最后生成结果时，节点更多意味着数据更加分散，整合的效率会降低。

1. **并行处理是map-reduce的重要优势**

map-combine-shuffle几个操作过程中，由于没有文件数据交互，因此具有非常强的并行性，这也带给程序极大的效能提升，这一点在处理更大规模数据集时会变得更加突出。这也告诉我在学习一个框架时，要发扬他的优势，以达到性能最优的目标。

## **1.4 实验总结**

### 1.4.1 实验中遇到的问题及解决

1）多线程编程

为了模拟map-reduce的分布式节点，我采用了多线程的方式，但是我从前没有接触过并行计算。为了完成实验任务，我首先阅读了《深入理解计算机系统》等资料，了解线程的概念，线程和进程的区别，然后学习了python中thread库的使用，包括join，start，lock等操作，实现了较好的模拟效果。

2）shuffle过程出现某一行只有键（单词），没有值

报错信息如图1-19所示，经排查，报错的原因是在程序运行出现combine文件中有3行只有单词，没有其计数值，我意识到这可能是在读取文件过程中出现了不合法的数据，导致字典无法处理。于是我在combine操作时加上了过滤操作，保证所有的数据均为合法，否则触发exception（代码如图1-20所示），成功解决问题。

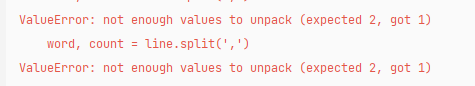


图1-20 shuffle过程报错截图

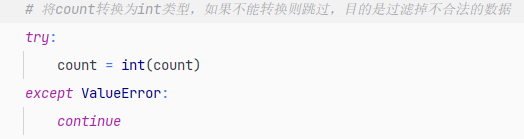


图1-21 解决报错代码

### 1.4.2 实验总结

本次实验让我第一次接触了Map-Reduce的大数据处理框架。在数据规模不断增加的今天，许多习以为常的方法都会发生变化，通过巨大规模数据的处理，我们可以获得更加精准，更加有价值的信息。通过实验，我熟悉了map-reduce的运行原理，挖掘了他的优势和不足，更是对多线程，并行计算，分布式节点等新兴概念有了初步的认识和应用，属实大开眼界。“纸上得来终觉浅，绝知此事要躬行。”我将继续努力。

# 实验一 参考文献

[1] "MapReduce." 维基百科，自由的百科全书, 19 May 2023, https://zh.wikipedia.org/wiki/MapReduce.

[2]https://www.researchgate.net/figure/Word-count-program-flow-executed-with-MapReduce-5\_fig6\_270448794

[3] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[4] Mining Massive Datasets — online class with free textbook

# 实验二 PageRank算法及其实现

## **1.1实验目的**

1、学习pagerank算法，了解其计算过程，并熟悉其推导过程；

2、实现pagerank算法，理解阻尼系数的作用，运用阻尼系数优化pagerank算法，减少算法本身的漏洞；

3、将pagerank算法运用于实际，实现pagerank算法以及加入teleport β后的修正过程，对结果进行对比分析。

## **1.2 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以及处理后的sent\_receive.csv数据以供参考）。

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。

## **1.3 实验过程**

### 1.3.1 系统整体框架

PageRank算法最开始是用来评估网页重要性的算法，是LinkAnylsis领域的重要算法。PageRank算法基于一个简单的假设，即一个页面的重要性取决于其他页面对它的引用。如果一个页面被其他页面引用得越多，那么它的权重就越高。同时，如果一个页面的权重很高，那么它引用的其他页面的权重也会相应地提高。 在基础的PageRank算法上，学者引入了teleport β，用来避免dead ends，spdier trap等问题。图2-1非常形象的展示了PageRank的基本假设。[1]图2-2则简单展示了在原有算法基础上加入teleport β机制的效果。[2]

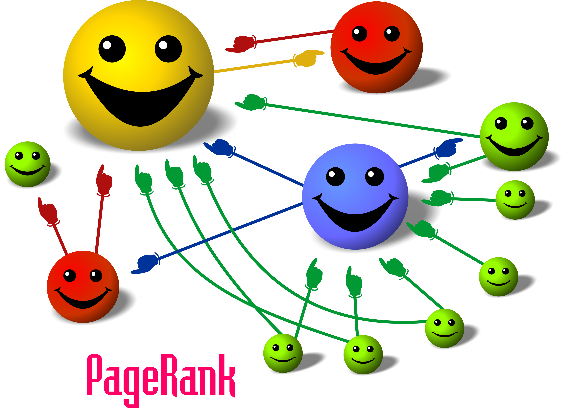


图2-1 PageRank算法概念图

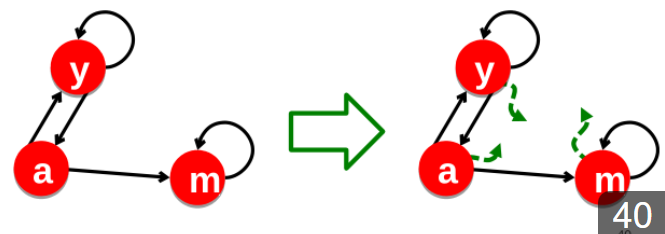


图2-2 PageRank算法加入teleport β机制示意图

### 1.3.2 编程思路与实现

**1）数据处理**

由于数据已经被预处理过，我们得到了开箱即用的数据集send\_receive.csv。（感谢学长姐）图2-3展示了数据集的部分数据。send\_id表示发送方，receive\_id表示接收方。整个数据处理过程分为两部分，第一部分为读入信息，统一人名；第二部分为构造有向图并写入文件（图2-4展示了此部分的代码）。通过这样的处理，能够极大的方便后续矩阵的构建。

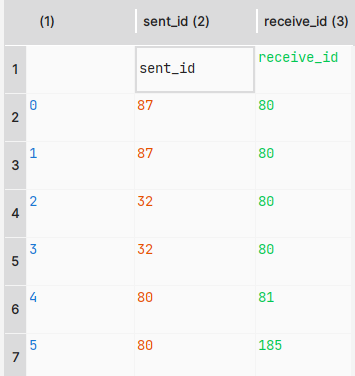
  
图2-3 send\_receive.csv数据截图



图2-4 数据预处理部分代码

**2）矩阵构建**

在构建矩阵时，根据人数，我们可以构建一个N×N的矩阵mat，其中N为总人数，mat[i][j]表示页面i是否链接到页面j，若不为0意味着界面i有链接到页面j，否则没有。之后对整个矩阵进行归一化处理，这样可以使得每个页面的PageRank值更加准确地反映其在整个网络中的重要性，避免了权重被平均分配的问题，从而提高了PageRank算法的准确性。



图2-5 矩阵构建代码展示

**3）PageRank算法及其迭代过程**

对于基本的PageRank算法，其迭代过程较为简单，数学表达式如图2-6所示。通过数学推导（此处省略具体过程），可以表示为两个矩阵相乘，一个矩阵为有向图的邻接矩阵，另一个为存储着PageRank值的pr矩阵。

