

**课 程 实 验 报 告**

**课程名称： 大数据分析**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期：**

**计算机科学与技术学院**

**目录**

[实验一 wordCount算法及其实现 5](#_Toc134772970)

[1.1实验目的 5](#_Toc134772971)

[1.2 实验内容 5](#_Toc134772972)

[1.3 实验过程 5](#_Toc134772973)

[1.3.1 编程思路 5](#_Toc134772974)

[1.3.2 遇到的问题及解决方式 7](#_Toc134772975)

[1.3.3 实验测试与结果分析 9](#_Toc134772976)

[1.4 实验总结 11](#_Toc134772977)

[实验二 PageRank算法及其实现 13](#_Toc134772978)

[2.1实验目的 13](#_Toc134772979)

[2.2 实验内容 13](#_Toc134772980)

[2.3 实验过程 13](#_Toc134772981)

[2.3.1 编程思路 13](#_Toc134772982)

[2.3.2 遇到的问题及解决方式 14](#_Toc134772983)

[2.3.3 实验测试与结果分析 15](#_Toc134772984)

[2.4 实验总结 15](#_Toc134772985)

[实验三 关系挖掘实验 16](#_Toc134772986)

[3.1 实验内容 16](#_Toc134772987)

[3.2 实验过程 16](#_Toc134772988)

[3.2.1 编程思路 16](#_Toc134772989)

[3.2.2 遇到的问题及解决方式 20](#_Toc134772990)

[3.2.3 实验测试与结果分析 22](#_Toc134772991)

[3.3 实验总结 24](#_Toc134772992)

[实验四 kmeans算法及其实现 27](#_Toc134772993)

[4.1实验目的 27](#_Toc134772994)

[4.2 实验内容 27](#_Toc134772995)

[4.3 实验过程 28](#_Toc134772996)

[4.3.1 编程思路 28](#_Toc134772997)

[4.3.2 遇到的问题及解决方式 31](#_Toc134772998)

[4.3.3 实验测试与结果分析 32](#_Toc134772999)

[4.4 实验总结 34](#_Toc134773000)

[实验五 大作业推荐系统 35](#_Toc134773001)

[5.1实验目的 35](#_Toc134773002)

[5.2 实验内容 35](#_Toc134773003)

[5.3 实验过程 37](#_Toc134773004)

[5.3.1 编程思路 37](#_Toc134773005)

[5.3.2 遇到的问题及解决方式 44](#_Toc134773006)

[5.3.3 实验测试与结果分析 44](#_Toc134773007)

[5.4 实验总结 46](#_Toc134773008)

# 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；

2、应用map-reduce思想解决wordCount问题；

3、（可选）掌握并应用combine与shuffle过程。

## 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 编程思路

本实验是需要使用MapReduce的算法思想来完成WordCount的内容。MapReduce有这么几个算法步骤：

第一步是分割数据，也即map步骤，需要将大量的输入数据分割成一些小的块，这些小块可以并行处理。在WordCount中是需要将source01~09的文件中的单词拆分成单独的键值对。在Python中，我们可以使用with open来打开文件，并使用readlines方法逐行读取文件内容。然后可以使用字符串的split方法将每一行拆分成一个单词列表，并将这些单词存入一个数组中。最后，我们需要将这些单词按照键值对的格式写入对应的Map文件中，以便于Reduce过程的处理。再逐个按照键值对的格式写入对应的map文件。

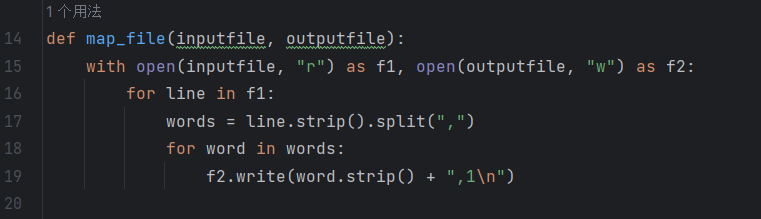
在map.py程序中，实现上述功能的代码如下：

图1.1读取文件

第二步是合并，也即combine步骤，Combine步骤通常用于一些可以在本地进行合并的场景，比如WordCount算法中对每个单词出现次数进行计数，就可以在Map阶段使用Combine步骤对每个Mapper的输出进行本地合并，减少Reduce阶段需要处理的数据量。在本实验中，我们对map处理后的文件，将里面含有相同键的单词进行统计次数，并将结果写入combine01~09的文件中。

在combine.py程序中，实现上述功能关键代码如下：

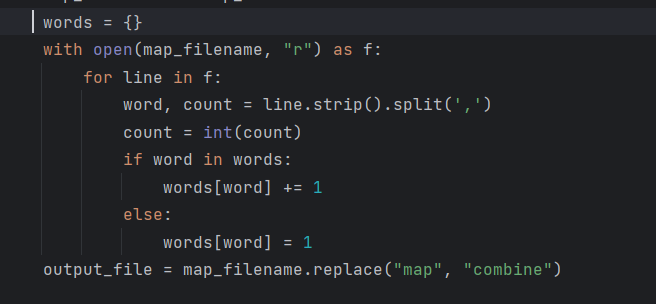


图1.2将本地先进行合并

第三步是分组，也即shuffle步骤，shuffle的作用是将combine阶段的中间结果按照键进行重新分配和排序，以便于Reduce阶段对数据的处理。在WordCount中是需要通过某种分配方式，让combine01~03的文件中单词尽量均匀的分配给后续处理的节点。在这里我们使用hash()方法来计算其哈希值，并且余3去判断应该分配到哪个节点。

在shuffle.py程序中，实现上述功能关键代码如下：

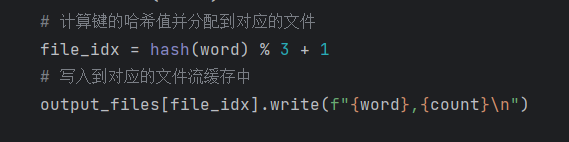


图1.3 shuffle步骤哈希分配结点

在上述程序处理完毕后，shuffle的文件大小基本平均，如下图：

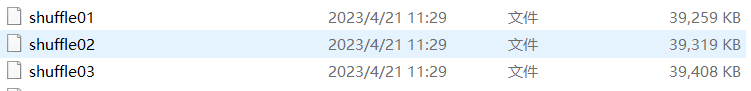


图1.4 分配后的文件大小

第四步是归约，也即reduce步骤，在reduce操作中，Reduce节点对每个键对应的值进行合并和计算，从而得到最终的输出结果。我们在这里需要将combine操作后的三个文件合并统计，采用多线程的方法调用python的threading库，在本实验中经过对比，采用ThreadPoolExecutor可以很大的减少程序需要花费的时间，帮助我们完成任务。

在reduce.py程序中，实现上述功能关键代码如下：

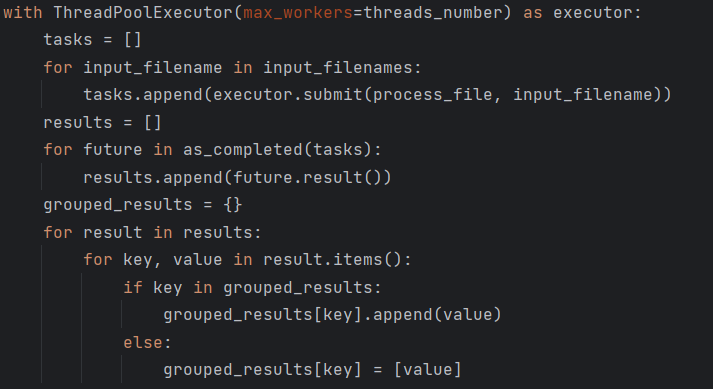


图1.5多线程过程

最后为方便比较算法消耗时间，在主体部分的两端加上time.time()来获取时间，相减得到运行时间。并且在main.py文件中，通过os库中的system方法让上述四个部分的程序按照顺序执行。

如下是main.py文件。

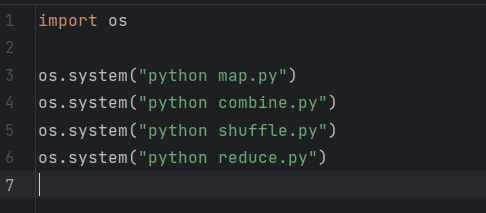


图1.6 main文件运行四个步骤

### 1.3.2 遇到的问题及解决方式

问题一，**如何将单词均匀分配给三个结点来合并统计，减少reduce处理的复杂度。**

一开始我是打算根据单词首字母直接将a-i、j-r、s-z这样直接分配给reduce的结点，但是这样处理没有考虑到部分首字母的单词比较多，最终还是可能会导致某个结点数据较大，延缓整个处理过程。所以我们需要一个更好的，更公平的方式来去分配这些工作。而python中的hash()函数可以很好的帮助我们。哈希函数将输入映射到一个固定大小的输出，该输出通常是一个整数，可以作为该python对象的唯一标识或者索引。对于本实验涉及到的单词，因为他们是不可变的对象，因此对于每一次程序运行时，他们的哈希值是固定不变的。通过对每个余三再加一的方式，分配好每个单词对应的处理结点。最后得到的三个文件大小十分相近，如下图所示：

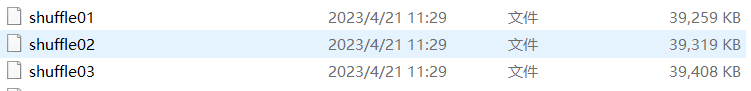


图1.7 shuffle操作后三个文件大小

问题二，**如何进行多线程方式来处理数据，而不是简单的for循环来进行处理。**

在第一次完成实验中，我采用的是for循环方式将每个source文件的单词先统计，最后直接将每个合并下来，这样是不符合实验要求，没有体现出MapPreduce的算法思想。着重在于，要实现模拟9个map节点与3个reduce节点实现wordCount功能，采用多线程方式来处理数据。这里我使用了python中的ThreadPoolExecutor库以及threading库，前者可以用来执行一组函数或方法，每个函数或方法都可以在一个独立的线程中执行。当有多个任务需要执行时，可以将这些任务提交给 ThreadPoolExecutor，它会自动将它们分配到可用的线程中执行，直到所有任务都完成为止，并且可以十分方便的控制其中的参数，例如线程池大小，线程池中线程最大数量，任务队列大小等。后者通过threading.Thread可以让我们实现线程，也即独立的指令序列，可以与程序中的其他线程并发运行。

相关程序如下图：

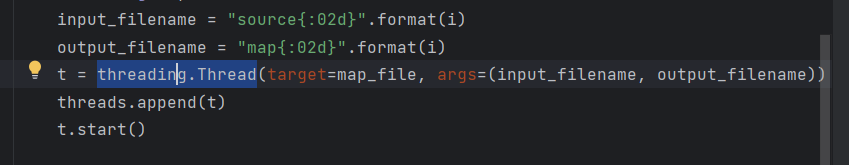
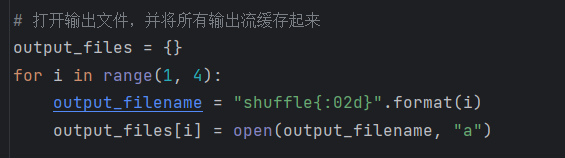


图1.8实现多线程操作

问题三，**在shuffle处理中，文件生成过慢，大概每个文件生成速度为6KB每秒，初版代码完成处理时间为10分钟，处理缓慢的原因是什么，该如何解决？**

原因是在shuffle处理中，需要同时写入大量数据，频繁地打开和关闭文件会降低程序效率。可以将所有的文件操作放到一个上下文管理器中，同时将每个文件的输出流缓存起来，统一处理写入操作。而如果我们采用使用缓冲区，可以减少写入操作的次数，从而提高程序的效率。



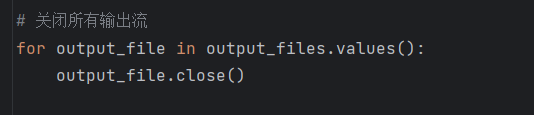


图1.9 将所有文件缓存统一操作

改进后shuffle处理步骤时间大幅度减少，从原来的十几分钟降至5秒内，如下图所示：



图1.10 改进后运行时间大幅度减少

### 1.3.3 实验测试与结果分析

执行main.py文件，得出各部分的文件处理结果，以及处理时间。

各部分处理时间，如下图所示：

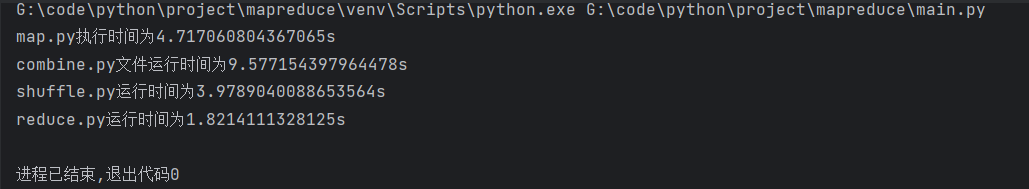


图1.11 各部分处理时间

map.py处理后，共9个文件，如下图所示：

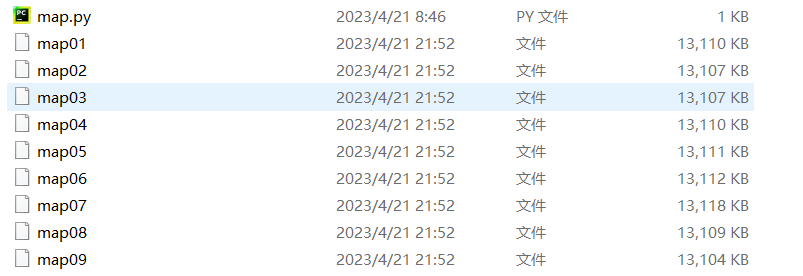


图1.12 map处理后文件

combine.py处理后，共九个文件，如下图所示：

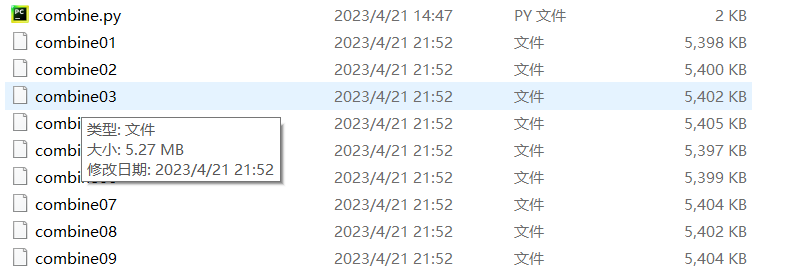
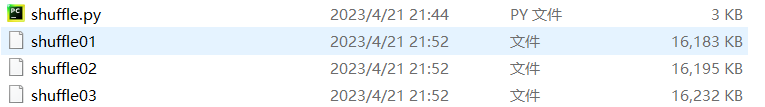


