

**课 程 实 验 报 告**

**课程名称： 大数据分析**

**专业班级： CS1804**

**学 号： U201814596**

**姓 名： 左振**

**指导教师： 崔金华**

**报告日期： 2020.12.30**

**计算机科学与技术学院**

目录

[实验一 wordCount算法及其实现 4](#_Toc60305757)

[**1.1 实验目的** 4](#_Toc60305758)

[**1.2 实验内容** 4](#_Toc60305759)

[**1.3 实验过程** 5](#_Toc60305760)

[1.3.1 编程思路 5](#_Toc60305761)

[1.3.2 遇到的问题及解决方式 6](#_Toc60305762)

[1.3.3 实验测试与结果分析 6](#_Toc60305763)

[**1.4 实验总结** 7](#_Toc60305764)

[实验二 PageRank算法及其实现 8](#_Toc60305765)

[**2.1 实验目的** 8](#_Toc60305766)

[**2.2 实验内容** 8](#_Toc60305767)

[**2.3 实验过程** 8](#_Toc60305768)

[2.3.1 编程思路 8](#_Toc60305769)

[2.3.2 遇到的问题及解决方式 9](#_Toc60305770)

[2.3.3 实验测试与结果分析 9](#_Toc60305771)

[**2.4 实验总结** 9](#_Toc60305772)

[实验三 关系挖掘实验 10](#_Toc60305773)

[**3.1 实验内容** 10](#_Toc60305774)

[**3.2 实验过程** 10](#_Toc60305775)

[3.2.1 编程思路 10](#_Toc60305776)

[3.2.2 遇到的问题及解决方式 11](#_Toc60305777)

[3.2.3 实验测试与结果分析 11](#_Toc60305778)

[**3.3 实验总结** 13](#_Toc60305779)

[实验四 kmeans算法及其实现 14](#_Toc60305780)

[**4.1 实验目的** 14](#_Toc60305781)

[**4.2 实验内容** 14](#_Toc60305782)

[**4.3 实验过程** 15](#_Toc60305783)

[4.3.1 编程思路 15](#_Toc60305784)

[4.3.2 遇到的问题及解决方式 16](#_Toc60305785)

[4.3.3 实验测试与结果分析 16](#_Toc60305786)

[**4.4 实验总结** 16](#_Toc60305787)

[实验五 推荐系统算法及其实现 17](#_Toc60305788)

[**1.1 实验目的** 17](#_Toc60305789)

[**1.2 实验内容** 17](#_Toc60305790)

[**1.3 实验过程** 19](#_Toc60305791)

[1.3.1 编程思路 19](#_Toc60305792)

[1.3.2 遇到的问题及解决方式 20](#_Toc60305793)

[1.3.3 实验测试与结果分析 20](#_Toc60305794)

[**1.4 实验总结** 24](#_Toc60305795)