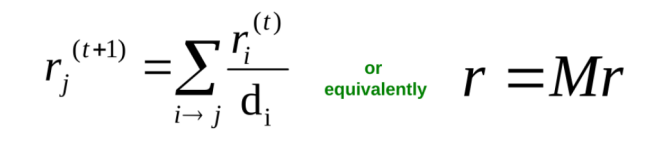


图2-7 pagerank算法计算过程

具体地，pr矩阵在初始化后进入迭代，通过和前一次计算误差err，然后更新pr矩阵并进行下一次迭代。其中err代表pagerank值的变化程度，err越小，代表pagerank值越来越趋于稳定。当误差err小于精度eps时，近似认为pagerank值可看作不变，迭代结束。



图2-7 pagerank算法迭代代码展示

**4）引入了teleport β的PageRank算法及其迭代过程**

传统的PageRank算法会遇到诸如deads end，spider trap等问题。图2-8是上述两问题的示意图，deads end意味着某一个节点没有链接到任何一个其他页面；spider trap意味着某几个节点互相链接，并且没有链接到任何一个外部页面。他们都会导致pagerank值的异常变化，获得全部权重。[3]

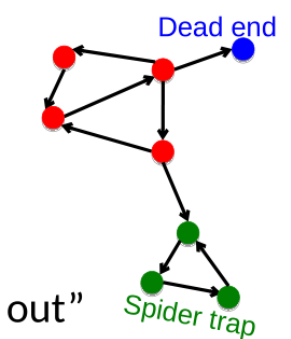


图2-8 dead ends和spider trap示意图

为了解决这个问题，我们在原有pagerank算法的基础上引入了随机游走（random walk）策略和teleport β，在一定概率下会随机跳转到另外的界面，这样能够很好的解决上述问题。我们将其用公式表示（图2-9所示），并将其化简（图2-10所示），使其易于实现，最终的公式，其

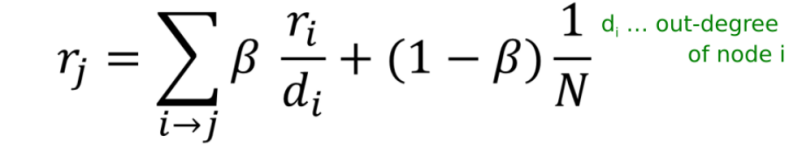


图2-9 引入teleport β的迭代公式

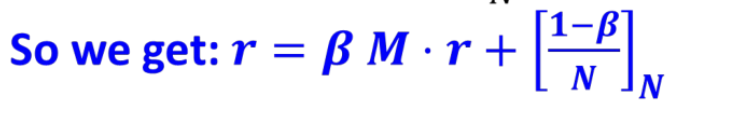


图2-10 化简后的迭代公式

相较于一般的pagerank算法，引入teleport β的迭代过程大致相同，仅仅是每次pagerank值迭代时用概率转移矩阵进行了修正。

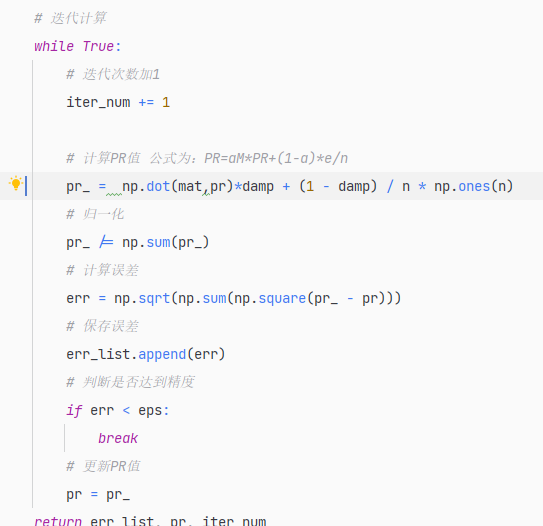


图2-11引入teleport β的pagerank算法代码展示

### 1.3.3 实验测试

**1）实验环境**

本实验使用python进行开发，版本为3.11.2。使用的库为numpy，pandas等。

实验的目录设置和基本参数如图2-12所示，默认精度（eps）为10-8，默认阻尼系数（damp）为0.85。[4]



图2-12 实验基本参数设置

**2）PageRank算法测试**

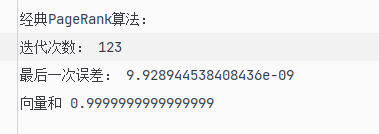


图2-13 pagerank算法运行结果截图

**3）引入teleport的PageRank算法测试**

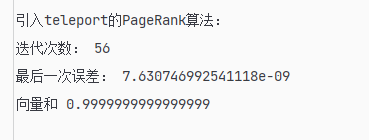


图2-14 引入teleport的pagerank算法运行结果截图

测试结果表明，pagerank算法运行正常，能够顺利收敛，同时在引入teleport β机制后，误差下降速度较快，整体收敛速度提升明显。

### 1.3.3 对比实验及结果分析

为了更加直观的展现误差err的下降过程，我绘制了err值随迭代次数变化的曲线图（图2-15）。从图中可以看出，err值在迭代次数处在[0,20]区间时下降较快，之后趋于平稳。这可能是因为记录pagerank值的pr矩阵在初始化时全部赋值为1/n，所以在最初的迭代中，pagerank值变动会较大。

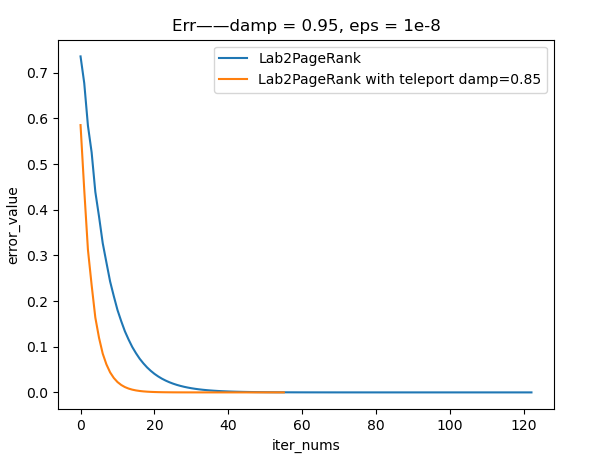


图2-15 pagerank收敛状态示意图

我注意到，teleport β机制的转移概率0.85是一个经验值，于是我探究了不同转移概率下pagerank次数的变化（图2-16）。实验中我分别将转移概率设置为0.70和0.95。从实验结果来看，随着转移概率β的提升，其更接近基础的pagerank算法，迭代次数也会增加；同时一味降低转移概率β，虽然在一定程度上降低了收敛次数，但这和本实验中pr值初始化有关，并不是一个普适的结论。事实上，将转移概率β设为0.7-0.85，pagerank都能达到不错的收敛效果。