图1.13 combine处理后文件

shuffle.py处理后，共三个文件，如下图所示：

图1.14 shuffle处理后文件

最后经过reduce.py处理后，汇总到reduceResult文件中，确认程序运行成功，结果正确。以下是本次实验流程图：

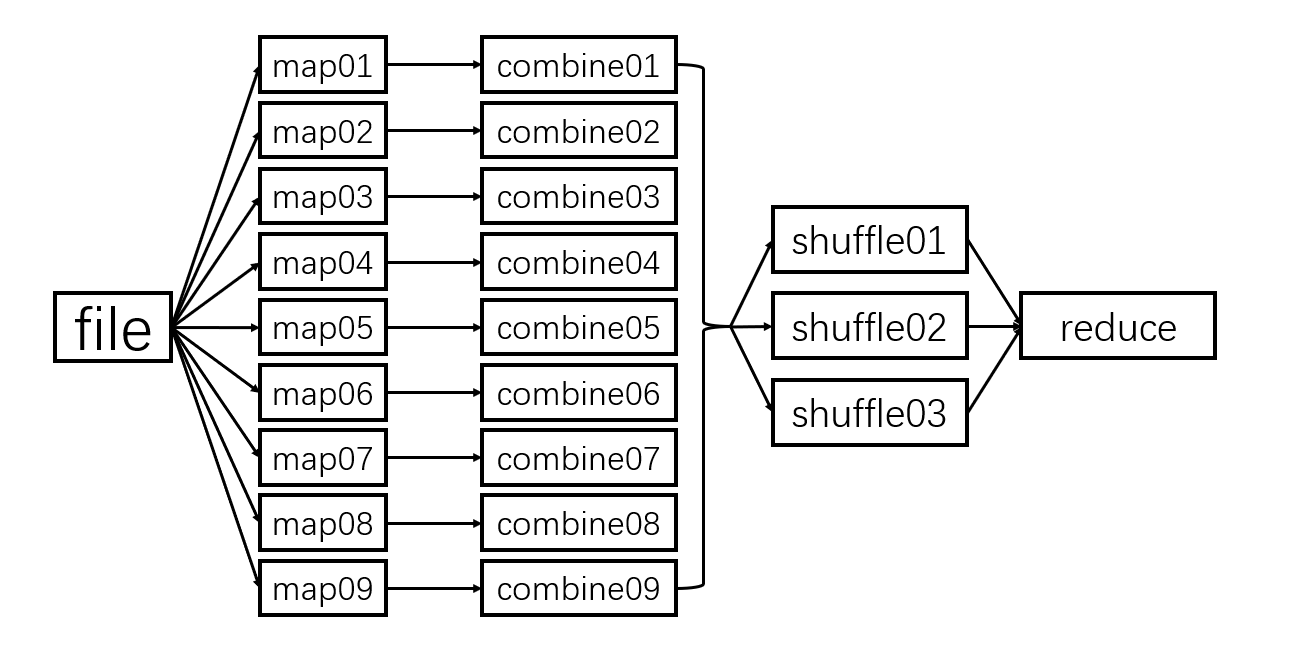


图1.15 实验流程图

在与其他同学的程序运行结果比较过程中，以及助教指出尚可改进之处，我对我的代码结果有了如下分析：

1. 整体大致实现了MapReduce思想来完成WordCount的功能，但是在combine和shuffle步骤上太过繁琐，可以将map、combine、shuffle三个步骤合成一个程序进行分布式处理，这样可以减少每一步的处理时间。
2. 多线程部分处理不够仔细，多线程同时写入文件可能会引起资源竞争和死锁问题。对于这个漏洞，我在上网查询后知道了可以使用线程锁来保护共享资源，避免多个线程同时访问和修改同一个文件。在写入文件时，每个线程需要获取锁来确保写入操作的原子性和可靠性。
3. 对于原文件处理不够完善，由于原文件内容较多、文件较大，可以使用python中的mmap模块，直接将文件映射到内存中，以便更加高效地对文件进行读取写入的相关操作。
4. 多线程分布任务不够完善，部分文件采用for循环来将任务文件分给线程处理，可能会导致程序运行缓慢，应该多查询相关库函数的用法，体会MapReduce算法中的分布式处理和合并这关键处理步骤。

## 1.4 实验总结

本次实验通过WordCount这一个简单但又不简单的内容，让我对MapReduce有了更深的理解。对于后者算法来说，其核心是在于将大量的数据分割成几个部分，并且将这些部分分配个多个结点来处理，最后将处理的结果进行合并得到最后的结果。

我们首先要实现的就是分布式这一个工作，在本次实验中化简成多线程处理。我使用了Python中的`threading`模块，创建了9个线程，每个线程读取一个文件块，处理其中的单词，并将结果输出到一个中间文件中。

我们不仅要实现分配或者说多线程处理，更重要的是将任务尽量均匀分配给结点，让每个结点都尽可能高效率的进行任务处理。而在本次实验中，我采用了hash方法来实现均匀分配任务。

接着上一段来说，得到中间文件后，但是还没有进行最终的统计。因此我们需要进行reduce操作，即将相同单词出现的次数进行合并，并统计出每个单词的总出现次数。在本次实验中，我使用了三个reduce节点，每个reduce节点处理其中的一部分，将结果合并成最终的输出文件。在这一过程中我们使用了ThreadPoolExecutor的方式来将任务加入线程池，实现reduce归并的工作。

最后关于MapReduce的一点感想，作为一种分布式计算框架，被用于大规模数据集的并行处理和生成计算结果。极大的方便了对当今互联网产生的大量数据进行处理，是一种十分高效、可信的处理方法。

# 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，理解阻尼系数的作用；

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。

## 2.3 实验过程

### 2.3.1 编程思路

根据相关的定义，相关的迭代式如图所示：

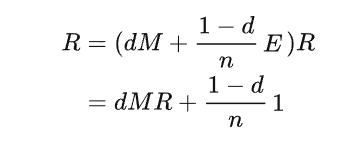


图2.1迭代公式图

其中d为阻尼因子，也即实验中的alpha，其值为0.85，M为原有向图构建的邻接矩阵经过归一化处理后得到的矩阵。对于本实验，我们无需担心迭代次数问题，我们只需要关注迭代前和迭代后两次的R矩阵的范数之差是否在我们所给范围内，如果是则可以输出对应的id以及迭代算出的pagerank值。

首先我们需要根据sent\_receive.csv文件，构建对应的有向图，python中提供了networkx库方便处理复杂的图网络结构。我们只需要将csv文件中第二列以及第三列id，当作有向边的两个点添加到提前定义好的有向图graph之中即可。

相关程序部分如下所示：

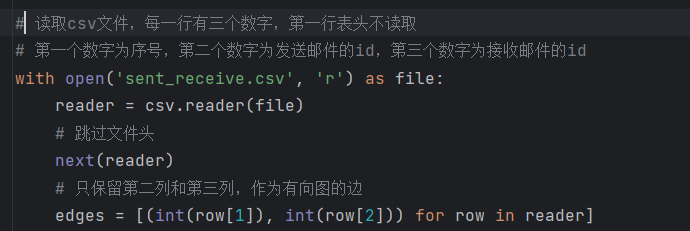


图2.2读取文件转化为有向图

然后我们需要根据这个有向图，构建对应的邻接矩阵，然后进行归一化处理，R的初始矩阵处理也和下面相似，相关代码如下所示：

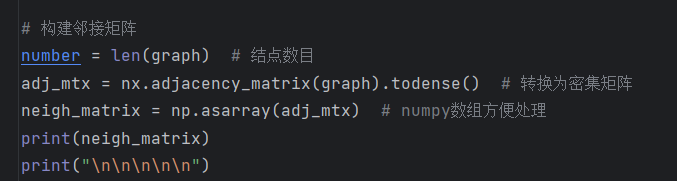


图2.3有向图转换为邻接矩阵

最后我们只需要进行迭代处理即可，迭代部分如下所示：

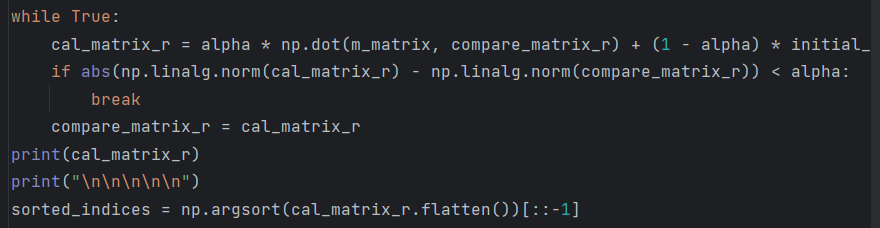


图2.4进行迭代处理

### 2.3.2 遇到的问题及解决方式

问题一，如何将得到的有向图转换为邻接矩阵。

在networkx库中给我们提供了networkx.adjacency\_matrix方法，可以十分快捷的将有向图直接转换为邻接矩阵。由于邻接矩阵大部分时候都常为稀疏矩阵，而稀疏矩阵会导致后续计算中的一些过程会降低处理速度，所以使用todense()将稀疏矩阵转换为密集矩阵。

问题二，如何衡量矩阵大小，如何将其做差比较与我们所给误差的大小。

在numpy库中给我们提供了numpy.linalg.norm方法，帮助我们直接计算矩阵的范数。关于范数，是矩阵元素的某个函数，可以帮助我们衡量矩阵的大小。

### 2.3.3 实验测试与结果分析

在执行PageRank.py程序后，我们按照PageRank值的大小从大到小将结果写入result.csv文件。结果符合预期值，结点入度越多，出度适当较小PageRank值越高，代表其结点更重要。

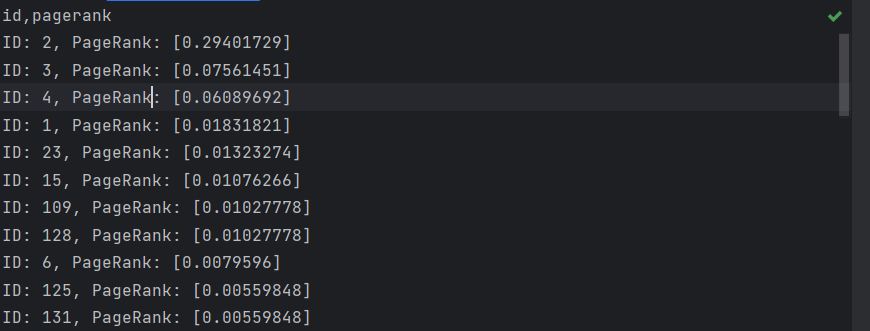


图2.5部分输出结果

## 2.4 实验总结

关于实验部分的提出的问题思考：

preprocess.py文件实现了以下的步骤：一是读取别名信息，构建别名到PersonId的映射关系。二是读取人名信息，构建PersonId到Name的映射关系。三是对每个邮件中的MetadataTo和MetadataFrom进行格式化处理，将所有人名转换为小写并去除逗号，并根据别名信息将别名转换为对应的PersonId。完成了映射。

在查询PageRank算法中，我在网上了解了一下信息：

Pagerank算法在搜索引擎领域中被广泛应用，是Google搜索引擎排名算法的核心之一。它通过对网页之间的链接关系进行分析和评估，对每个网页赋予一个权重，从而使得搜索引擎能够更好地为用户提供有用的搜索结果。Pagerank算法基于网络中节点之间的链接关系，通过随机游走模型计算每个节点的重要性，其计算过程比较简单，但结果准确性较高，尤其在大型复杂网络中的应用效果显著。