# 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

总思路如图1-1所示：

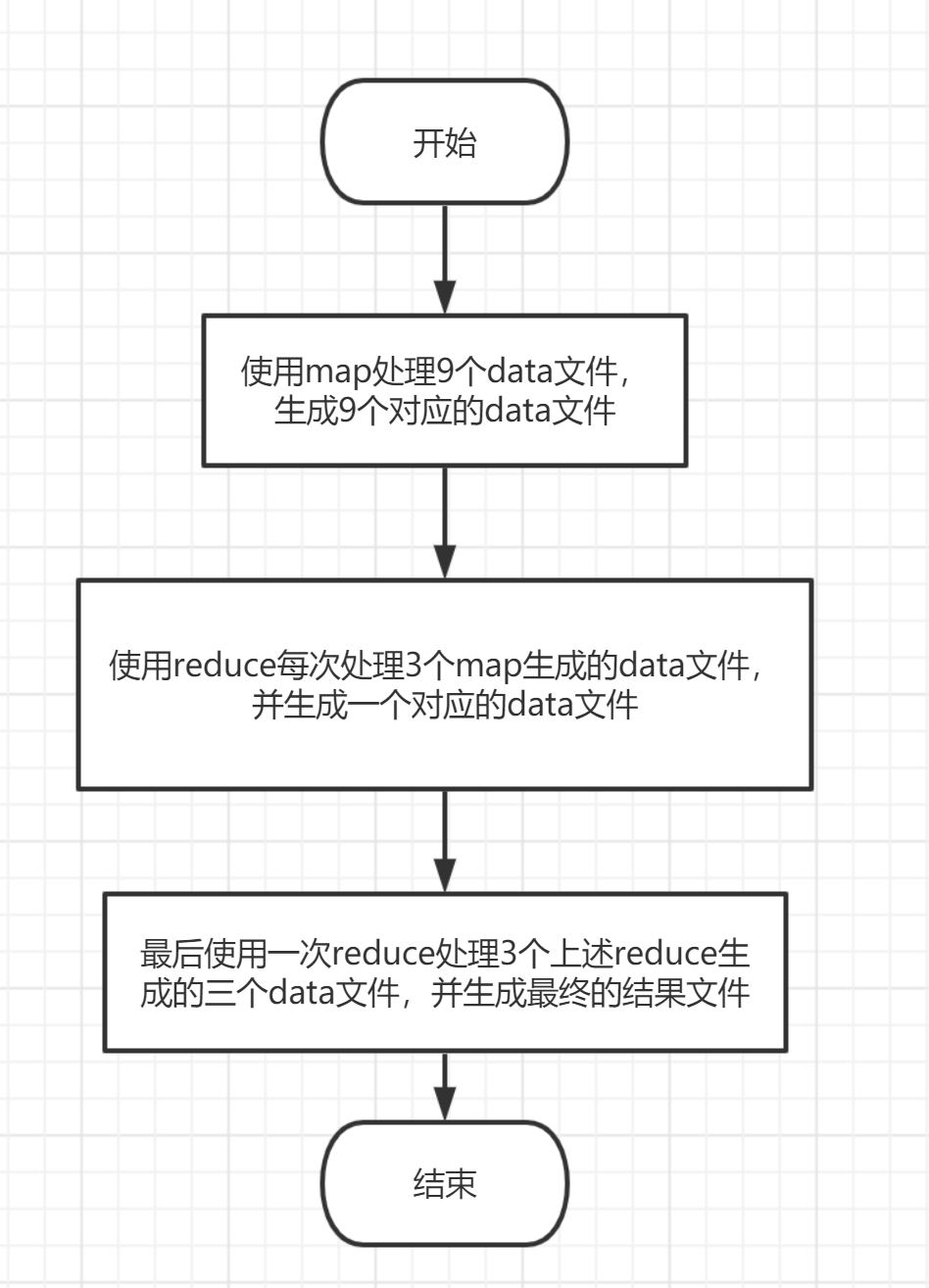


图1-1：总体逻辑实现

（1）map：

1）编写面向一个data文件的map函数，具体实现逻辑就是对出现过的单词进行记录，生成形如“单词，1”的键值对，然后存入另一个data文件；

2）主函数里开9个线程，同时处理9个data文件，并生成9个map处理后的文件。

（2）reduce：

1）编写面向三个data文件的reduce函数，具体实现逻辑就是对三个文件里的单词进行出现次数统计，生成形如“单词，出现次数”的键值对，然后存入一个data文件；

2）主函数里先开三个线程分别处理9个map生成的文件，同时会生成3个经过reduce处理的文件，最后等待三个线程处理结束后，将刚刚reduce生成的3个文件作为参数再次进行一次reduce操作，生成最终的结果文件。

### 1.3.2 遇到的问题及解决方式

（1）map：

1）文件处理时的细节：由于我是按照特定的字符切割行来读取的文件流，但是文件的最后一行与其他行相比少了一个换行符，由此会出现一个单词统计的错误，解决方案就是在做文件解析之前先将文件里写入一个”\n”，保证所有行之间的一致性，这样就能避免此错误；

2）线程的实现：采用的是threading包里的Thread方法来实现多个线程。

（2）reduce：

1）map文件处理与reduce文件处理的一致性：由于经过map处理后的文件键值对的第二项均为1，reduce处理后的文件键值对的第二项为单词出现次数，原reduce函数实现时记录单词出现次数的方式肯定不能再用了，改为加上键值对第二项的数值这一逻辑后即解决了问题；