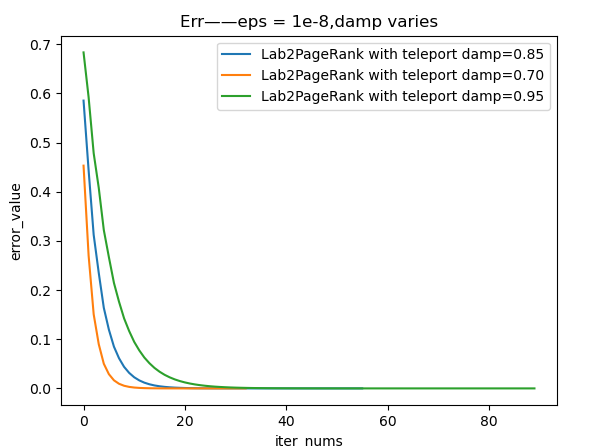


图2-16 不同的转移概率下pagerank收敛状态示意图

图2-17

## **1.4 实验总结**

### 1.4.1 实验遇到的问题及解决

**1）pr矩阵在迭代过程中未归一化**

在实验中，我获得了图2-17所展示的结果，很显然，途中的向量和（pagerank值）并不为1，经过排查，我发现在我的迭代过程中，没有对pr矩阵进行及时的归一化，导致pr值的计算越来越小，产生问题。解决方案即在每次迭代时增加归一化处理。



图2-17 错误输出截图



图2-18 解决方案

**2）矩阵读入错误**

最开始我的迭代次数高达181次，对于一个较小的数据集来说有些异常。在和同学的讨论和交流后，我发现在使用“send\_receive.csv”数据生成矩阵时，我没有考虑到有些页面没有任何的链接，直接使用person的数量作为矩阵的维度（这样算出来的维度为185\*185）。为了解决这个问题，我在代码中增加判断，得到最终的矩阵维度（182\*182）。

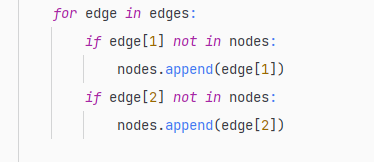


图2-19 判断某个页面是否完全没有外链接

### 1.4.2 实验总结

我一直记得老师上课时对于pagerank算法的评价。老师说这个算法正因为非常的简单易用（实现起来的确如此），同时能够保持较好的性能，从而得到了广泛的应用。这样“简单就是美”的设计理念让我陶醉。通过实验我也深入了解了pagerank算法的数学原理，迭代过程，亲手实现了一个小pagerank。

同时，我还关注到，pagerank算法并不完美，或许大部分算法也是这样，在应用中会产生更多的现实问题，这恰恰是一个算法优化改进的动力，通过引入teleport β机制，解决了dead ends，spider trap等问题，极大地提升了可用性和泛用性。我们在设计算法时，也要在应用中不断改进，不断优化，以达到最好的效果。

# 实验二 参考文献

1. "PageRank." 维基百科，自由的百科全书, 19 May 2023, <https://zh.wikipedia.org/zh-hant/PageRank.>
2. Sullivan, Danny. "What is google pagerank? A guide for searchers & webmasters." Search engine land (2007).
3. Nikolakopoulos, Athanasios N. "Random Surfing Revisited: Generalizing PageRank's Teleportation Model." arXiv preprint arXiv:2008.12916 (2020).

[4] Mining Massive Datasets — online class with free textbook

# 实验三 关系挖掘实验

## **3.1 实验目的**

1、理解Apriori算法思想与流程，了解其生成频繁项集的具体步骤。

2、应用Apriori算法解决关联规则提取问题；

3、掌握PCY算法，理解PCY算法相对于Apriori的改进思路和流程变化，验证PCY和Apriori算法的性能。

## **3.2 实验内容**

**必做：**

1. 实验内容

编程实现Apriori算法，要求使用给定的数据文件进行实验，获得频繁项集以及关联规则。

2. 实验要求

以Groceries.csv作为输入文件

输出1~3阶频繁项集与关联规则，各个频繁项的支持度，各个规则的置信度，各阶频繁项集的数量以及关联规则的总数

固定参数以方便检查，频繁项集的最小支持度为0.005，关联规则的最小置信度为0.5

**加分项：**

1. 实验内容

在Apriori算法的基础上，要求使用pcy或pcy的几种变式multiHash、multiStage等算法对二阶频繁项集的计算阶段进行优化。

2. 实验要求

以Groceries.csv作为输入文件

输出1~4阶频繁项集与关联规则，各个频繁项的支持度，各个规则的置信度，各阶频繁项集的数量以及关联规则的总数

输出pcy或pcy变式算法中的vector的值，以bit位的形式输出

参数不变，频繁项集的最小支持度为0.005，关联规则的最小置信度为0.5

## **3.3 实验过程**

### 3.3.1 系统整体框架

Apriori算法是一种常用的关联规则挖掘算法，用于发现数据集中的频繁项集及其关联规则。通过逐层筛选频繁项集，我们可以挖掘出最终的规则。Apriori算法的框架如图3-1所示（仅展示两个pass）[1]。

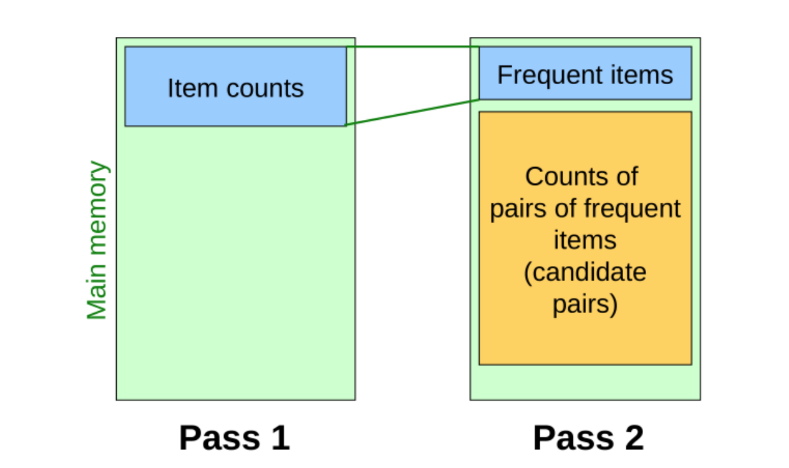


图3-1 Apriori算法框架示意图

在Apriori的基础上，PCY算法引入了哈希函数来降低内存消耗和提高性能。通过使用哈希函数将项集映射到内存中的一个存储桶（bucket）中，并使用bitmap标记频繁项桶的信息，能够减少生成频繁项集的时间消耗，快速得到频繁项候选集。PCY算法的框架如图3-2所示。[3]