# 实验三 关系挖掘实验

## 3.1 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

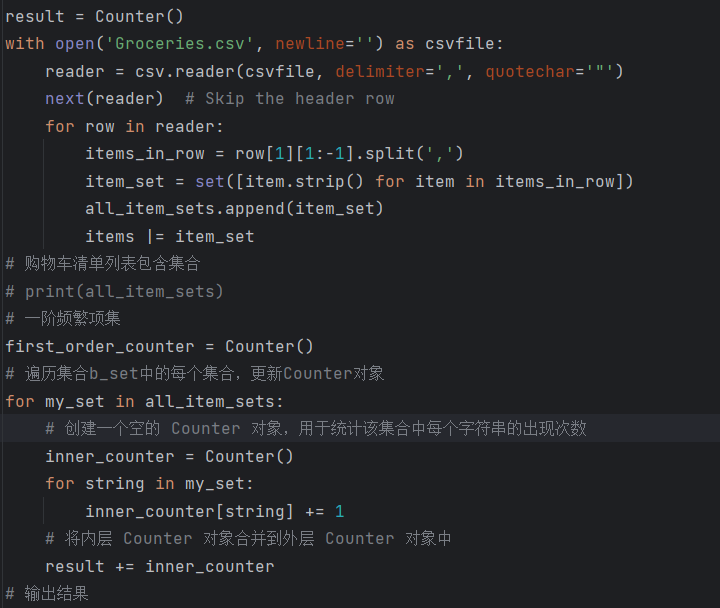
## 3.2 实验过程

### 3.2.1 编程思路

本实验需要使用Apriori算法，根据Groceries.csv文件中的物品列表，找出数据集中集中频繁出现的数据集合以及可能会有关联的物品组。

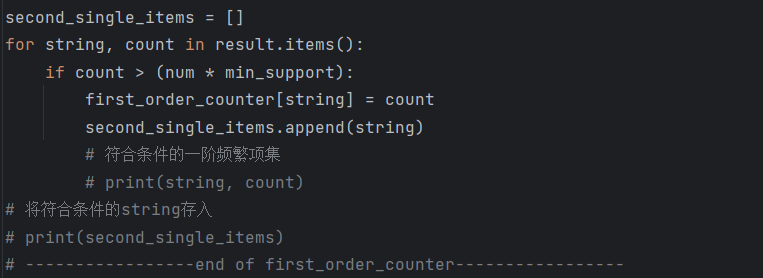
本实验设计到三个重要的概念：其一是关联规则，关联规则是形如X指向Y的表达式，并且X、Y互不相交，其可以通过支持度和置信度来衡量。其二是支持度，对于某个由物品构成的集合来说，其支持度等于该集合在所有购物清单集合中出现的次数，例如对于集合{A,B}，其在清单{A,B,C}中出现一次，而不在{A,C,D}中出现。其三是置信度，对于关联规则集合A指向集合B来说，置信度计算公式为，后两者是衡量频繁项集和关联规则的重要指标。值得注意的是，满足最小支持度是计算频繁项集和关联规则的前置条件！

本次实验，我们首先要实现csv格式文件的第二行中的购物清单进行读取，由于我们要得每个序号购物的集合以及出现的所有物品的集合，我们需要准备两个数据类型来保存这两组数据。观察到我们需要保存所有物品和所有物品清单，我们将每个清单视为一个集合，将所有清单集合保存在all\_item\_sets的集合当中，同时我们也将所有出现的物品保存在items的集合。同时观察到我们需要实时统计每个物品出现的次数，可以采用collections库中的Counter类型，其为字典dict的字类。值得注意的是对于dict类型的数据而言，其key为待统计的次数的数据，value为对应元素出现的次数。详细代码如下图所示：



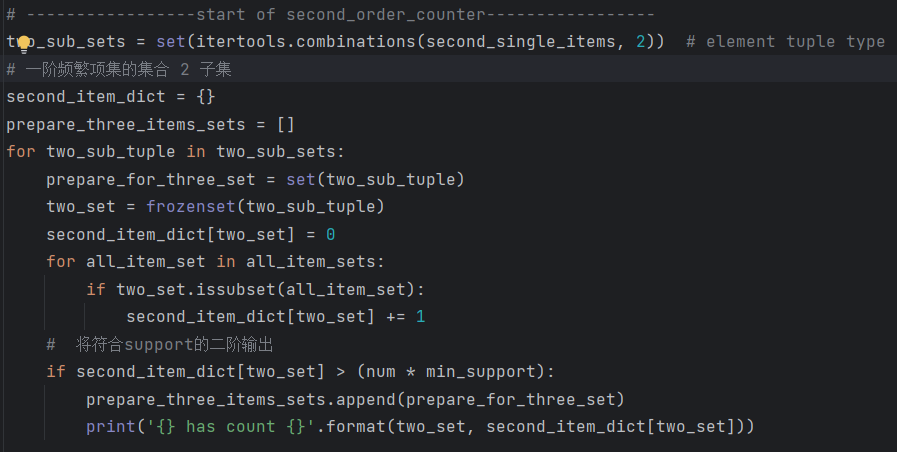
图一 读取文件部分

在读取好Groceries.csv文件后，我们要开始计算一阶频繁项集，并且为计算二姐频繁项集的备选集做准备。对与一阶频繁项集的筛选，我们初步只需要考虑最小支持度的要求即可。由于我们直接统计了次数，所以在筛选的过程中，我们只需要把Counter类型result中的value值和(物品数量\*min\_support)相比较即可。同时，作为符合最小支持度的物品存入sencond\_single\_items的集合作为二阶频繁项集的备选集。这一步的代码如下图所示：



图二 遍历result将符合最小支持度的存入备选集

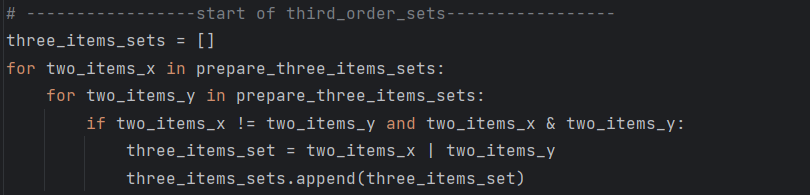
第三步，我们需要计算二阶频繁项集并且选择符合条件的作为三阶频繁项集的备选集。首先我们要从符合最小支持度条件second\_single\_items中选择两个元素作为二阶项集。这里我使用了itertools库中的combinations函数，从second\_single\_items集合中构造元素数为2的集合，并且存入second\_item\_dict中。并且在后续筛选符合条件的二阶频繁项集过程中，我们同时将符合条件的二阶频繁项集存入prepare\_three\_items\_sets中。相关详细代码如下图所示：



**找到元素数为2的子集**

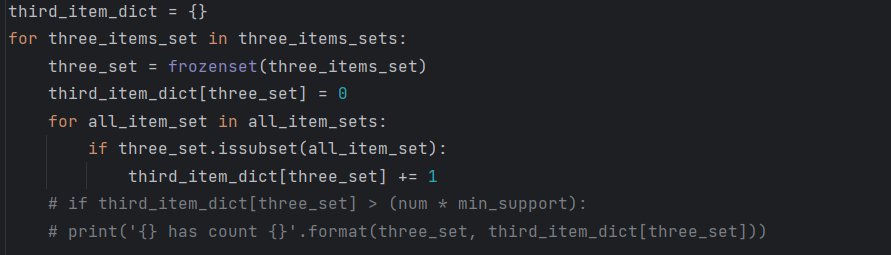
图三 得到二阶备选集，筛选二阶频繁项集

第四步，我们需要从三阶频繁项集的备选集prepare\_three\_items\_sets之中，获得元素数为3的三阶项集，然后去进行与上面相同的筛选。这里涉及到集合的运算，对于本次实验二阶频繁项集来说，若存在{A,B}和{A,C}那么可以得出一个备选集{A,B,C}。因此我们需要将相交的集合但不相等的集合进行合并，所以根据条件我们可以写出集合运算如下，假设这里的集合为set1、set2，则运算式为(set 1 != set2) and (set1 & set2)，然后再”new\_set = set1 | set2”即可得到三阶项集了。详细代码如下图所示：



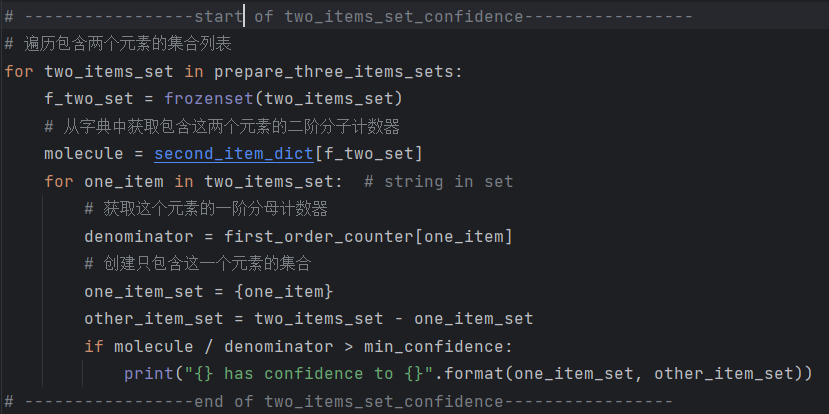
图四 对三阶频繁项集备选集进行集合运算得到三阶项集

接着为了得出结果我们只需要和上面相同的筛选过程即可。相关详细的代码如下图所示：



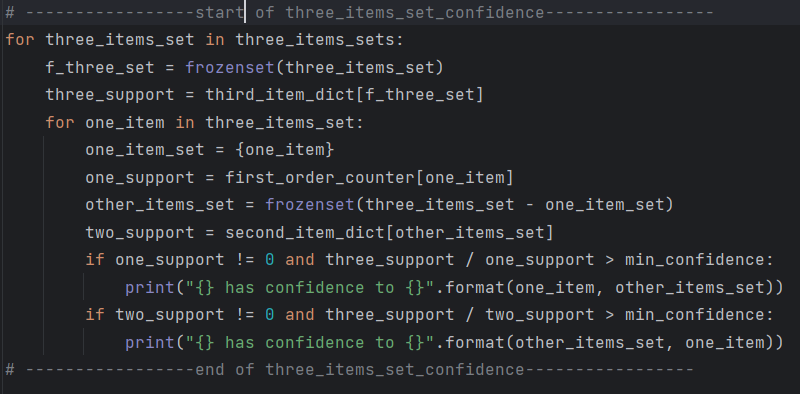
图五 筛选三阶项集，保存次数

到这里我们已经完成了频繁项集的计算，接下来只需要根据上面计算的结果去计算关联规则的置信度即可。由于置信度公式：（插图），所以我们得到其概率之比即为对应集合出现次数之比，而次数我们已经统计到各个字典型数据当中了，所以这一步操作较为简单。对于二阶频繁项集来说，比如项集set1中为{A,B}我们需要计算A对B的置信度，以及B对A的置信度。当我们知道二阶项集后，其实我们只需要判断A、B出现的次数是否小于(min\_convince \* set1\_cnt)即可，相关关键代码如下图所示：



图六 由二阶频繁项集计算一对一的关联规则

计算完一对一的关联规则，接下来就是计算一对二以及二对一的关联规则。这一部分相对于二阶频繁项集来说较为麻烦，我们需要先遍历所有的三阶频繁项集，然后在对三阶频繁项集进行遍历，将其拆分成两个集合，这里我将其命名为one\_item\_set以及other\_items\_set，我们将遍历过程的单个物品字符串转化为集合，再通过集合运算得到other\_items\_set。在进行筛选的过程中，我们需要排除support为0的一些特殊情况。相关关键代码如下图所示：



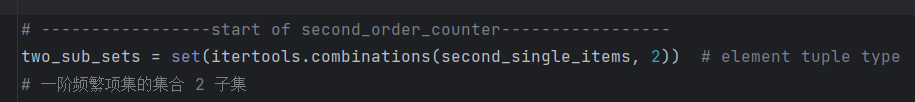
图七 由三阶频繁项集计算一对二、二对一的关联规则

### 3.2.2 遇到的问题及解决方式

**问题一，如何根据一阶频繁项集得到我们二阶项集的备选集？**

我一开始想通过直接遍历的方式，将一阶的这些字符串直接互相组合构成二阶项集的备选集，这样的可能会涉及两次for循环，导致该步骤的时间复杂度陡然增加，延缓代码运行。在网上查询集合的操作函数过后，可以选择itertools库中的combinations函数来进行该项操作，这个函数重在“组合”返回一个tuple元组类型的数据，每个元组中有2个元素。最后转化为集合类型方便后续处理。

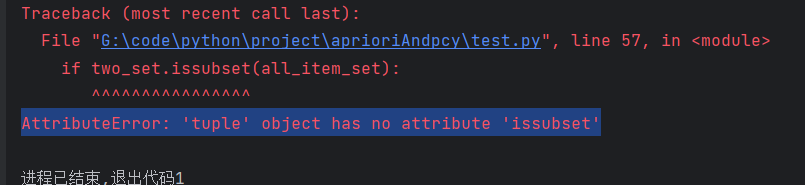
相关程序处理如图所示：



图八 由一节频繁项集获得二阶项集备选集

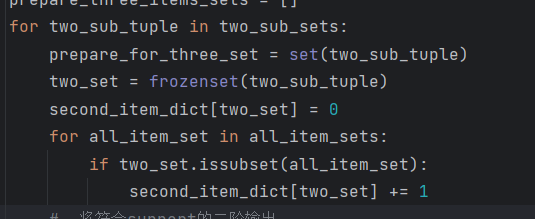
**问题二，如何将集合作为列表查询的索引来存储出现次数？**

我一开始没有细想写代码的时候直接将上一阶项集集合作为列表的索引，后来发现报错如下所示：



图九 索引出现错误

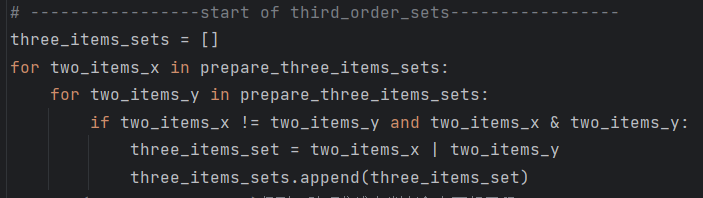
这个问题出现的原因是，作为一个集合来说，它可以自由的添加、删改元素。但是对于列表的索引来说，却不能是动态的，必须要求是固定的。所以这里我们需要将这个集合“固定”，这一步我们可以通过forzenset()的函数来实现。这个函数会返回一个冻结的集合，冻结后的集合不能再添加或者删除任何元素。另设一个作为索引的冻结集合two\_set，在遍历中将其赋值two\_set = forzenset(two\_sub\_tuple)，程序正常运行，相关代码如下所示：



图十 将集合进行frozenset()冻结操作

**问题三，如何根据二阶频繁项集得到我们三阶项集的备选集？**

这个问题我没有很好的处理办法，想着就是通过遍历一个个去生成三阶项集。通过网上查询，我得到了一个比较好的生成办法：在二阶频繁项集，将有两个元素重复的项集可以进行合并生成两个三阶项集，对于只有一个相同元素的，就无需考虑了。这一步设计到几个集合运算规则：首先我们假设两个集合set1和set2，(set1 & set2)是判断两个集合是否有交集，如果有就返回1，没有就返回0；(set1 | set 2)是将两个集合进行合并。所以对于本操作，只需要将不相等且有交集的进行合并的操作即可，相关关键代码如下图所示：