2）线程等待的实现：使用join()方法实现主函数的阻塞，在三个线程结束之后才开始最后结果文件的生成。

### 1.3.3 实验测试与结果分析

（1）经过map处理后生成的data文件如下图1-2，图1-3所示：

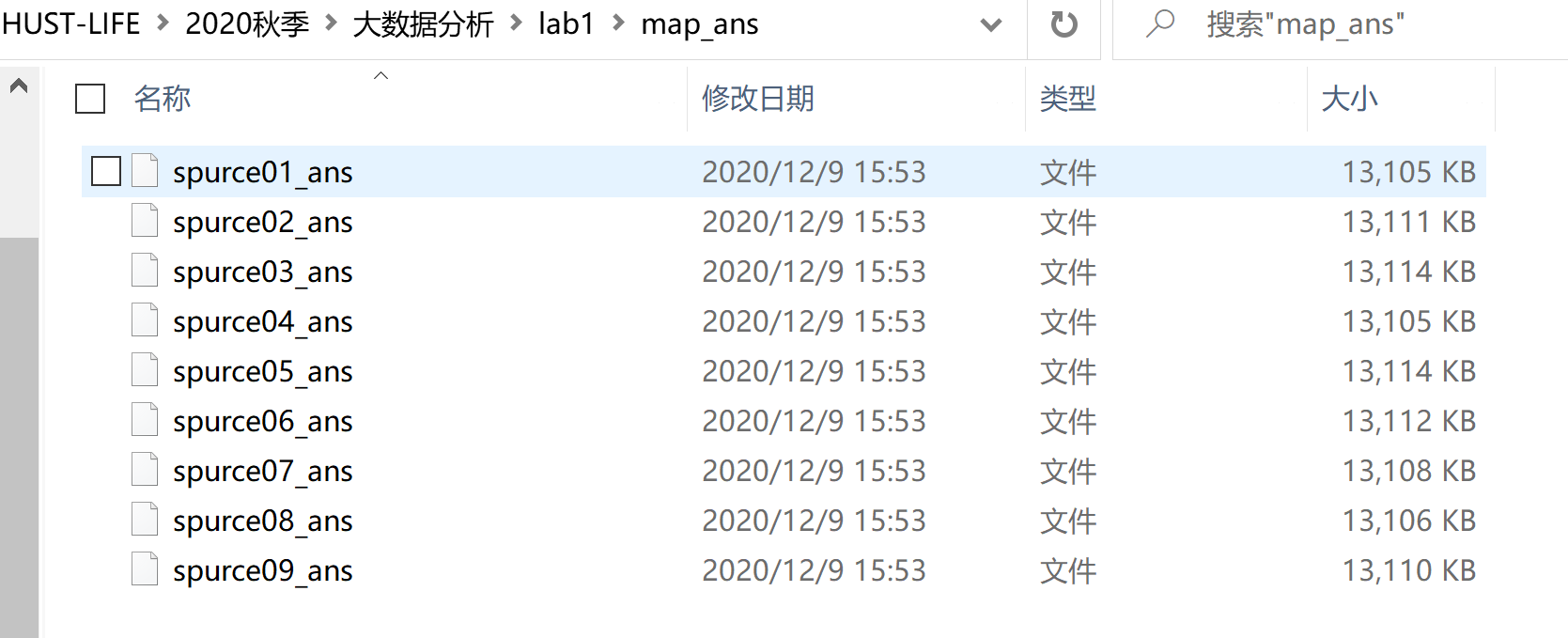


图1-2 map处理后生成的文件

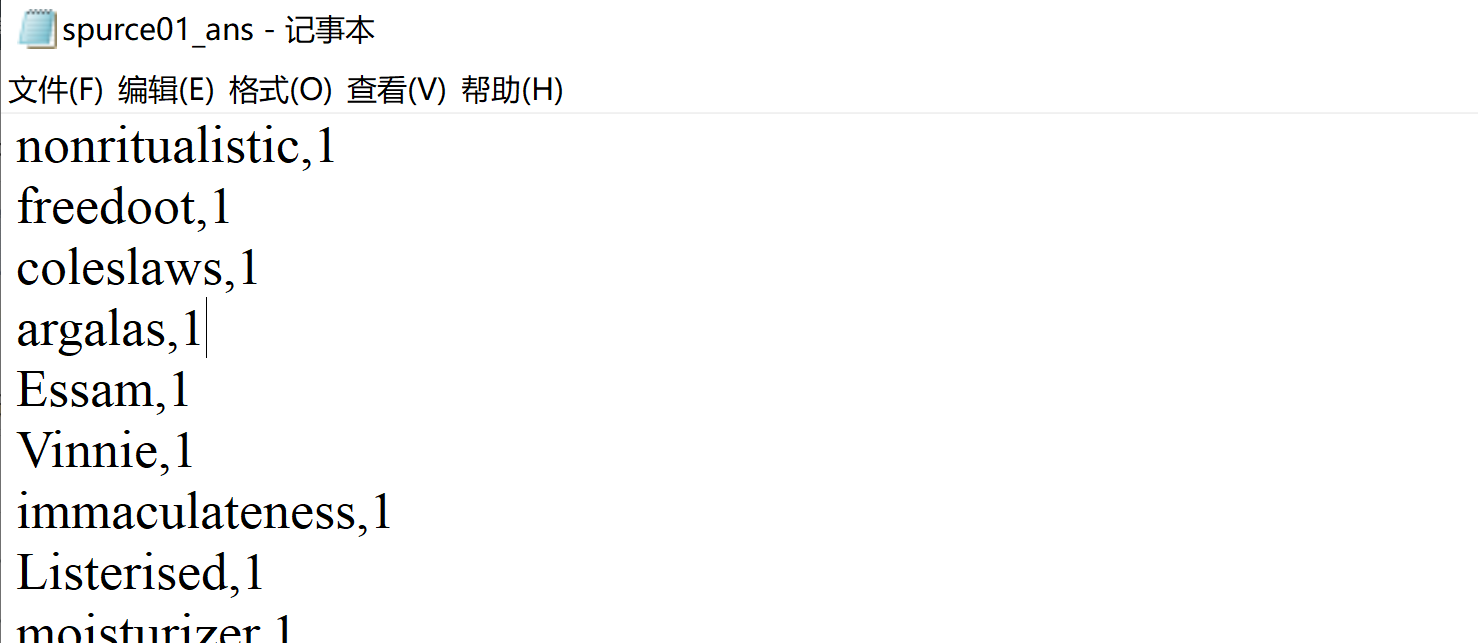