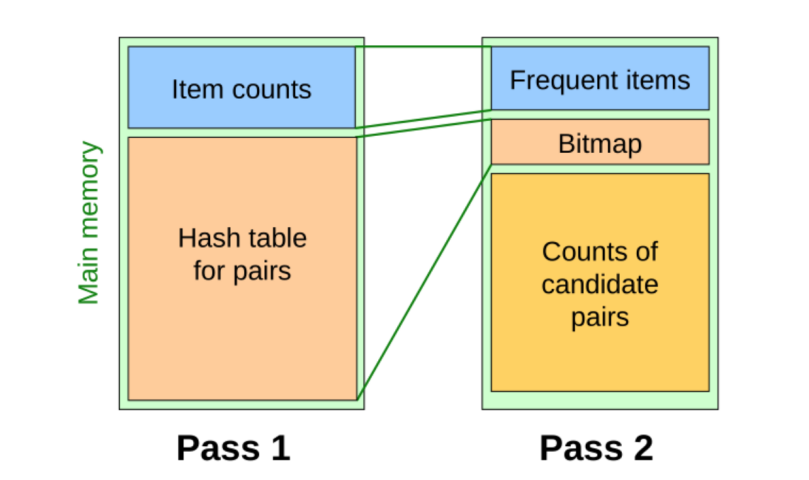


图3-2 PCY算法框架示意图

在PCY的实现上，可以采用multi-stage（多阶段），multi-hash（多哈希）等变式，进一步优化算法性能。multi-stage在每个pass中均采用了哈希映射的方式，multi-hash在单个pass中将pairs分割成多个hash table，分别进行映射。两种变式的框架如图3-3，3-4所示，由于其仅为PCY的变式，本次实验不做实现。

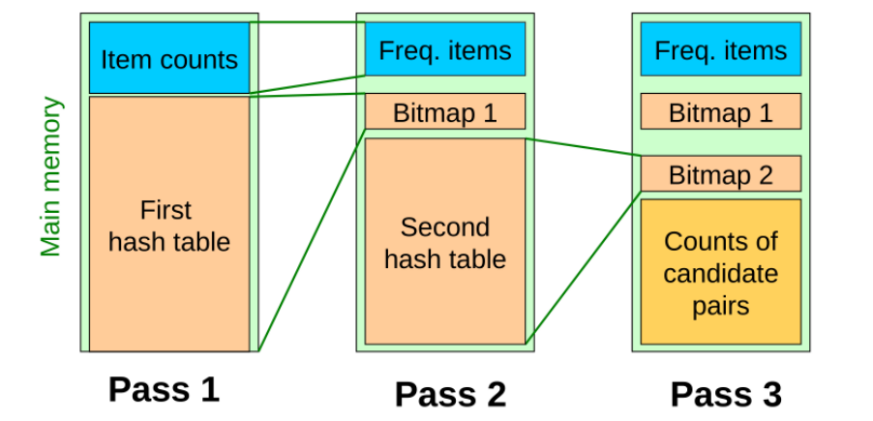


图3-3 multi-stage框架示意图

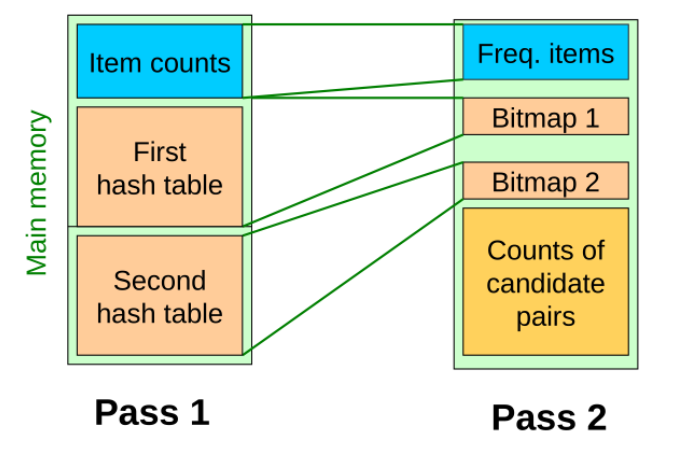


图3-4 multi-hash框架示意图

### 3.3.2 编程思路与实现

**1）数据处理**

原始数据集Groceries.csv中的数据格式如图3-5所示，每一行代表一个“购物车”（basket），第二列代表购物车中的物品集（items）。

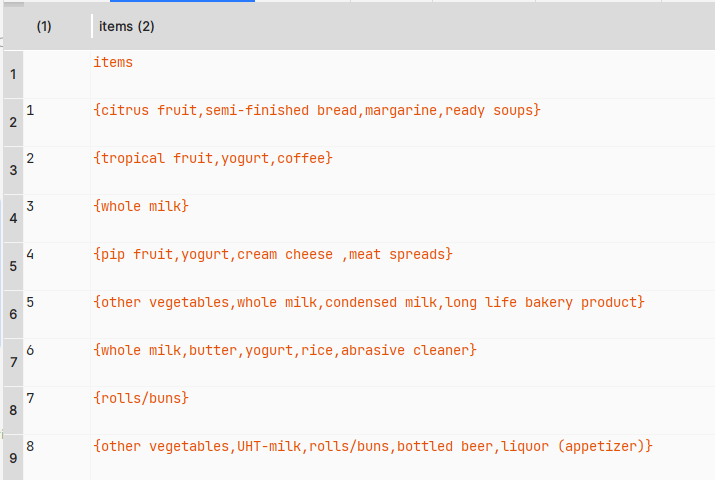


图3-5 原始数据集截图

然而，这样的数据无法直接读取，我们首先采用list[list]的格式将数据存储进dataset中方便进一步的操作。为了方便展示数据格式，我将其写入了txt问题件（代码如图3-6所示，结果如图3-7所示）。由于本实验中涉及到多个不同的数据结构，接下来会对其进行总结。



图3-6 数据处理代码截图

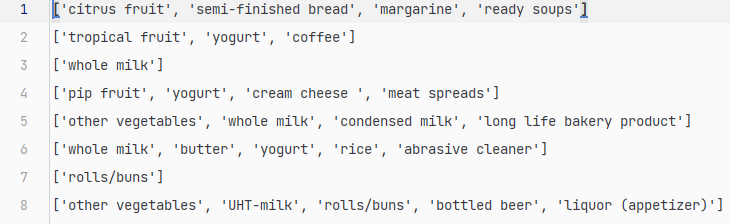


图3-7 dataset数据展示

**2）数据结构选择**

A）dataset

1．类型：list[list]二维列表，如[[item1,item2],[item3.item4],…….]