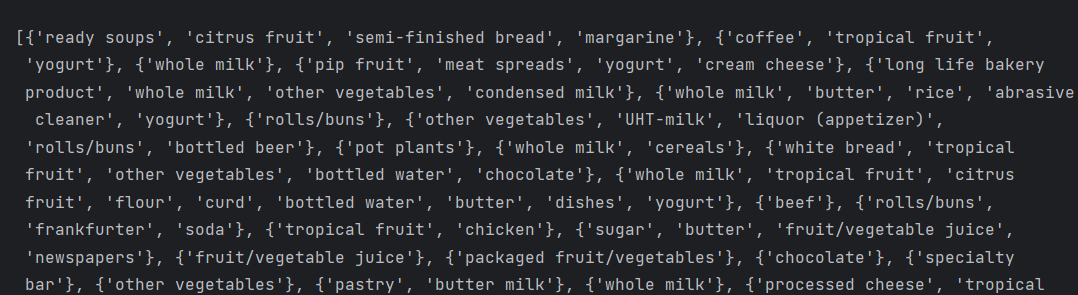


图十一 由二阶频繁项集得到三阶项集的备选集

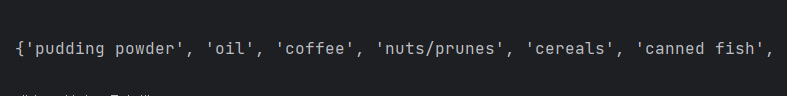
### 3.2.3 实验测试与结果分析

在实验测试步骤中，先将所有的输出语句注释掉，在代码最后逐个输出涉及到的所有数据：购物车清单列表all\_item\_sets、物品列表items、一阶频繁项集first\_order\_counter、二阶频繁项集second\_item\_dict、三阶频繁项集third\_item\_dict、

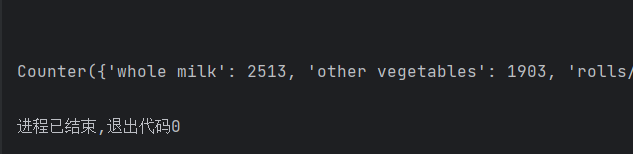
三阶备选集three\_items\_sets。相关的输出如下图所示：



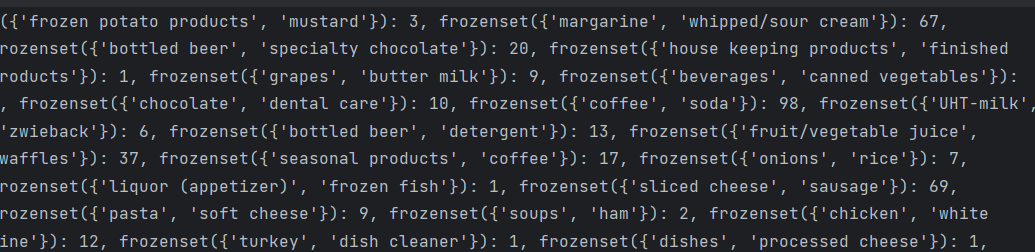
图十二 部分all\_item\_sets内容输出



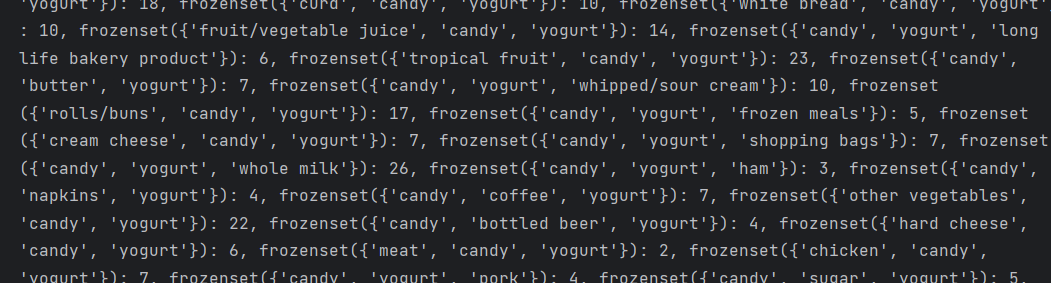
图十三 部分items内容输出



图十四 部分first\_order\_counter内容输出



图十五 部分second\_item\_dict内容输出



图十六 部分third\_item\_dict内容输出

对结果分析发现：一是，在所有关联规则中，‘whole milk’ ‘vegetables’ ‘yogurt’三词出现频率较高，这反映出三者是购物中经常会和其他商品一起购买的物品，这个结论**可以帮助商家更好的了解市场需求，从而优化产品组合**；二是，这些出**现次数多的物品，更容易成为关联规则的前提**，可以帮助我们挖掘出更多的关联规则；三是，由于这些物品在很多交易中都出现过，这个物品也**可能不具有区分度**，和其他物品出现频率相近，那么**可能挖掘出来的关联规则没有意义**。

整体代码思想、步骤正确，能够得到最终的正确结果。

对本次实验进行结果分析，与其他同学商讨算法的编写步骤，以及在助教指出相关问题后，我对我的代码有了如下分析：

第一项，**数据命名不规范还可以重新改进**，以一个固定的规范来进行命名。比如本实验涉及到的几个频繁项集，我没有统一命名，会显得有些杂乱，在结果输出时也会让代码显得不整洁。在中间的算法过程中，也会出现一些数据命名混乱的问题，最后的课设过程中需要加以改进。同时部分数据的类型也有混乱的情况，还需要统一进行类型转换，或者使用type()的函数来判断其类型，来帮助后续代码的编写。

第二项，可以采用在MapReduce中出现的**多线程函数来加快求频繁项集的备选集**的过程。由于我们在处理备选集的时候，是对一个有很多数据的文件同时处理的，所以可以考虑使用之前涉及到的threading库进行多线程处理。在本实验的代码中，处理数据速度较慢，因为里面有较多次的for循环以及for循环的嵌套，导致数据处理慢。

第三项，**有一些计算步骤是重复的**，比如根据备选集去挑选符合最小支持度的项集这一步，可以加以改进。比如建立一个函数select\_support\_set(set1,set2,set3)这里的set1是备选集，set2是购物车清单，set3是将符合要求的元素存入的集合。这样可以减少代码的重复过程，减少代码长度，节省空间。

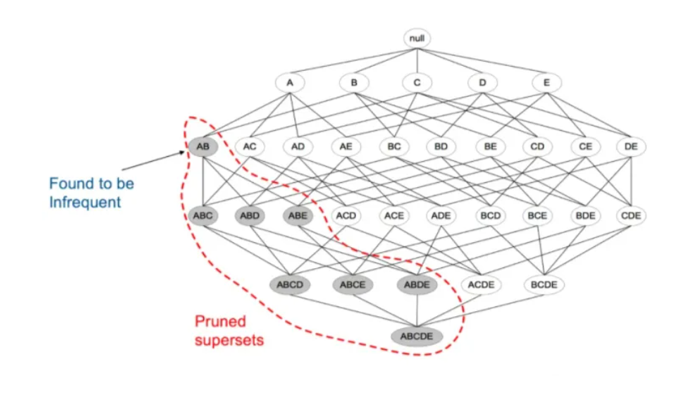
## 3.3 实验总结

首先是针对3.2.3中提到的“这个物品也**可能不具有区分度**，**可能挖掘出来的关联规则没有意义**”的问题，我们可以在A-priori算法中最小置信度和最小支持度的基础上添加提升度（lift）来衡量一个关联规则的是否有意义。假设这个关联规则涉及购买物品集合A和购买物品集合B，我们可以知道A对B的置信度和B对A的置信度分别为：

那么可以求出提升度L（lift）：

如果提升度=1说明A和B没有任何关联，如果<1，说明A事务和B事务是排斥的，>1，我们认为A和B是有关联的，但是在具体的应用之中，一般认为提升度>3才算作值得认可的关联，但是在实际应用过程中受购物总单数的影响比较大，需要多方面来考虑关联规则是否具有现实意义。

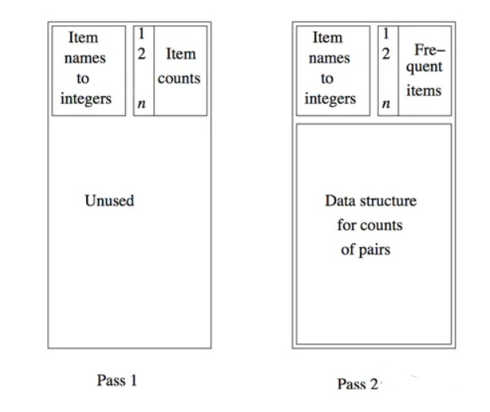
接下来对A-priori算法进行分析，本次A-priori算法有以下的优点：一是算法实现较为简单，只需要按照最小置信度、最小支持度的要求遍历求解，将符合要求的存入作为备选集进行下一阶挑选即可；二是数据结构较为简单，实验中我统一采用了集合结构，这种组织方式不仅易于处理，还可以有效的减少内存的消耗；三是处理过程中我们采用类似于剪枝的操作，即根据原理如果一个物品集合不频繁，那么它的子集也不会频繁，可以根据这个原理来减少备选集的数量，提升算法的效率；四是A-priori算法可以快速处理稀疏的数据集以进行数据挖掘，由于三中采用了剪枝的操作，所以我们加快处理速度，如下图所示：



图十七 剪枝操作

本次A-priori算法有以下的缺点：一是对数据扫描次数过多，由于算法中多次出现for循环来遍历查询数据导致算法的时间复杂度比较大，会占用大量的资源；二是算法可能会产生很多候选集，这些候选集会导致算法的内存占用量较大，降低运行效率；三是算法适应面窄，只采用支持度和置信度来衡量关联规则可能会存在实际意义不大的结果，这一点上面也提到了。

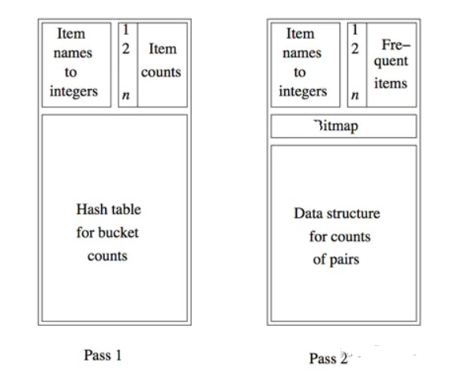
针对A-priori算法内存使用率不够这一缺点我们可以改进使用PCY算法。PCY算法相对于A-priori算法而言有了以下几个改进：一是通过使用哈希减少扫描数据的次数，在PCY算法中我们只需要进行两次扫描分别实现生成哈希桶和计算候选集的支持度；二是增加了内存的使用效率，在PCY算法中只需要在第一次扫描中存储哈希桶的计数值我们就可以减少存储空间的使用；三是在计算备选集时使用了哈希桶来计算，减少备选集的生成，方便处理更大的数据集。如下图所示，为A-priori算法两次扫描图



图十八 A-priori算法两次扫描图

根据这个图可以知道第一遍有大部分空间没有使用，在第二遍中，进行了最小支持度筛选得到后续阶的备选。

再来看PCY算法扫描两边后的内存占用图：



图十九 PCY算法两次扫描图

PCY算法在第一遍扫描时对购物篮进行检查,且不仅对篮中的每个项的计数值加1，而且通过一个双重循环生成所有的项对。对每个项对我们将哈希结果对应的桶元素加1，而且项对本身并不会放到哈希桶中，因此它只会影响桶中的单个整数。也就是第一遍扫描时图中下面的部分。第一遍扫描结束时，每个桶中都有一个计数值，记录的是所有哈希到该桶中的项对的数目之和，如果该桶的计数值高于支持度阈值，那么该桶称为频繁桶。

所以很直接就能看出PCY算法相对于A-priori算法减少了算法时间加快效率，减少了空间复杂度，

在现实生活中关系挖掘算法对人们的生活影响很大，它可以通过挖掘数据之间的关系来帮助人们更好的分析数据，进而发现数据中的规律，提高决策的准确率。比如最经典的购物，帮助商场更好的摆放商品顺序。在学习和实践过程中，我了解到体会到关系挖掘的挑战性。一方面，关系挖掘需要我们具备一定的数据分析和建模能力；另一方面，关系挖掘也需要我们具备一定的业务知识和领域理解能力。在现在的信息社会中，随着数据量的不断增加和数据应用场景的不断拓展，关系挖掘技术还存在一些待解决的问题和挑战。例如，如何改进算法来让其更好的处理大量的数据，又或者如何添加衡量标准来确定关联规则。

# 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,进一步认识聚类算法的实现；

2、分析kmeans流程,探究聚类算法原理；

3、掌握kmeans算法核心要点；

4、将kmeans算法运用于实际，并掌握其度量好坏方式。

## 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。效果展示图可如图1.1所示。



图4.1 葡萄酒数据集在黄酮和总酚维度下聚类图像（SSE为距离平方和，Acc为准确率）

## 4.3 实验过程

### 4.3.1 编程思路

本实验需要运用到k-means算法，根据归一化.csv文件中的十三维数据，来将这些数据点在二维平面图中聚成3个簇中，并且以点图的形式表示出来。

本次实验涉及到几个比较重要的概念：其一是簇，簇是指具有相似特征的数据对象的集合，在此次实验中我们将把这些点分成三个簇；其二是簇中心，这个点是簇中所有数据对象的均值向量，可以一定程度上表示簇的特征，在本次实验中其未簇内所有数据点的平均值；其三是距离，本次实验中我们使用欧式距离来衡量数据对象之间的相似度；其四是损失函数，我们使用簇内平方误差和(SSE)来判断，这个值是来衡量聚类的效果，其计算公式如下：