图1-3：spurce01\_ans文件部分内容

（2）经过reduce处理后生成的文件如下图1-4，1-5所示：

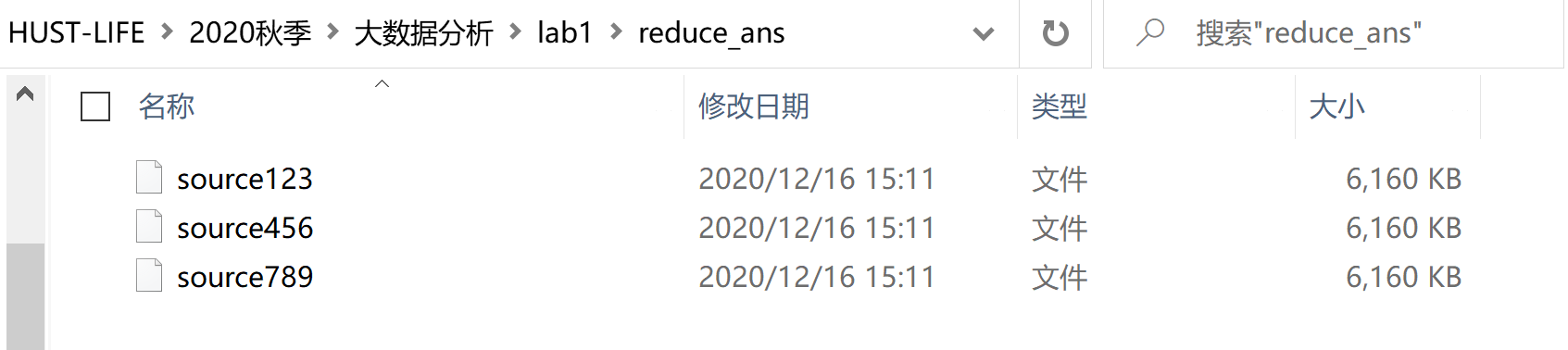


图1-4：reduce处理后生成的文件



图1-5：source123文件部分内容

（3）将三个reduce处理后生成的文件再进行一次reduce处理后生成的最终结果文件，如下图1-6所示：

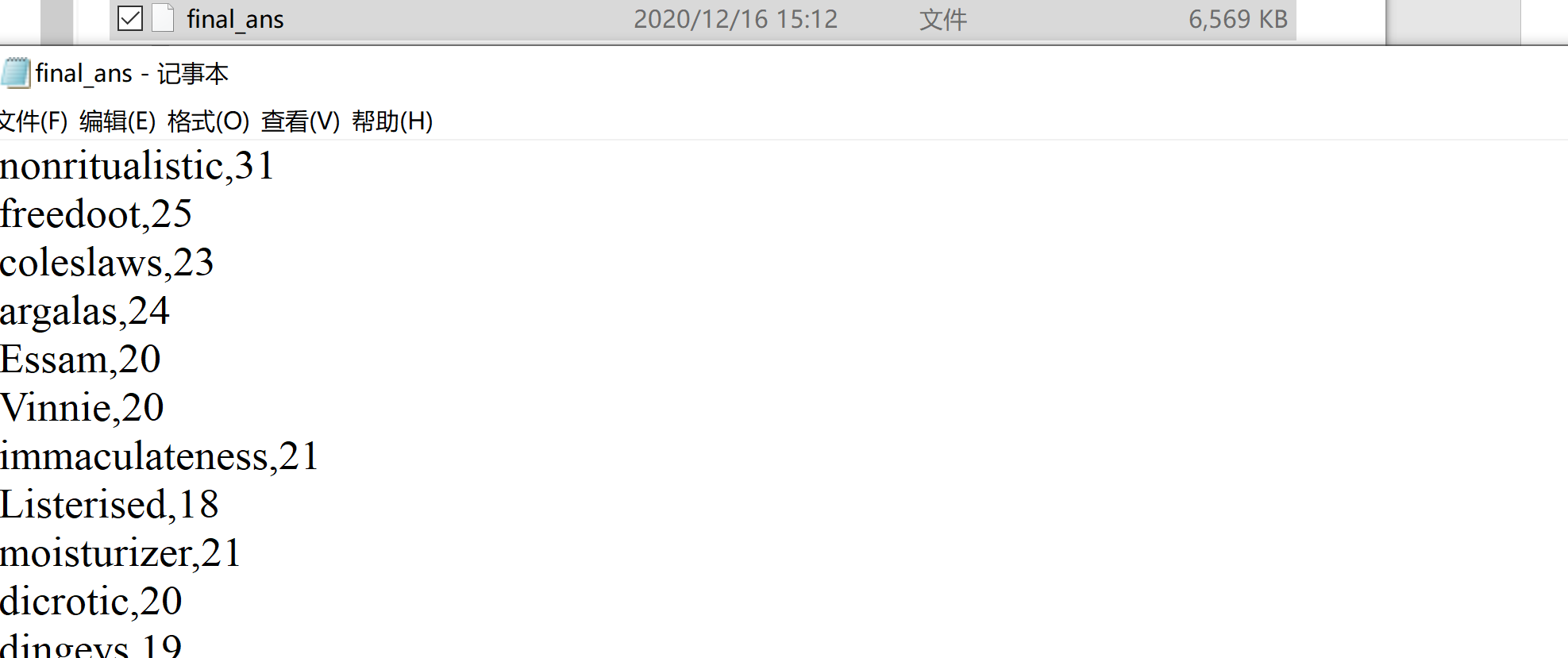