2．功能：转换原有数据集格式，简化后续读取操作。

B）频繁项候选集Ck

1．类型：set(frozenset)类型。

2．功能：存储第k次频繁项候选集。其中k代表每个frozenset中有k个item。事实上，frozenset（不可变集合）的选用是在查阅网上资料后得出的结论，大部分实现均采用了此种手段，其具体原因我将在3.4.1节进行阐述。

C）频繁项集Lk

1．类型：set集合类型，集合内的元素格式为frozenset。

2．功能：存储第k次频繁项集。其中k代表每个frozenset中有k个item。

D）频繁项集及其置信度frequent\_setk（conf）

1．类型：dict{frozenset,float}字典类型。

2．功能：存储第k次频繁项集及其置信度。

E）关联规则rule\_list

1．类型：list[set,set,float]列表类型。

2．功能：存储由频繁项集生成的关联规则。

F）hash\_table

1．类型：list列表类型。

2．功能：存储每个哈希桶中集合的数量。

G）bucket

1．类型：list[list]列表类型。

2．功能：存储每个哈希桶中集合的具体items。

F）bitmap

1．类型：list列表类型。

2．功能：判断每个哈希桶是否是频繁桶，如果其值为1，则为频繁桶。

**3）Apriori算法实现**

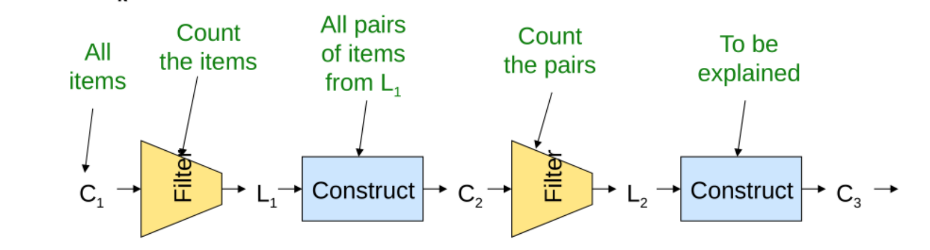


图3-8 Apriori算法流程图

Apriori算法实现的部分流程如图3-8所示，具体地可分为四个原子操作：生成初始频繁项候选集C1，生成下一个频繁项集Lk，生成下一个频繁项候选集Ck，生成关联规则rule\_list。下面我将具体阐述些操作。[2]

A）生成初始频繁项候选集C1

此部分仅需遍历dataset中所有的basket，将每个basket的所有item以frozenset的类型add进C1的集合中。代码如图3-9所示



图3-9 生成初始频繁候选集C1代码展示

B）生成下一个频繁项集Lk

由频繁项候选集生成频繁项集的过程是筛选候选集合的过程。首先遍历数据集中的每一个basket，再遍历候选集中的每一个k阶集合是否为其子集，若是则其频次增加。之后计算每个k阶集合的支持度（support），计算方式为：支持度 = 出现频次/总项集数，即出现在数据集中的交易中的频率。如果支持度大于最小支持度（通常为0.005），则视其为频繁项，加入频繁项集Lk中。此部分的代码如图3-10所示。



图3-10 生成下一个频繁项集Lk代码展示

C）生成下一个频繁项候选集Ck

在生成频繁项候选集的过程中，Apriori算法采用了剪枝策略（Pruning Strategy）和连接策略（Join Strategy）。剪枝策略基于一个先验性质，即如果一个项集是频繁的，那么它的所有子集也一定是频繁的。连接策略的思想是若两个长度为k-1的频繁项集，如果他们前k-2项均相同，则可以将他们连接形成一个长度为k的频繁项候选集。

具体地，在创建频繁项候选集时，我们应用两种策略。首先将两个频繁项集（每个集合为k-1项）取并集，若其长度为k则满足连接策略。判断子集是否是频繁项集，若不是则执行剪枝操作。通过应用这两种策略，能够减少搜索的集合范围，提升运行效率。



图3-11 生成下一个频繁项候选集Ck代码展示

D）生成关联规则rule\_list

由频繁项集生成关联规则的过程中，我首先对每个集合进行了置信度（conf）判断，将超过最小置信度的规则判定为有效规则并输出。在生成规则后，写入结果文件中。



图3-12 生成关联规则代码展示

**4）PCY算法实现**

在PCY算法实现中，可分为三个步骤：生成哈希表统计每个桶的数量，生成存储集合的桶，判断是否为频繁桶（bitmap），生成下一个频繁项候选集。下面我将详细阐述这些过程：