其中，K是簇的个数，是第i个簇中的所有数据点的集合，是第i个簇的中心点（即簇中所有数据点的平均值向量），SSE的值越小说明聚类生成的簇效果越好。

首先讲一下k-means算法的大概思路：我们首先会提前确定几个常数：k、tolerance、max\_iter。它们分别是簇类的个数、差值和迭代次数，本次实验没有给定迭代次数和差值，我们假设其为300和0.0001方便后续迭代时处理。算法开始时，会随机选择3个点作为第一次迭代的初始中心点。对于数据集中的每个数据点，我们去计算这个点和3个聚类中心点之间的距离，然后根据距离大小将其划分到距离最近的那个聚类中心所在的簇。这样分好所有的点后，我们对于每个簇中的点再去重新计算簇中所有点的平均值，得到了下一次迭代的新的中心点。重复上述的几个步骤直到到达迭代次数或者满足差值为止，我们会得到3个簇，到此k-means算法结束，我们只需要再去分析精准度（accuracy）和SSE即可。

我们首先要读取“归一化数据.csv”文件，文件内容第一行是已经分好的类别（1，2，3），后面的十三栏是葡萄酒的十三维数据，部分内容如下图所示：

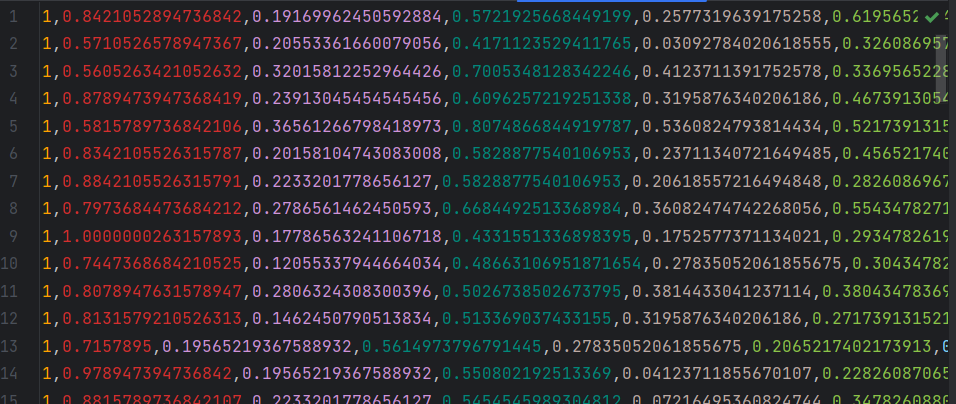


图4.2 归一化数据.csv部分内容

我们通过pandas库中的read\_csv函数来读取文件内容，并将csv文件中除了第一列的数据以外的数据转换为Numpy类型数据。在后续主函数中，利用list函数将其类型改变。相关代码如下图所示：

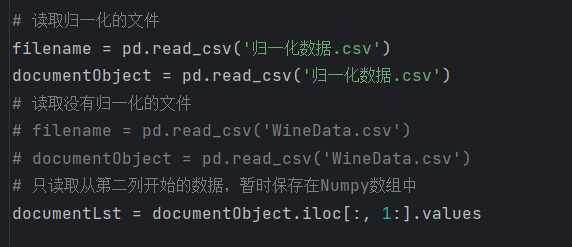


图4.3 将文件中的除第一列数据外，转化成Numpy数组

接着我们定义一个python的类，名为“k-means”，并且在“\_\_init\_\_”方法中将提前设定好的几个参数定义出来，方便后续的迭代处理，相关代码如下图所示：

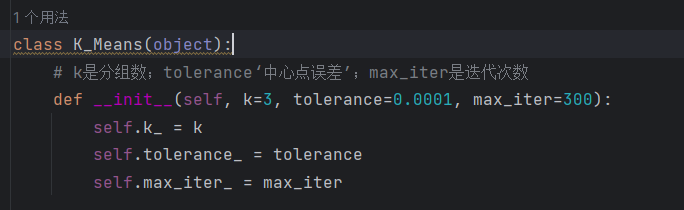


图4.4 k-means类中定义的三个参数

接着我们在fit方法中实现迭代步骤，首先要初始化3个中心点，以开始第一次的迭代。这一步使用random函数来实现随机初始化3个中心点，关键代码如下图所示：

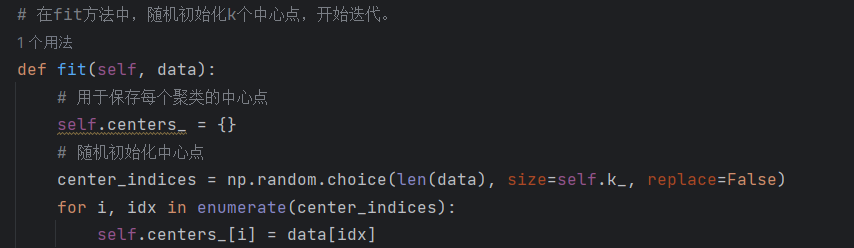
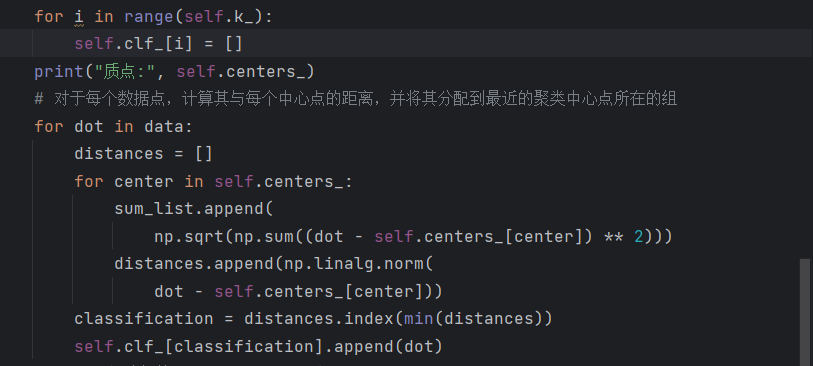


图4.5 随机初始化3个中心点

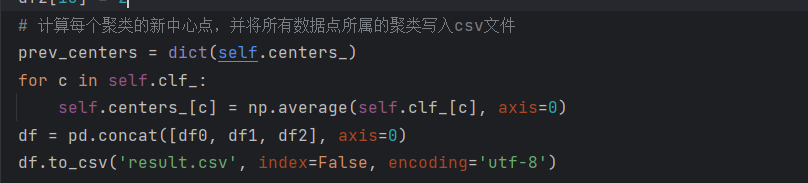
确定好3个中心点后，先初始化分好三个簇存入self.clf中，在遍历中对于每个点去计算到这三个中心点的距离，并且根据距离的大小将其分配到不同的中心点中去。在这一步利用两个for循环，先遍历所有点然后对每个点遍历所有的中心点，计算距离后找到最小距离的中心点，将这个点存入对应的簇。相关代码如下图所示：



**计算到中心点的距离**

图4.6 计算每个点到中心点的距离

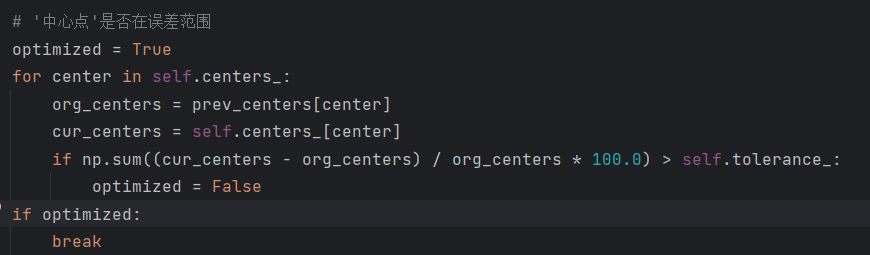
根据每个簇中的数据重新计算簇的中心点，方便下一次迭代的处理，并且判断两者误差是否符合tolerance的值，相关代码如下图所示：



**计算新的中心点**

图4.7 计算每个簇中的平均值，找寻下一个中心点

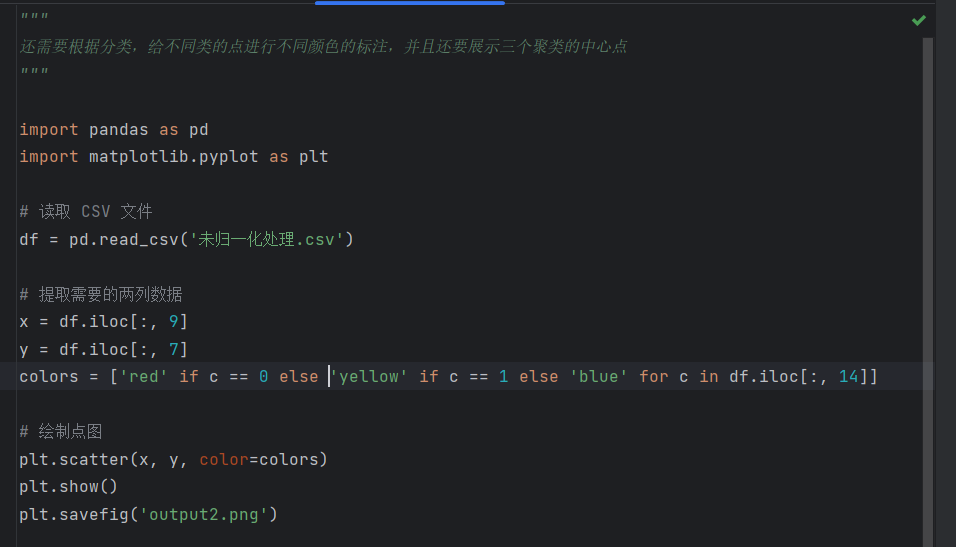
在算法的每次迭代中，会计算每个聚类的新中心点，然后比较新旧中心点之间的误差。当所有聚类的中心点误差都小于给定的tolerance值时，就认为聚类结果已经优化，此时变量optimized会被置为True，迭代结束。如果某个聚类的中心点误差超过tolerance值，则将optimized置为False，继续迭代。相关代码如下图所示：



**判断是否在误差范围内**

图4.8 判断是否在误差范围内

我们需要将分配好的点，以三个颜色来区分不同簇的点，对于k-means算法处理得到的CSV文件进行数据读取和提取，然后对其中两列数据进行可视化绘制。首先，代码利用 Pandas 库的 read\_csv() 方法读取名为 'result.csv' 的 csv文件，使用 iloc 方法提取出文件中的第9列和第7列的数据分别赋值给 x 和 y 两个变量。接下来，根据 csv 文件中第14列的分类信息，将不同簇的点分别标记为红色、黄色和蓝色，在二维点图展示出来。最后通过使用 plt.show() 方法来显示绘制好的点图，并使用 plt.savefig() 方法将图像保存为名为 'output.png' 的文件。相关代码如下图所示：



**根据两列数据绘制点图**

图4.9 根据处理后的csv文件绘制点图

### 4.3.2 遇到的问题及解决方式

**问题一：如何实现随机生成3个中心点？**

在我们从csv文件获取所有点的数据之后，我们了解到点的个数为178，所以我们只需要在1~178中随机生成三个整数作为点的列表索引即可得到三个随机的中心点。在python中，random库给我们提供了random.sample的函数，我们需要确定范围使用range(1,178)，相关代码如下图所示：

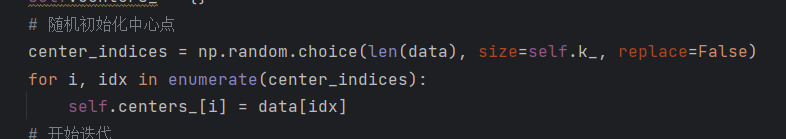
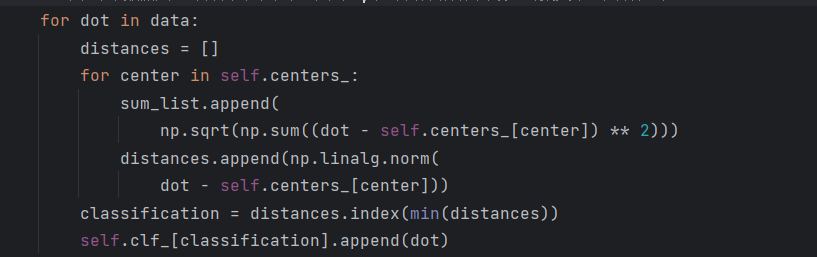


图4.10 随机化三个簇的中心点

**问题二：如何计算SSE，也即如何留存每个点到中心点的距离？**

由于每次迭代中，我们都提前求出过所选中心点的坐标，所以我们只需要遍历一下存入了所有点数据的data列表里，采用欧式距离计算到中心点的距离。不仅要将同一个点的三个距离存入一个列表distances里，还要将这些距离存入一个单独的列表里，方便后续处理。相关代码如下图所示：



**根据距离大小将其分给不同的簇**

图4.11 计算SSE

### 4.3.3 实验测试与结果分析

**实验测试：**

本次实验需要将组类平方和SSE和精确度输出，将所有代码按照要求运行，所得的最后结果如下图所示。值得注意的是由于第一遍选取点的随机，最后的SSE和准确度都不会相同，这次结果尽量选取了SSE较小的一次结果，实际上会随选取第一轮中心点的不同而改变，结果如下图所示：