图1-6：final\_ans文件部分内容

## **1.4 实验总结**

通过实验一更好的理解了mapreduce的核心思想，也是一次对于线程操作的实践。

# 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]](#endnote-1)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

（1）对csv文件进行解析，生成一个阿拉伯数字到节点的映射，一个边集用来表示节点的关联关系；

（2）通过边集与映射关系生成一个转移矩阵；

（3）通过计算公式进行迭代计算，当误差小于0.00000001时，迭代结束，公式为：r，为阻尼系数，值设为0.85

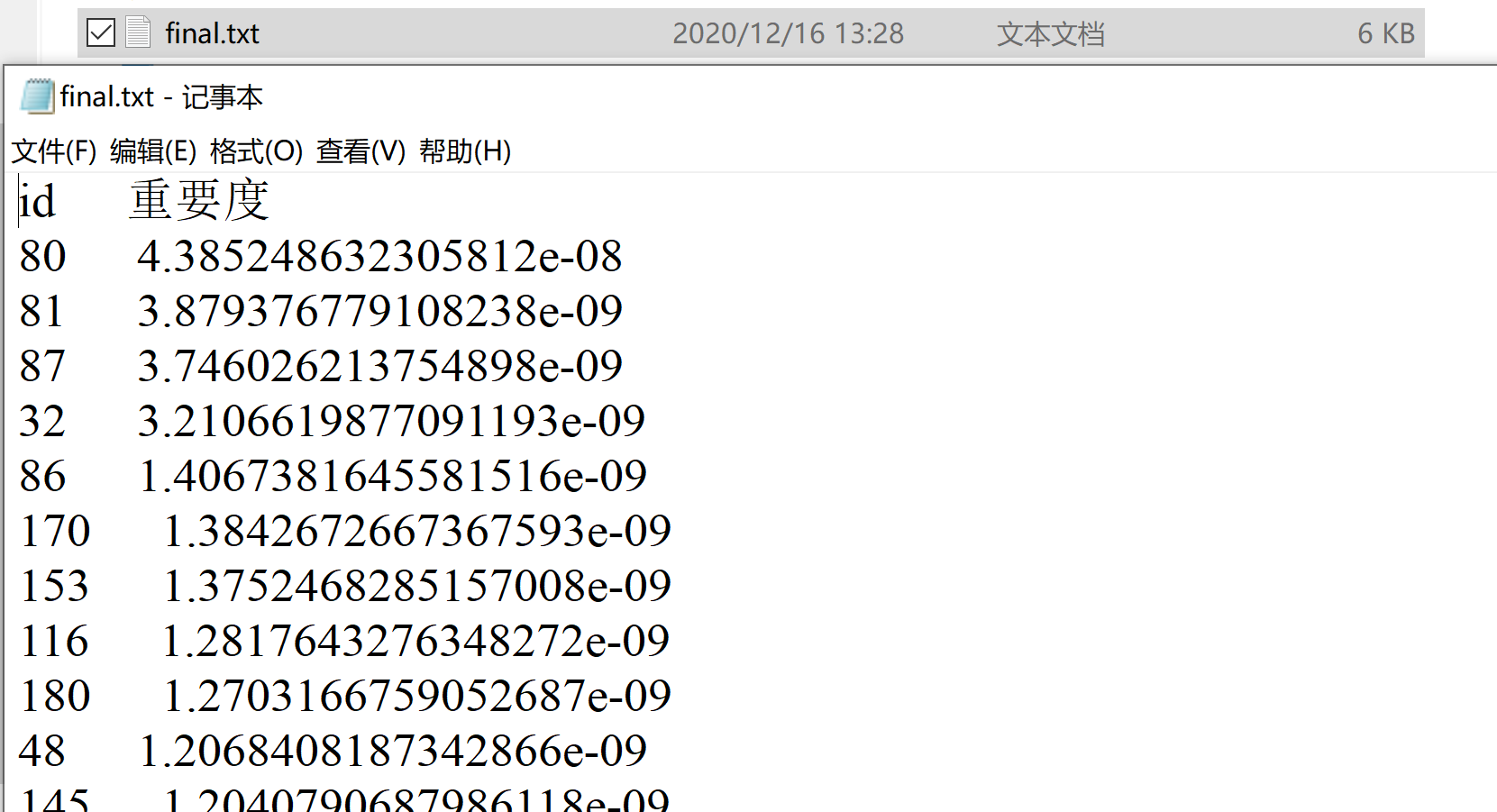
（4）根据映射关系将迭代后的结果转化成一个结果字典，最后对字典按照推荐度从大到小排序，最后写入文件。