1. 生成哈希表（hash\_table）和桶（bucket）

由原始数据生成第一次频繁项候选集（C1）时，我们遍历每一个“购物车（basket）”中的所有物品对，将其哈希到某个桶里，相应地修改哈希表的值和桶的集合。

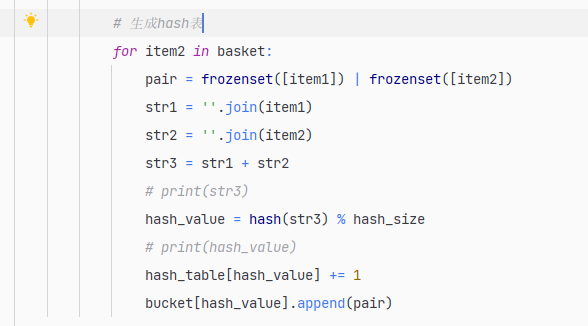


图3-13 生成哈希表和哈希桶代码展示

1. 判断是否是频繁桶

对于每一个桶，通过计算其支持度（即出现的频率），如果其超过最小支持度，就认为其为频繁桶，bitmap值置为1，反之则置0。



图3-14 判断频繁桶代码展示

1. 生成频繁项候选集

当我们完成了bitmap后，生成频繁项候选集会变得十分高效，只需将所有频繁桶中的集合加入候选集即可，其代码如图3-15所示。

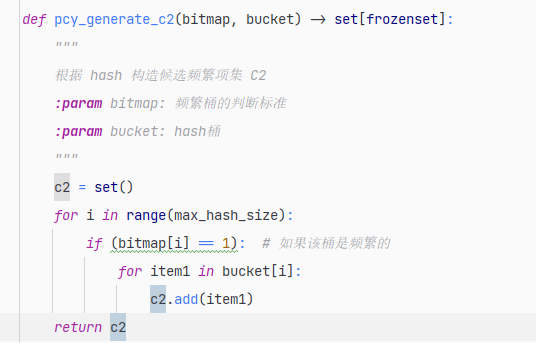


图3-15 生成频繁项候选集代码展示

### 3.3.3 实验测试

**1）实验环境**

本实验使用python进行开发，版本为3.11.2。使用的库为numpy，os，time，pandas等。

实验的目录设置和基本参数如图3-16所示：频繁项集最小支持度为0.005，关联规则最小置信度为0.5。



图3-16 实验基本参数配置

**2）Apriori算法测试**

首先测试该算法的正确性，生成各阶段频繁项集L1、L2、L3和关联规则。结果如图3-17所示。关联规则的部分截图如图3-18所示。



图3-17 Apriori算法运行结果

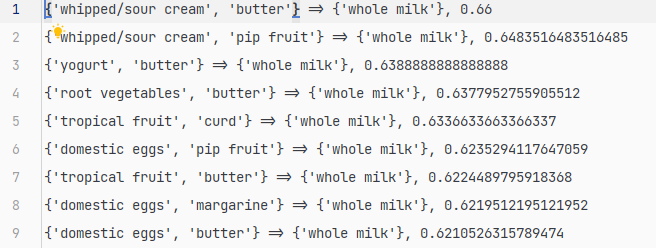


图3-18关联规则截图（降序排序）

3）PCY算法测试

经过测试，PCY算法得到的结果于Apriori算法相同，其部分bitmap值如图3-19所示。

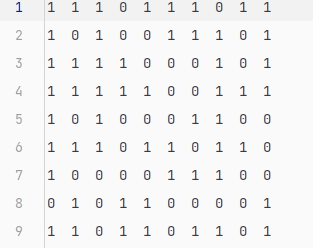


图3-19 部分bitmap值截图

### 3.3.4 对比实验与结果分析

本实验的对比主要分为两个方向，一个是Apriori算法和PCY算法在运行中的性能比较，第二个是PCY算法中不同参数的运行结果。

1. **哈希算法带来的性能改进**

在使用PCY算法之后，整个算法运行的时间有了明显的增加。为了更进一步的分析原因，我输出了每一个阶段所使用的时间。具体的结果如图3-20所示。从实验结果来看，PCY带来最大的优化就是在生成C2时减少了集合的数量，降低了搜索的范围，所以生成L2频繁集时可以大幅减少耗时。事实上，由C2生成L2这个过程是耗时最长的过程。不过，由于在生成C1和L1时加入了哈希操作，其耗时会对应增加。



图3-20 Apriori算法和PCY算法对比

1. **选择合适的哈希桶数量**

从理论上分析，如果选择更多的哈希桶，那么每个哈希桶中集合的数量会更少，更加均匀，由此生成的C2候选集数量会进一步减少，性能会相应地增加。但是我们还应注意到，更多的哈希桶会导致在判断频繁桶，遍历频繁桶的操作时，会有更多的耗时。但是就本实验来讲，增加哈希桶的数量的确促成了程序性能的提升，具体结果如图3-21所示。



图3-21 PCY算法中选用不同哈希桶数量运行结果

## **3.4 实验总结**

### 3.4.1 实验中遇到的问题及解决

**1）使用frozenset**

由于高阶候选集中的元素都是集合，一开始我使用简单的set当作其元素，但是发生了报错，问题是set类型为可变量，但是set中的元素是不可变的。在网上查阅资料后，我了解到了frozenset类型（不可变集合），既能满足集合的特性，又能够存储在集合或者字典的键之中。

1. 生成哈希表

生成哈希表需要遍历每一个basket的每一对集合，在一开始我没有完全理解这种操作，没有去掉相同item组成的pair，导致在生成L2时频繁项的数量不正确，也导致了程序耗时增加。在检索到问题后，我重新书写了遍历方式，得到了正确的结果。解决方法如图3-22所示（其实很简单，但是很有效）

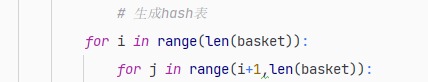


图3-22 解决问题的代码

### 3.4.2 实验总结

本次实验引入了关联规则，从生活中的具体现象抽象出一般的关系，从“购物车”行为中提取出商品相关的关系。这让我有两点感悟。一是，我们习以为常的平常背后都存在着充分地考虑。所以我们在生活中，不仅要对于异常的现象发问，更要对平常发问。而是，在计算机领域，学会抽象本质是我们进行科学研究非常需要的能力。以后我也会多锻炼这些能力，多实践，多思考。

# 实验三参考文献

1. Mining Massive Datasets — online class with free textbook
2. Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." Proc. 20th int. conf. very large data bases, VLDB . Vol. 1215. 1994.
3. Soo Park, Jong & Chen, Ming-syan & Yu, Philip. (1997). An Effective Hash-Based Algorithm for Mining Association Rules. Proceedings of the 1995 ACM SIGMOID international conference on Management of data ACM SIGMOID. 24. 10.1145/568271.223813.

# 实验四 kmeans算法及其实现

## **4.1实验目的**

1、加深对聚类算法的理解，进一步掌握聚类算法的算法原理和实现流程。

2、分析常用的聚类算法kmeans的操作流程,探究聚类算法原理。

3、掌握kmeans算法的代码实现，优化算法实现细节。

4、将kmeans算法运用于实际，解决现实中的无监督分类问题并掌握其评估指标和性能优化策略。

## **4.2 实验内容**

提供葡萄酒识别数据集，数据集已经被归一化。同学可以思考数据集为什么被归一化，如果没有被归一化，实验结果是怎么样的，以及为什么这样。

同时葡萄酒数据集中已经按照类别给出了1、2、3种葡萄酒数据，在cvs文件中的第一列标注了出来，大家可以将聚类好的数据与标的数据做对比。

编写kmeans算法，算法的输入是葡萄酒数据集，葡萄酒数据集一共13维数据，代表着葡萄酒的13维特征，请在欧式距离下对葡萄酒的所有数据进行聚类，聚类的数量K值为3。

在本次实验中，最终评价kmean算法的精准度有两种，第一是葡萄酒数据集已经给出的三个聚类，和自己运行的三个聚类做准确度判断。第二个是计算所有数据点到各自质心距离的平方和。请各位同学在实验中计算出这两个值。

实验进阶部分：在聚类之后，任选两个维度，以三种不同的颜色对自己聚类的结果进行标注，最终以二维平面中点图的形式来展示三个质心和所有的样本点。效果展示图可如下图所示。



## **4.3 实验过程**

### 4.3.1 系统整体框架

K-means是一种常见的无监督学习算法[1]，也是常见的聚类算法，用于将一组无标注的点按照距离分成多个类别或者簇别。特别的，其衡量距离的方式为常见的欧氏距离。K-means算法的主要步骤如图4-1所示：

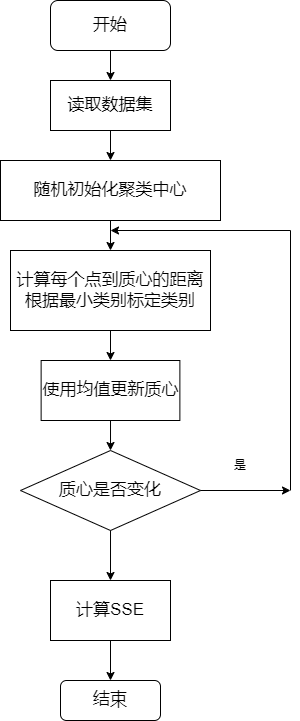


图4-1 K-means算法流程图

### 4.3.2 编程思路及实现

**1）数据处理**

原始数据（Wine.csv）需要先经过归一化处理，消除各个特征之间的量纲差异。不过贴心的学长姐为我们准备了归一化数据，可以直接使用，部分截图如图3-2所示。数据共有14列，第一类代表其类别，方便获得准确率，其余13列为13个特征。特别地，在读取特征数据时需要将第一行去掉。