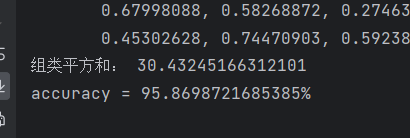


图4.12 实验结果

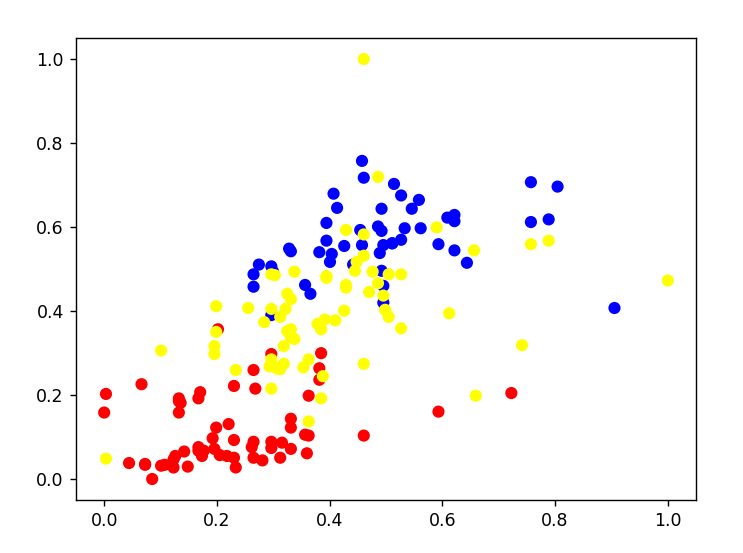
 在k-means算法结束后，使用create\_map.py将分类的结果根据所选取的两列数据来画出二维点图，最后得到的结果如图所示：

图4.13 自己编写的算法运行后的二维点图

下面是直接使用库中的k-means函数直接产生的二维点图：

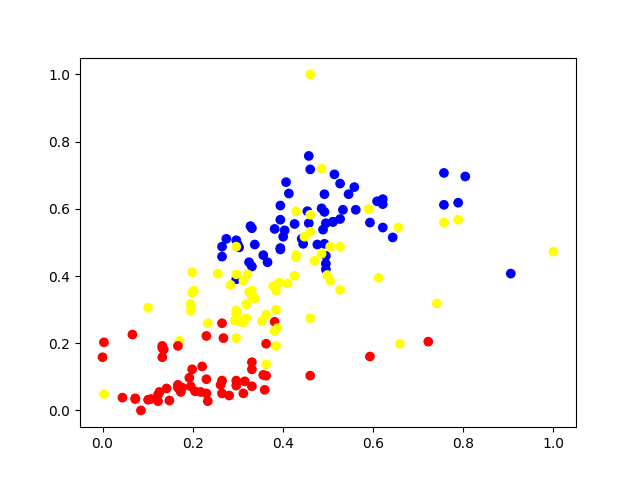


图4.14 直接使用库函数产生的聚类结果二维点图表示

下面是直接使用没有归一化的数据直接产生的二维点图：

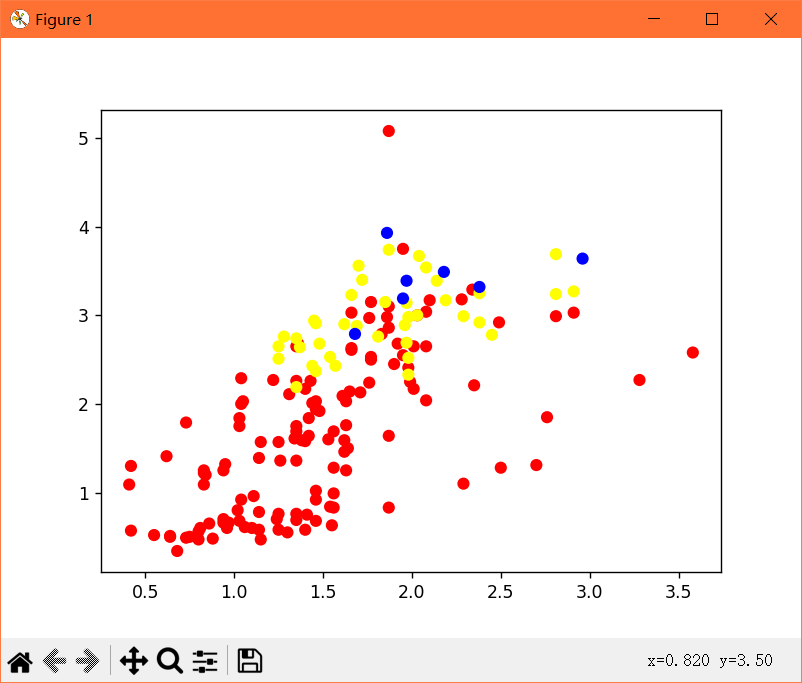


图4.15 直接使用未归一化聚类结果二维点图表示

**结果分析：**当以使用库函数的二维点图结果作为衡量标准时，我们可以发现本次实验的聚类效果还不错，只有少数点的聚类出现了问题。我认为这可能是由于随机选取的点或者预设的常量不够准确所致，需要进行更多次的迭代或降低容忍度来改进。

对于未归一化的数据，我们可以明显看出聚类效果非常差。这主要是因为数据需要进行归一化处理。我查询了一些资料后了解到，以本次葡萄酒数据集为例，数据集中的每个样本都包含了13个特征，包括ph值、酒精含量、含糖量等等。这些特征的单位不同，甚至单位的量级也各不相同。如果不对这些特征进行归一化处理，K-Means聚类算法中的某些特征将比其他特征更具有权重，从而影响聚类结果。例如，某些维度的数据量级在几十到几百，而有些数据只有零点几的量级，这会严重影响聚类算法的效果。

## 4.4 实验总结

本次实验我首先认识到了归一化处理的重要性。通过归一化处理我们可以将数据的特征缩放到我们所需要的一个比例和范围内，使得所有特征对聚类结果的影响差距尽量减小甚至消除。假设我们不对这些数据进行归一化，可能让聚类结果不准确。

这次实验主要学习并实践了k-means算法，可以将数据集划分为多个簇，每个簇内部的数据相似度较高，而簇之间的数据相似度较低。我们选择使用葡萄酒数据集进行聚类分析，该数据集包含了 多个特征，用来描述葡萄酒的不同属性。有些特征的数据范围不同，因此需要对数据进行归一化处理，以确保所有特征的数据范围都在 0-1 之间，避免某些特征对聚类结果产生过大的影响。为了用精确数据来衡量聚类结果，我们可以通过SSE和准确度来判断，其中我们在迭代中**尽量使得簇内距离越来越小，而簇与簇之间的距离越来越大。**在分好类后，我们通过和数据集给出的类别作比较，我们可以计算精确度。并且我也通过使用matplotlib.pyplot库中的相关函数来绘制二维点图。

最后，本次实验让我了解聚类算法的原理和实现方法，并通过实际操作来了解算法的应用场景和需要注意的地方。同时，也让我们掌握了一些数据分析和可视化的基本技能。

# 实验五 大作业推荐系统

## 5.1实验目的

1. 了解推荐系统的多种推荐算法并理解其原理。
2. 实现**User-User的协同过滤算法**并对用户进行推荐。
3. 实现**基于内容的推荐算法**并对用户进行推荐。
4. 对两个算法进行电影预测评分对比
5. 在学有余力的情况下，**加入minihash算法**对效用矩阵**进行降维处理**

## 5.2 实验内容

给定MovieLens数据集，包含电影评分，电影标签等文件，其中电影评分文件分为训练集train\_set和测试集test\_set两部分

基础版必做一：**基于用户的协同过滤推荐算法**

对训练集中的评分数据构造用户-电影效用矩阵，使用**pearson相似度计算方法计算用户之间的相似度**，也即相似度矩阵。**对单个用户进行推荐时，找到与其最相似的k个用户，用这k个用户的评分情况对当前用户的所有未评分电影进行评分预测，选取评分最高的n个电影进行推荐。预测评分按照以下方式计算：**

在测试集中包含100条用户-电影评分记录，用于计算推荐算法中预测评分的准确性，对测试集中的每个用户-电影需要**计算其预测评分**，**再和真实评分进行对比，误差计算使用SSE误差平方和**。

选做部分提示：此算法的进阶版采用minihash算法对效用矩阵进行降维处理，从而得到相似度矩阵，注意minihash采用jarcard方法计算相似度，需要对效用矩阵进行01处理，也即**将0.5-2.5的评分置为0，3.0-5.0的评分置为1**。

基础版必做二：**基于内容的推荐算法**

将数据集movies.csv中的电影类别作为特征值，**计算这些特征值的tf-idf值**，得到关于电影与特征值的n（电影个数）\*m（特征值个数）的**tf-idf特征矩阵**。根据得到的tf-idf特征矩阵，**用余弦相似度的计算方法，得到电影之间的相似度矩阵**。

对某个用户-电影进行预测评分时，获取当前用户的已经完成的所有电影的打分，通过电影相似度矩阵获得已打分电影与当前预测电影的相似度，按照下列方式进行打分计算：

**选取相似度大于零的值进行计算，如果已打分电影与当前预测用户-电影相似度大于零，加入计算集合，否则丢弃**。（相似度为负数的，强制设置为0，表示无相关）假设计算集合中一共有n个电影，score为我们预测的计算结果，score’(i)为计算集合中第i个电影的分数，sim(i)为第i个电影与当前用户-电影的相似度。如果n为零，则score为该用户所有已打分电影的平均值。

**要求能够对指定的userID用户进行电影推荐，推荐电影为预测评分排名前k的电影。userID与k值可以根据需求做更改。**

推荐算法准确值的判断：**对给出的测试集中对应的用户-电影进行预测评分，输出每一条预测评分，并与真实评分进行对比，误差计算使用SSE误差平方和**。

选做部分提示：进阶版采用minihash算法对特征矩阵进行降维处理，从而得到相似度矩阵，注意minihash采用jarcard方法计算相似度，特征矩阵应为01矩阵。因此进阶版的特征矩阵选取采用方式为，**如果该电影存在某特征值，则特征值为1，不存在则为0，从而得到01特征矩阵**。

**进阶部分：**

本次大作业的进阶部分是在基础版本完成的基础上大家可以尝试做的部分。进阶部分的主要内容是使用**迷你哈希（MiniHash）算法对协同过滤算法和基于内容推荐算法的相似度计算进行降维**。同学可以把迷你哈希的模块作为一种近似度的计算方式。

协同过滤算法和基于内容推荐算法都会涉及到相似度的计算，迷你哈希算法在牺牲一定准确度的情况下对相似度进行计算，其能够有效的降低维数，尤其是对大规模稀疏01矩阵。同学们可以**使用哈希函数或者随机数映射来计算哈希签名**。哈希签名可以计算物品之间的相似度。

最终降维后的维数等于我们定义映射函数的数量，我们设置的映射函数越少，整体计算量就越少，但是准确率就越低。**大家可以分析不同映射函数数量下，最终结果的准确率有什么差别**。

**对基于用户的协同过滤推荐算法和基于内容的推荐算法进行推荐效果对比和分析，选做的完成后再进行一次对比分析。**

## 5.3 实验过程

### 5.3.1 编程思路

**基于用户的协同过滤推荐算法编程思路：**

本次实验我们需要运用到User-User的协同过滤算法，这是一种通过用户评分来计算相似性然后来进行推荐的算法。算法中相关关键系数和概念介绍如下：其一是相似性度量，用于衡量用户之间相似性的方法，比如余弦相似度、杰卡德相关系数、皮尔逊相关系数、聚类算法中提及的欧氏距离等等……这些系数可以用来衡量用户对同一个电影的评分的相似性。在本次实验中将采用皮尔逊相关系数来判断用户之间的相似度，皮尔逊相关系数的公式如下所示：

公式5.1 皮尔逊相关系数

其二是邻居选择，需要输入邻居的数量k，以来从这k个和被推荐用户相似的用户评分情况对当前用户没有评分的电影进行预测。其三是预测方法，本次实验中我们需要预测评分，预测方式如下公式所示：

公式5.2 预测评分公式

首先讲一下我理解的User-User协同过滤算法大概思路：首先要进行数据准备，读取对电影的评分数据，构造对应的矩阵；然后计算用户之间的相似性，通过皮尔逊相关系数来计算相似性的矩阵；再根据输入的k，来选择与被推荐用户最像的k个邻居用户；最后我们根据上面的公式去预测目标用户评分，再据此去为被推荐用户去推荐物品的列表。实验代码的函数以及大致流程如下图所示：