### 2.3.2 遇到的问题及解决方式

（1）文件处理，即转移矩阵初始化问题：由于给的原始文件中，sender与receiver的id是没有规律的，所以就想着通过一个映射关系来简化操作，以便生成原始的转移矩阵，具体的映射方式就是按照节点在csv文件里的出现次序来赋值；

### 2.3.3 实验测试与结果分析

（1）结果文件如下图所示：



图：final.txt文件部分内容

## **2.4 实验总结**

PageRank实验是最简单的一个实验，只要将初始的转移矩阵构造出来，后面要处理的东西就势如破竹，直接套公式迭代即可，重要的还是拓展了视野，了解了网页排名算法的大致思想。

# 实验三 关系挖掘实验

## **3.1 实验内容**

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

## **3.2 实验过程**

### 3.2.1 编程思路

（1）文件处理：将文件解析成一个列表，列表里的元素也为列表，列表里包含各个单词字符串；

（2）编写创建1项候选集函数：即将文件解析后的列表解析为单项集；

（3）编写扫描数据集函数：该函数根据设置的最小支持度生成频繁项集以及候选项集的支持度字典；

（4）编写利用频繁项集构建候选项集函数：该函数即基于k阶频繁项集生成k+1阶候选项集；

（5）编写apriori主函数，循环处理三次生成1，2，3阶频繁项集，函数逻辑如下图3-1所示：

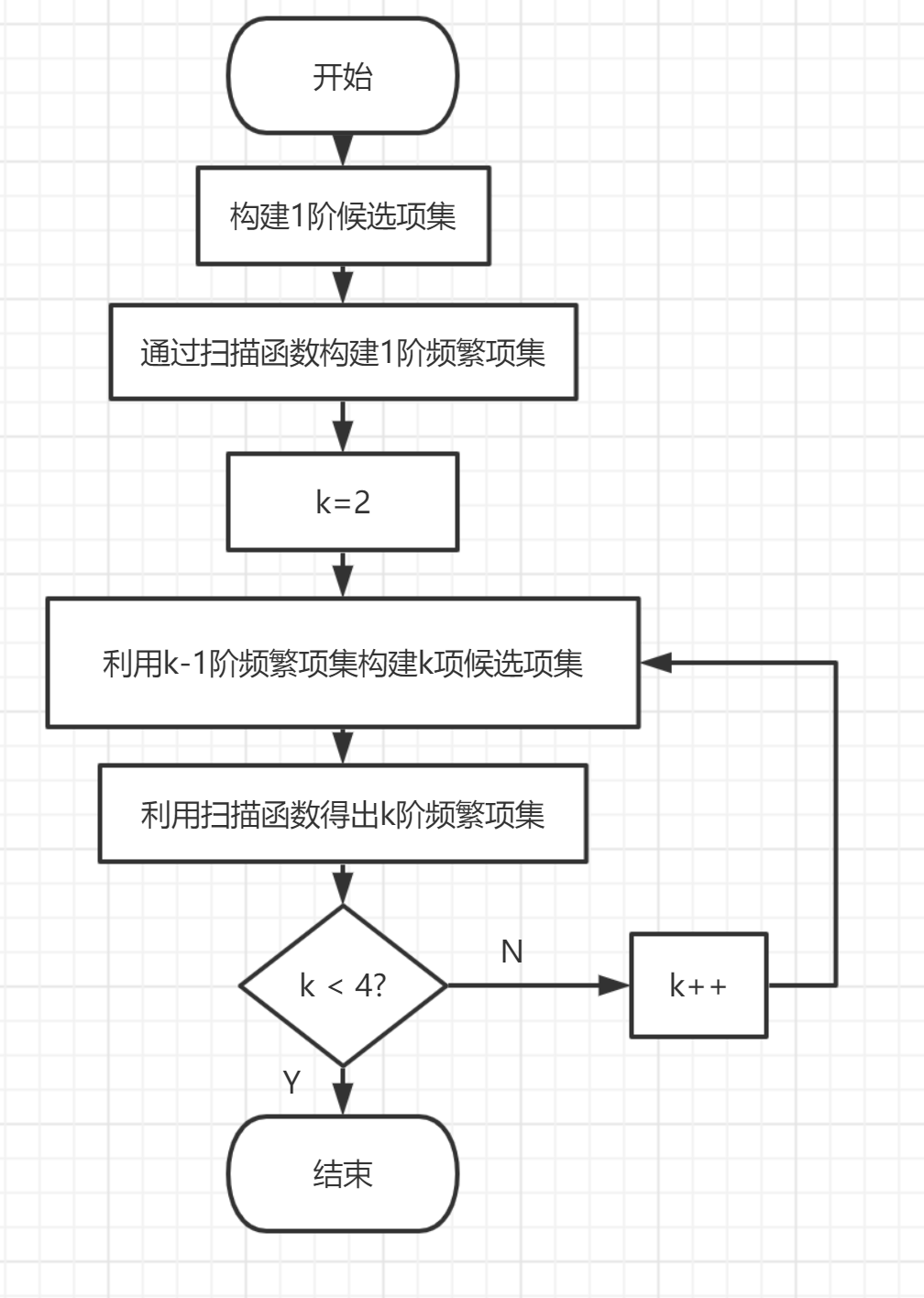


图3-1：apriori函数逻辑

（6）利用最小置信度0.5来遍历频繁项集生成关联规则。

### 3.2.2 遇到的问题及解决方式

（1）关联规则算法的实现：首先直观的想法就是生成各个映射对来计算置信度，但是这样的工作量比较大并且逻辑容易紊乱，最后我发现如果按照阶的顺序来处理并且将每一阶的频繁项集按序排列后，会形成之前处理过的对是后面即将要处理的对的子集的特殊情形，这样就大大简化了计算的逻辑。