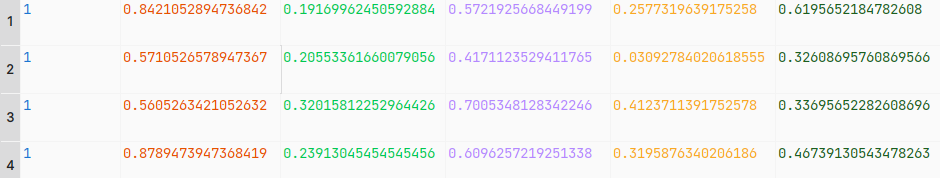


图4-2 归一化数据截图



图4-3 读取数据代码展示

**2）随机初始化质心**

初始化质心有两种方式，一是在[0,1]直接随机取随机数，二是在这一列坐标的最大值和最小值之间随机取值。我更偏向于第二种方法，在保证随机性的同时，也能够尽可能地靠近最终质心位置。



图4-4 随机初始化质心代码展示

**3）k-means迭代过程**

A）计算每个点的类别

每个点的类别根据其到三个质心的距离远近进行判断，分别对三个质心均进行欧氏距离计算，取最近的距离进行类别的标注。

我采用数组mat存储每个点的信息。第一列表示其到最近质心的距离，第二列表示k-means算法的分类结果，第三列表示其真实的分类结果。

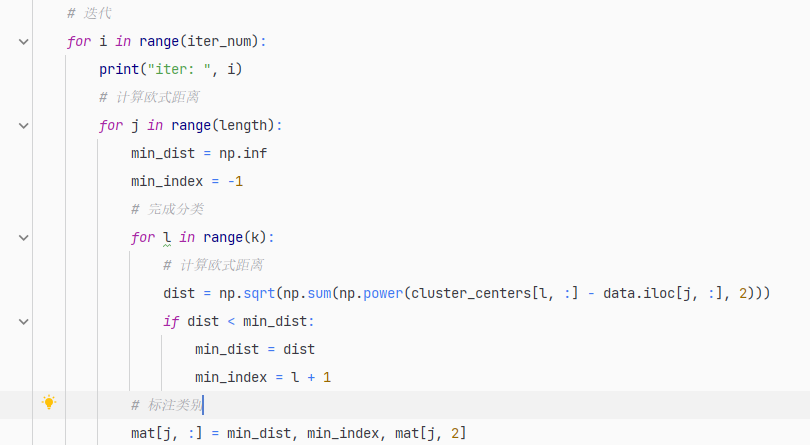


图4-5 计算距离标注类别代码展示

B）更新聚类中心

在标注完所有点的类别后，我们根据每个类别中的点，计算其均值，得到新的质心的坐标。当质心坐标不再变化（或变化值小于限制）或者超出预设迭代次数时，认为迭代完成，退出迭代。[2]

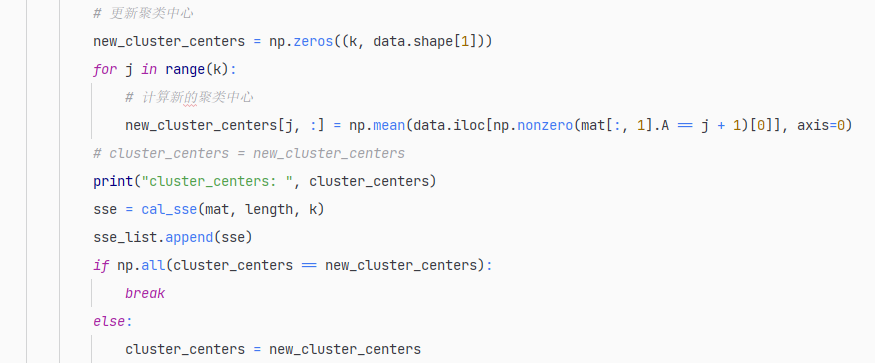


图4-6 更新聚类中心代码展示

**4）计算Acc（准确率）和SSE（残差平方和）**

在完成迭代后，我们将每个点的类别和其在数据集中的分类进行对比，得到分类的准确率。为了评估分类的效果，我们还计算SSE（Sum of Squared Errors），值越小代表分类的效果越好。SSE值的计算公式如图3-6所示，计算的代码如图3-7，3-8所示。

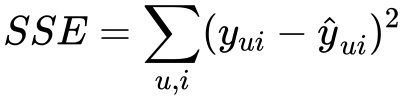


图3-6 SSE计算公式



图4-7 计算SSE代码展示

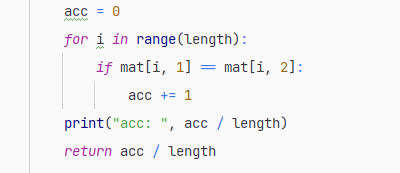


图4-8 计算准确率代码展示

**5）绘制分类图**

为了可视化分类结果，我采用matplotlib库进行绘制。由于源数据的特征为13维，此处任选两维（特征7，特征8）展现分类结果。图像结果将在4.3.3节进行展示。

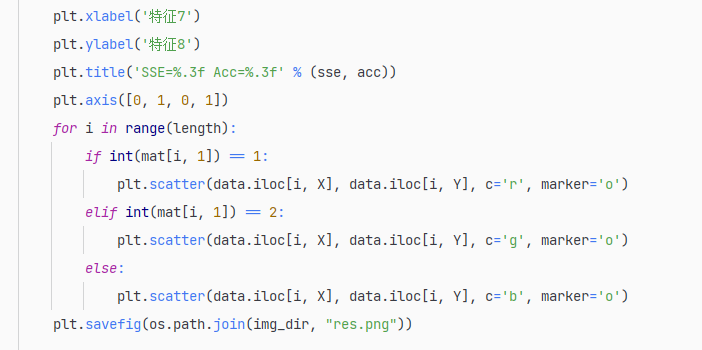


图4-9 绘图代码展示

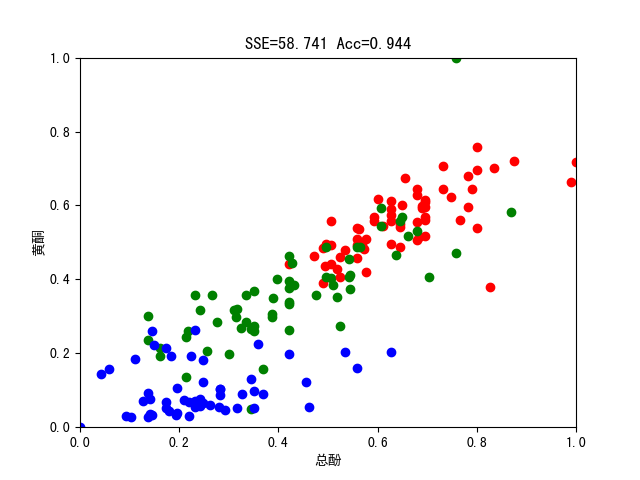
### 4.3.3 实验测试

**1）k-means算法结果测试**

我对k-means算法进行了多次测试，该数据集的迭代次数越6-10次，准确率约0.944，SSE值在[48,65]之间。实验结果总体符合预期。

**2）算法效果可视化展示**

图4-10，4-11展示了k-means分类的可视化效果，其中红色，蓝色，绿色分别为三个不同的类别，横纵坐标分别为两个特征（随机选取）。从效果可以看出，算法的分类结果较好，类别清晰，但是不同类别之间没有明显间距。这可能与绘图时特征选取有关。