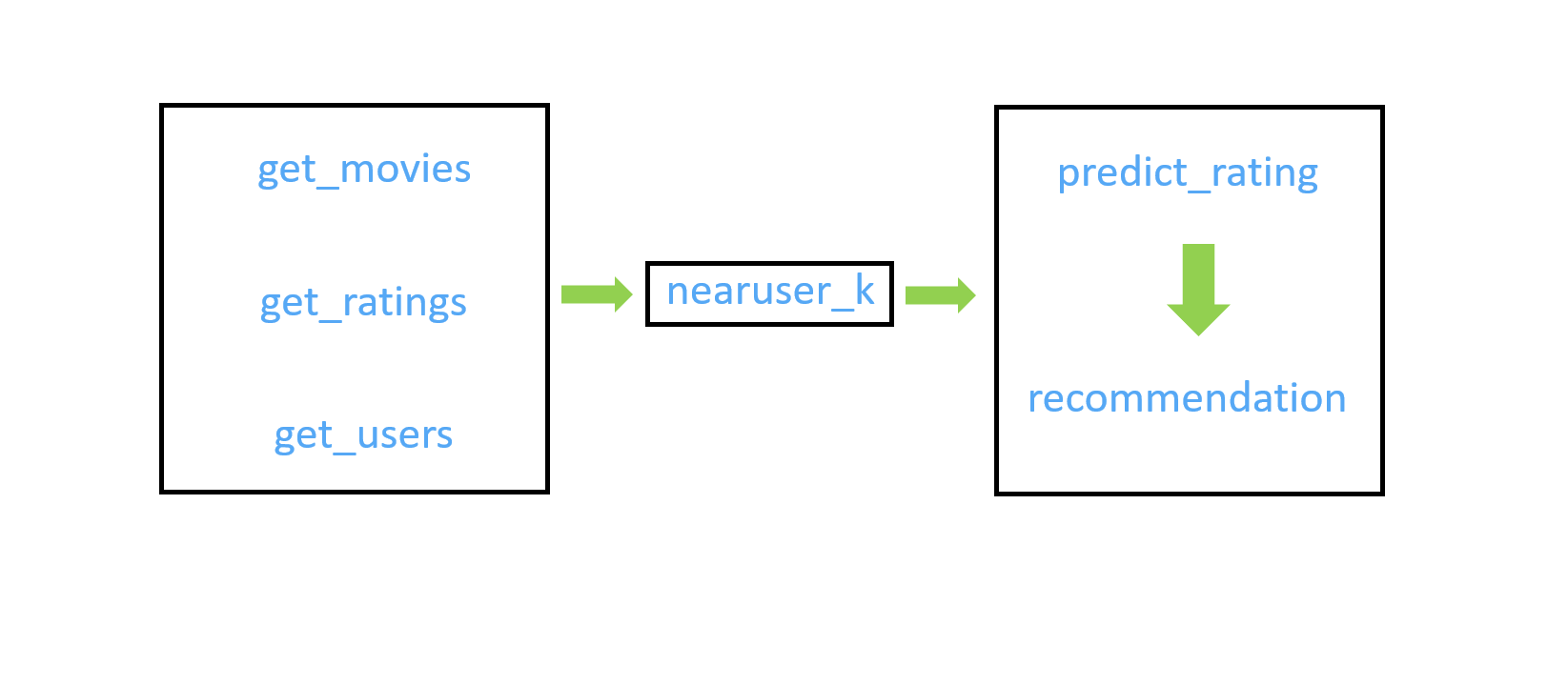
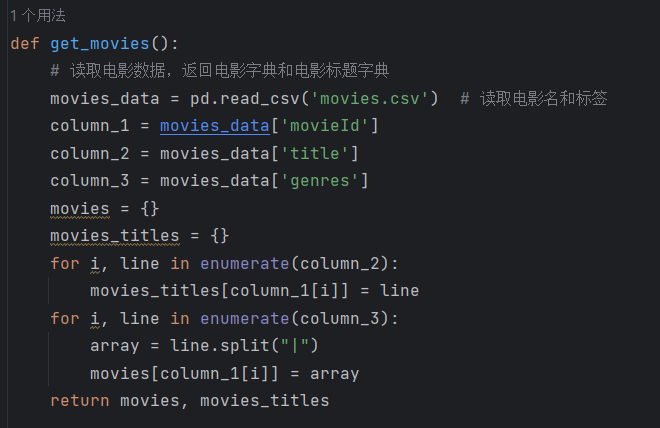


图5.1 实验函数大致流程

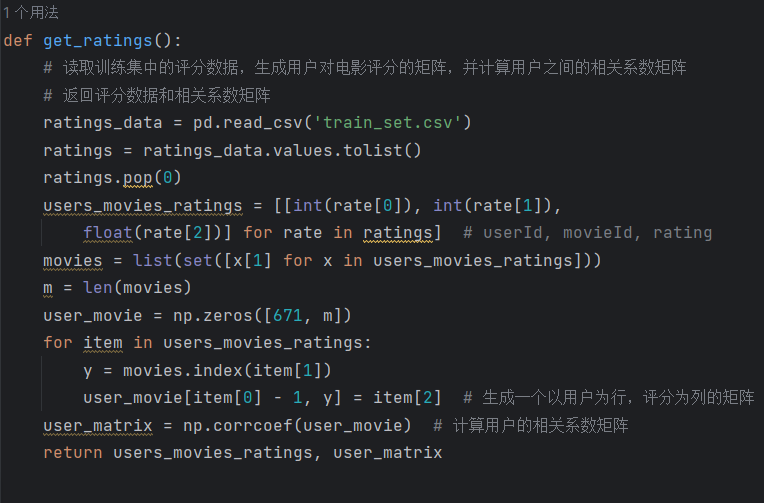
第一步定义**get\_movies()**函数来读取movies.csv文件，通过movieId来分别映射titles和genres，最后返回这两个映射。movies.csv文件中第一列是movieId，第二、三列分别是titles、genres。相关函数代码如下：



**通过movieId映射**

图5.2 get\_movies()函数

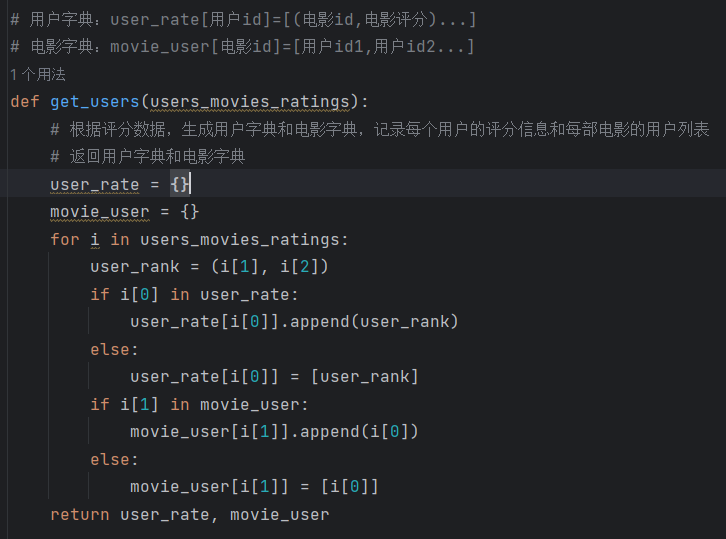
第二步定义**get\_ratings()**函数来生成用户-评分的矩阵、用户-用户之间的相关矩阵，返回评分数据和相关系数矩阵。对于用户-评分矩阵，我们可以通过for循环，来完成这个矩阵；对于用户-用户矩阵，我们可以直接使用numpy库中的corroef函数来计算用户与用户之间的皮尔逊相关系数的矩阵。相关代码如下图所示：



**构造两个矩阵**

图5.3 get\_ratings()函数具体实现

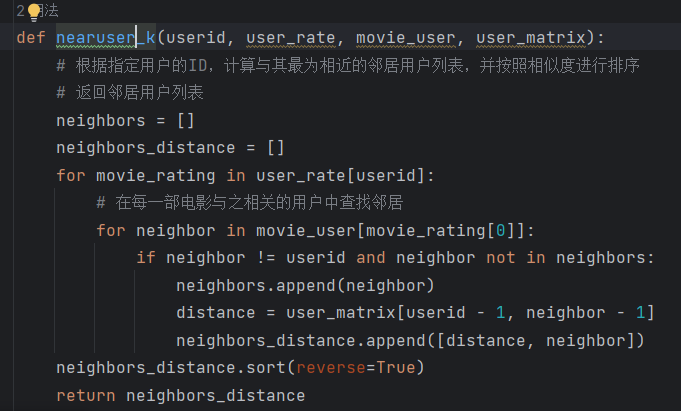
第三步定义**get\_users(users\_movies\_rating)**函数，通过上一个函数生成的用户-电影评分数据，在这一函数中去生成用户和电影的字典记录每个用户的评分信息和每部电影的用户列表，并返回这两个列表。相关代码如下图所示：



**构造两个字典**

图5.4 get\_users(users\_movies\_rating)函数具体实现

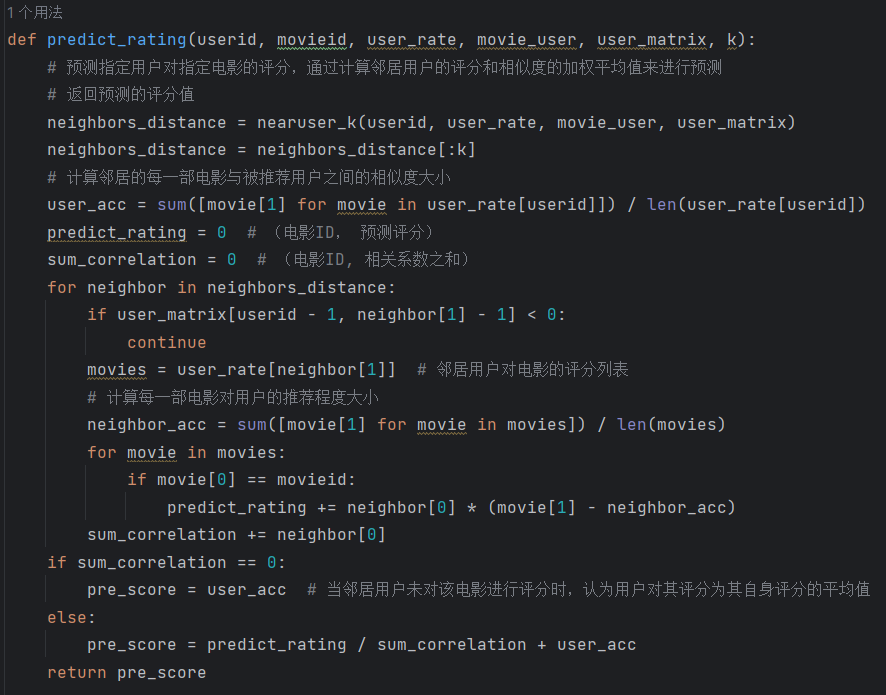
第四步定义**nearuser\_k(userid, user\_rate, movie\_user, user\_matrix)**函数。userid是被推荐者的ID，计算他的邻居用户列表；user\_rate是用户字典，可以查询每个用户的评分记录；movie\_user是电影字典，可以查询对此电影评分了的用户；user\_matrix是用户的皮尔逊相关系数矩阵。通过遍历在同一部电影下的所有的用户，去计算被推荐者对这个用户的距离（按照相似度来计算）。将所有和被推荐者有关的全部计算出来，然后按照相似度的由大到小顺序排序。相关代码如下图所示：



**生成邻近用户列表**

图5.5 nearuser\_k(userid, user\_rate, movie\_user, user\_matrix)函数具体实现

第五步定义**predict\_rating(userid, movieid, user\_rate, movie\_user, user\_matrix, k)**函数。相关参数有：userid被推荐者的id，计算预测他的指定电影评分；movieid是指定的电影id；user\_rate是用户字典，可以查询每个用户的评分记录；movie\_user是电影字典，可以查询对此电影评分了的用户；user\_matrix是用户的皮尔逊相关系数矩阵；k是预测时查询被推荐者k个邻近用户。通过公式来计算每部电影和被推荐用户之间的相似度大小。通过for循环遍历每个邻居，判断如果相似度（user\_matrix）小于零则跳过，否则获取这个用户的电影评分列表（movies）。接着计算每个邻居对电影的推荐程度，计算neighbor对所有电影评分的平均值，然后计算这个平均分和指定电影的评分差值，这个差值（predict\_rating）和邻居对被推荐者的相似度（neighbor[0]）进行加权计算，将最后的结果累加在邻居的相似度系数之和（sum\_correlation）。用于后续给被推荐者的推荐一些电影。相关代码如下图所示：



**计算邻居用户的相似度之和**

图5.6 predict\_rating(userid, movieid, user\_rate,

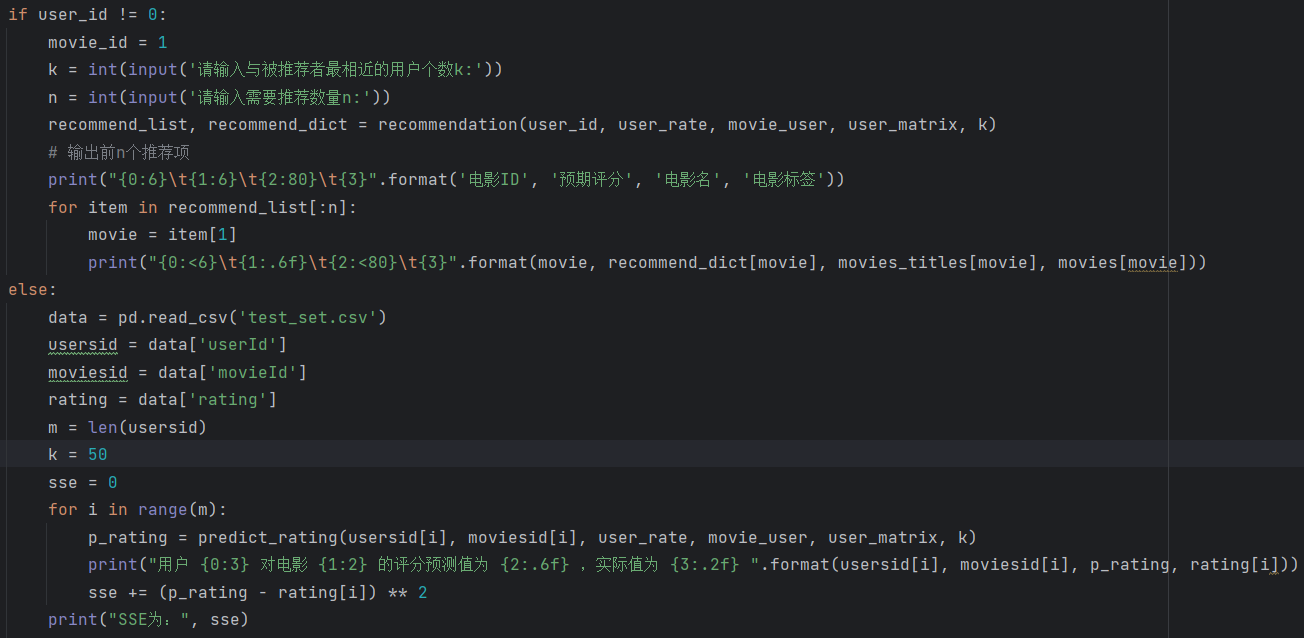
movie\_user, user\_matrix, k)函数具体实现

第六步定义**recommendation(userid, user\_rate, movie\_user, user\_matrix, k)**函数，这个函数的参数和上一个函数意义大致相同。这个函数基于用户之间的相似度，为指定用户生成推荐电影列表，并且计算推荐程度。为了计算每个邻居对电影的推荐程度大小，首先计算该邻居对所有电影的评分之和除以电影数量，得到邻居的电影评分的平均值。然后遍历邻居的评分列表，我们只用去看当前的指定电影，计算这个邻居对这个电影的推荐程度大小，还要将这些相似度累加。在计算推荐列表的时候，我们对每个电影去计算它的推荐程度，将用户-电影的相关系数之和去除以套公式计算的相关系数之和，在加上用户的平均评分得出一个电影的推荐程度，然后再将这个推荐表按照由大到小排序方便后续推荐。相关代码如下图所示：