  
图4-10 可视化展示

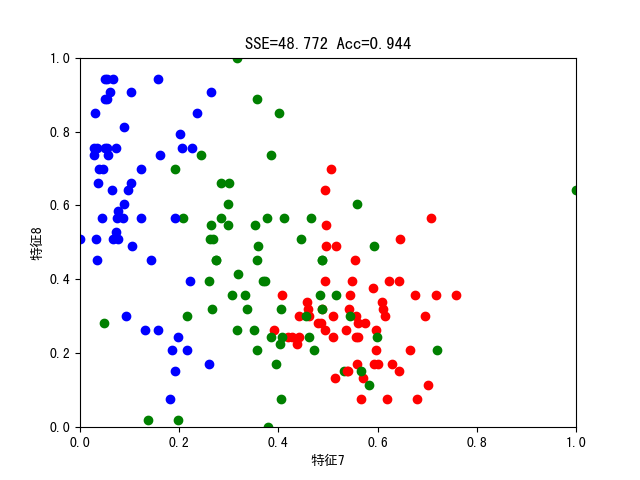


图4-11 可视化展示

### 4.3.4 对比试验及结果分析

**1）不同的质心初始化方式**

在4.3.2节中我提到过两种不同的质心初始化方案，我猜测不同的初始化可能对最后的分类效果影响较小，但是会影响到SSE值的变化。于是我分别绘制了两种不同方式SSE值的变化图。图4-12展示的是质心在[0,1]随机取值的SSE变化图，图4-13展示的是质心在该列最大最小值[min,max]之间随机取值的SSE变化图。

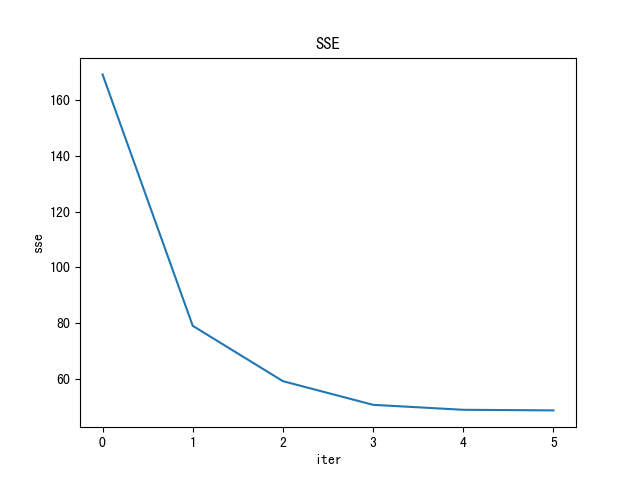


图4-12 在[0,1]间随机赋值

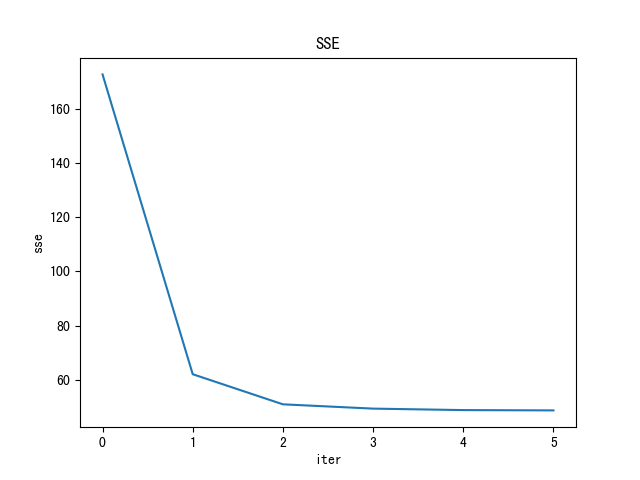


图4-13 在最大最小值间随机赋值

从图中可以得到，在[0,1]间随机赋值的SSE下降较慢，“肘部”也没有后一种方式那么明显，缩小随机取值空间确实能够使得质心更快接近最终答案。

## **4.4 实验总结**

### 4.3.2 实验中遇到的问题及解决方式

**1）准确率仅为2.2%**

在一次实验中，k-means算法正确率仅为2.2%，而同一代码多次实验的正确率可能发生浮动，34.5%，66.4%，94.4%均出现过，这样的问题引发了我的思考，但最终也找到了原因，问题并没有出现在代码上。

由于质心坐标的随机初始化，导致在分类时，每一个点分得的类别是随机的，而给定的类别是固定的。举个例子，比如A，B，C三个点，他们属于三个类别，k-means算法给他们标注的类别分别为0，1，2，但是在源数据中他们的类别为2，0，1。从结果来看，k-means算法的分类是正确的，但是在计算正确率是，他们和真实值并不匹配。这就造成了误判。所以发生了准确率的浮动，出现上述准确率均为正常，分别对应着一个，两个类别发生了误判。然而，即使发生误判，概率也会集中在0%，33%，66%，99%这几个概率上。

2）使用了14个特征

在一开始，我把其类别也当成了一个特征进行分类，这样就有14个特征，虽然从结果来看并没有非常明显的区别，甚至准确率更高（96%），但是明显不合理。我一开始并没有发现这个问题，直到我看到某一次实验中准确率达到了100%，这非常异常，经过检查才发现了多了一维特征。这也告诉我在使用数据集之前，必须先对数据集的结构进行分析。

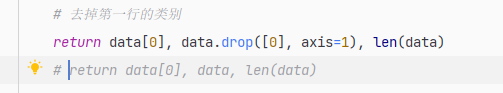


图4-14 使用正确的维度

### 4.3.3 实验总结

从实验的难度上来评判，k-means算法相对简单，流程清晰，运算简单，作为广泛使用的无监督聚类算法，k-means在几十年的发展过程中也有了许多改进和优化。我想起了一句名言，“其作始也简，其将毕也其必巨”，这是这样一个易于实现的算法，抛弃了奇淫技巧，纯粹而高效，得到了学术界和工业届的认可。这让我明白了我们需要的不仅是锦上添花，更是从头开始的原创性创新。

通过实现k-means算法，体会完整的迭代过程，我也将课堂上学习的内容同机器学习等知识进行了联系，这样课程直接的碰撞也帮助我从更多角度了解了k-means算法。

# 实验四 参考文献

1. Mining Massive Datasets — online class with free textbook
2. Alsabti, Khaled, Sanjay Ranka, and Vineet Singh. "An efficient k-means clustering algorithm." (1997).
3. "K-means clustering." Wikipedia, The Free Encyclopedia, 19 May 2023, https://en.wikipedia.org/wiki/K-means\_clustering.