图5.7 recommendation(userid, user\_rate, movie\_user, user\_matrix, k)函数具体实现

第七步编写main函数，分别实现测试集测试(test\_set.csv)和SSE，以及输入被推荐者ID，用于计算的邻居个数k，需要推荐的电影个数n，就可以输出n个推荐的电影、电影ID、电影预期评分、电影标签。我们根据用户第一个输入的数字使用if循环来判断，若0为测试集，不为0就是去给推荐者推荐n个电影。对于测试集而言，只需要先获得文件中的数据，遍历所有的用户去计算每个预测的评分即可。相关代码如下图所示：



**计算测试集中的预测评分**

图5.8 User-User算法主体运行部分

**基于内容的推荐算法编程思路：**

在这个算法中我们用来衡量的相似度为余弦相似度，其中余弦相似度的计算公式为如下所示：

公式5.3 余弦相似度计算公式

在这个算法中我们将把数据集movies.csv中的电影类别作为特征值，计算这些特征值的TF-IDF值，得到关于电影与特征值的TF-IDF特征矩阵。其中TF\_IDF计算公式如下图所示：

公式5.4 TF-IDF计算公式

首先，对于基于内容的推荐算法，我理解的流程如下：首先要将csv文件中的数据读取、进行数据准备，并且还要将每个电影类别作为特征值，求出一个特征向量；接着要进行TF-IDF计算，其中TF用于表示某个特征（类别）在电影中出现的频率，IDF表示这个特征在数据集中的重要性，而TF-IDF为这两者的乘积；然后通过构造特征矩阵，形成每个类别与每个电影在某个特征上的TF-IDF值；通过余弦相似度来计算每个电影之间的相似度得到一个两个电影之间相似度的矩阵，通过这个矩阵来进行推荐以及预测评分。实验代码的函数以及大致流程如下图所示：

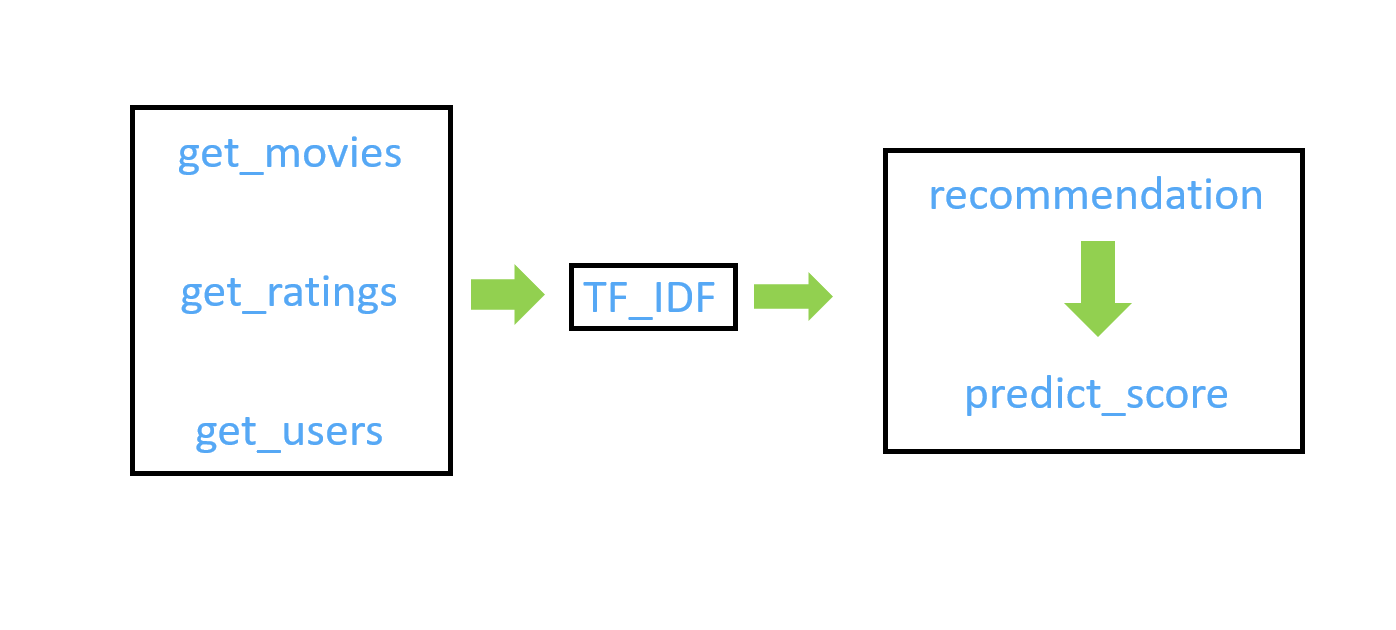


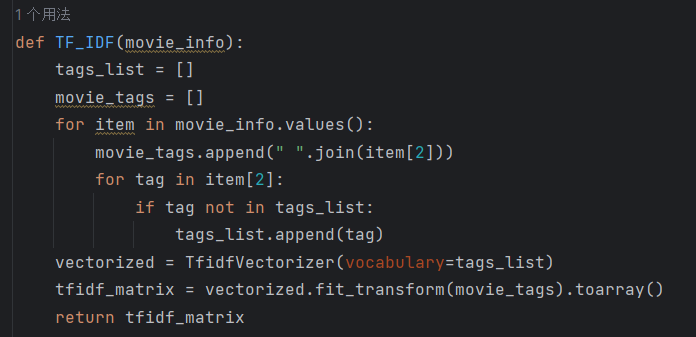
图5.9 基于内容的推荐算法函数大致流程

第一步定义**get\_ratings()**函数，负责从“train\_set.csv”文件读取用户的评分信息，并且读取以用户为行、电影评分为列的矩阵user\_movie。返回值r中存储userId，movieId，rating这三列数据。

第二步定义**get\_user(r)**函数，通过第一步函数返回的r用户-电影-评分列表，生成两个字典数据：user\_rate和user\_movie。具体的内容为： user\_rate[用户id]=[(电影id,电影评分)...]； movie\_user[电影id]=[用户id1,用户id2...]。

第三步定义**get\_movie\_info()**函数，这个函数和之前的算法类似，从movies.csv文件中读取电影信息（movieId，title，genre），将相关信息保存在movie\_info和movie\_ID之中，最后也返回这两个对象。

第四步定义**TF\_IDF(movie\_info)**函数，在这一步我们可以简化计算步骤，使用scikit-learn库中TfidfVectorizer函数。这个函数可以将文本集合转换成我们所需要的TF-IDF特征矩阵，相对于我们自己编写的多重for循环（所需时间比较久），时间复杂度减少了许多（改进后缩短至几秒以内）。相关函数如下图所示：



**构造TF-IDF特征矩阵**

图5.10 TF\_IDF(movie\_info)函数具体实现

第五步定义recommendation(user\_rate, user\_id, movie\_ID, cos\_sim, user\_movie)函数，这里面的参数前面函数都已经提及过这里就不再赘述。这个函数可以根据这些参数，为这个用户生成推荐电影列表。并且根据用户的评分和电影类别的相似度来计算预期评分并进行排序，这样方便主函数中给用户推荐电影。函数返回recommend\_list和recommend\_dict。

第六步定义predict\_score(user\_rate, user\_id, movie\_ID, cos\_sim, movieid)函数。这个函数根据传入的参数，以及用户的评分和电影类别的相似度来预测用户对某个电影的评分，最后返回pre\_score。得到用户的预测评分。

### 5.3.2 遇到的问题及解决方式

**问题一：基于内容的推荐算法运行速度相对于User-User的协同过滤算法较慢原因有哪些？**

比较明显的问题在于TF-IDF特征矩阵计算时，多个for循环极大的延缓算法的计算速度。我们可以通过使用TfidfVectorizer类，直接从movie-genre表中获得tf-idf特征矩阵。这个类所在的库经过了优化调优，并且利用了并行计算的能力，加快处理数据的速度。

### 5.3.3 实验测试与结果分析

**User-User的协同过滤算法测试集：**



图5.11 User-User的协同过滤算法测试集结果

**基于内容的推荐算法测试集：**

图5.12 基于内容的推荐算法测试集结果

对于相同的测试集，两种算法有截然不同的两种结果：不仅对电影的评分预测不同，算法结果的SSE值也不相同。从测试结果我们得知User-User的协同过滤算法测试集结果比较好（因为评分预测值差值并不大），可能原因有以下几条：其一是算法中可能忽略了电影类别的关系，基于内容的推荐算法更加关注于电影本身的内容特征；其二是可能是还需要更多用户对电影的类别选择的数据来改进最后的结果；其三是可能用户的观影选择还有更多的影响因素，也就是电影的类别并不能很好的概括用户的偏好导致推荐不能很好的满足用户的需求，最终导致基于内容的推荐算法的SSE值比较大。

通过输入被推荐者用户ID为5，需要推荐5个电影，在User-User的协同过滤算法只观察最好的5个邻居用户，我们得出以下的测试结果：

**User-User的协同过滤算法输入（5，5，5）：**

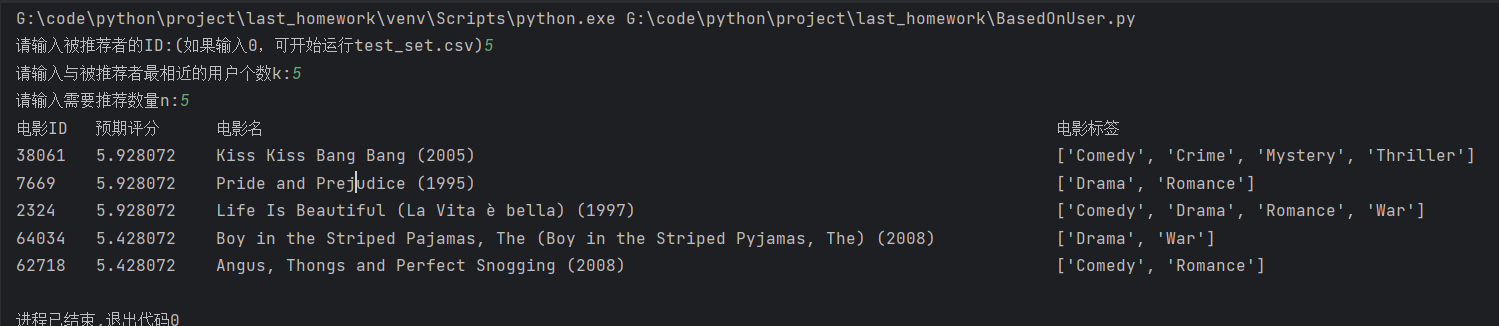


图5.13 User-User的协同过滤算法（用户ID 5，相邻5个邻居、推荐5个电影）

**基于内容的推荐算法输入（5，5，5）：**

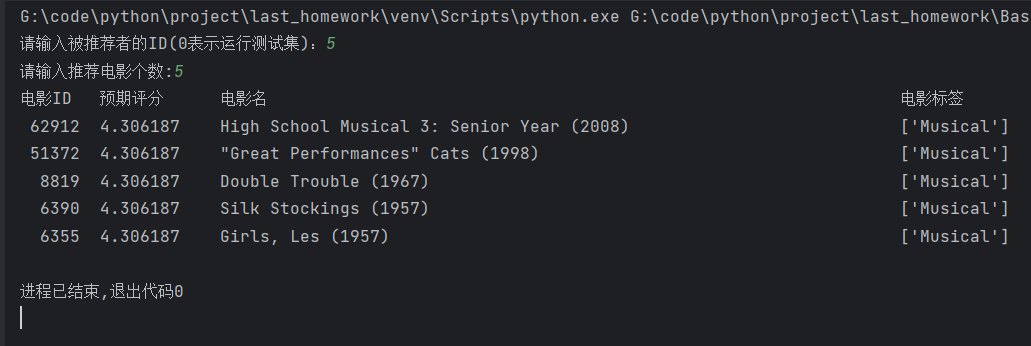


图5.14 基于内容的推荐算法（用户ID 5，推荐5个电影）

我们可以明显观察到基于内容的推荐算法结果在电影标签上是统一的‘Musical’。基于内容的推荐算法更加关注用户的个性化、偏好，考虑了用户对电影的喜好以及电影本身的类别，减少对用户评分的关注；User-User的协同过滤算法更加关注于用户评分以及邻居评分和相似系数等数据。

## 5.4 实验总结

本次实验我们主要使用User-User的协同过滤算法和基于内容的推荐算法，并且通过测试结果来比较衡量两者的相似之处与不同之处。从名字上我们很明显知道User-User的协同过滤算法是**从用户的角度（user）**，以用户评分以及用户的相似度来推荐以及预测电影；而基于内容的推荐算法是**从电影特征的角度（genre）**，来推荐以及预测特定用户的电影。由于角度的不同，二者更好解决的问题也不相同，前者可以来应对类似**user-movie的推荐问题**，比如网购平台中的“猜你想买”板块；后者可以用来应对类似**movie-movie的推荐问题**，比如网购平台在商品界面中的“类似商品推荐”板块。在计算推荐优先度的时候，我们需要用到一个比较重要的系数——相似度系数。前者需要使用皮尔逊相关系数来计算用户之间的相似度，后者需要使用TF-IDF以及余弦相似度来计算电影（genre）之间的相似度。从本次实验中，我了解到一些关于通过大数据来进行用户推荐的两种方法，并且不同方法之间有各自的优势区间。根据不同的大数据类型、使用场景以及希望的功能，我们需要灵活去选择不同的推荐算法。这也体现出对大量数据进行初步处理的重要性。