### 3.2.3 实验测试与结果分析

（1）结果文件如图3-2，3-3，3-4，3-5所示：

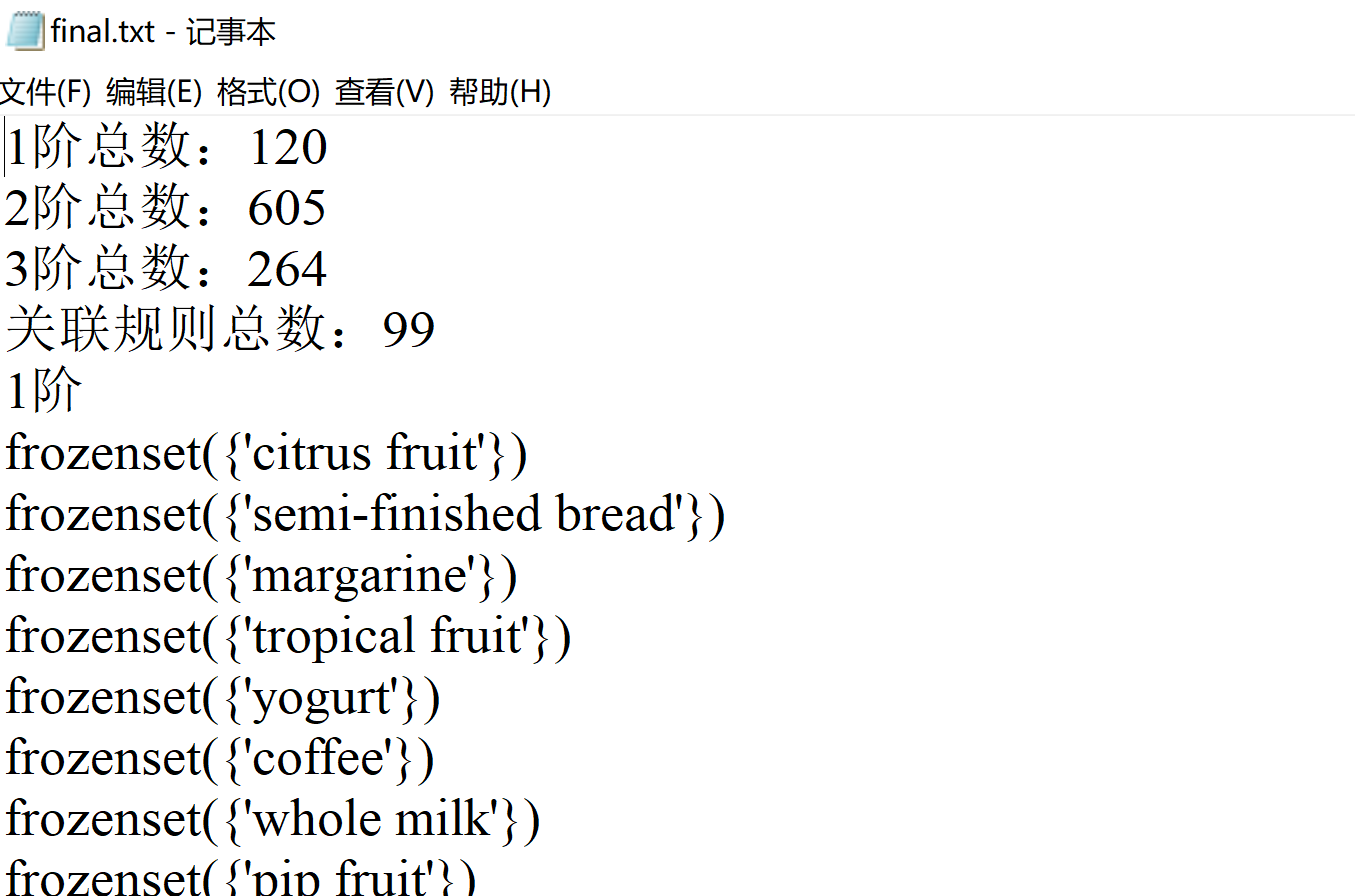


图3-2：结果文件（1）

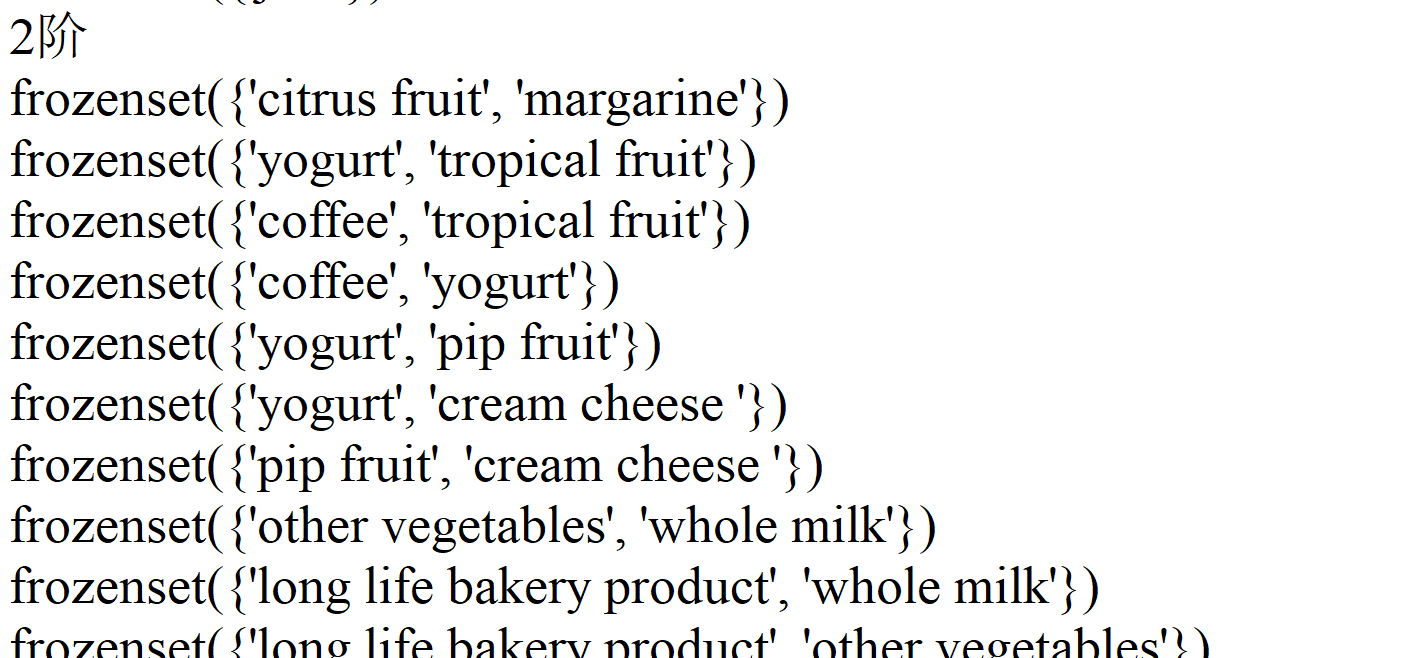


图3-3：结果文件（2）

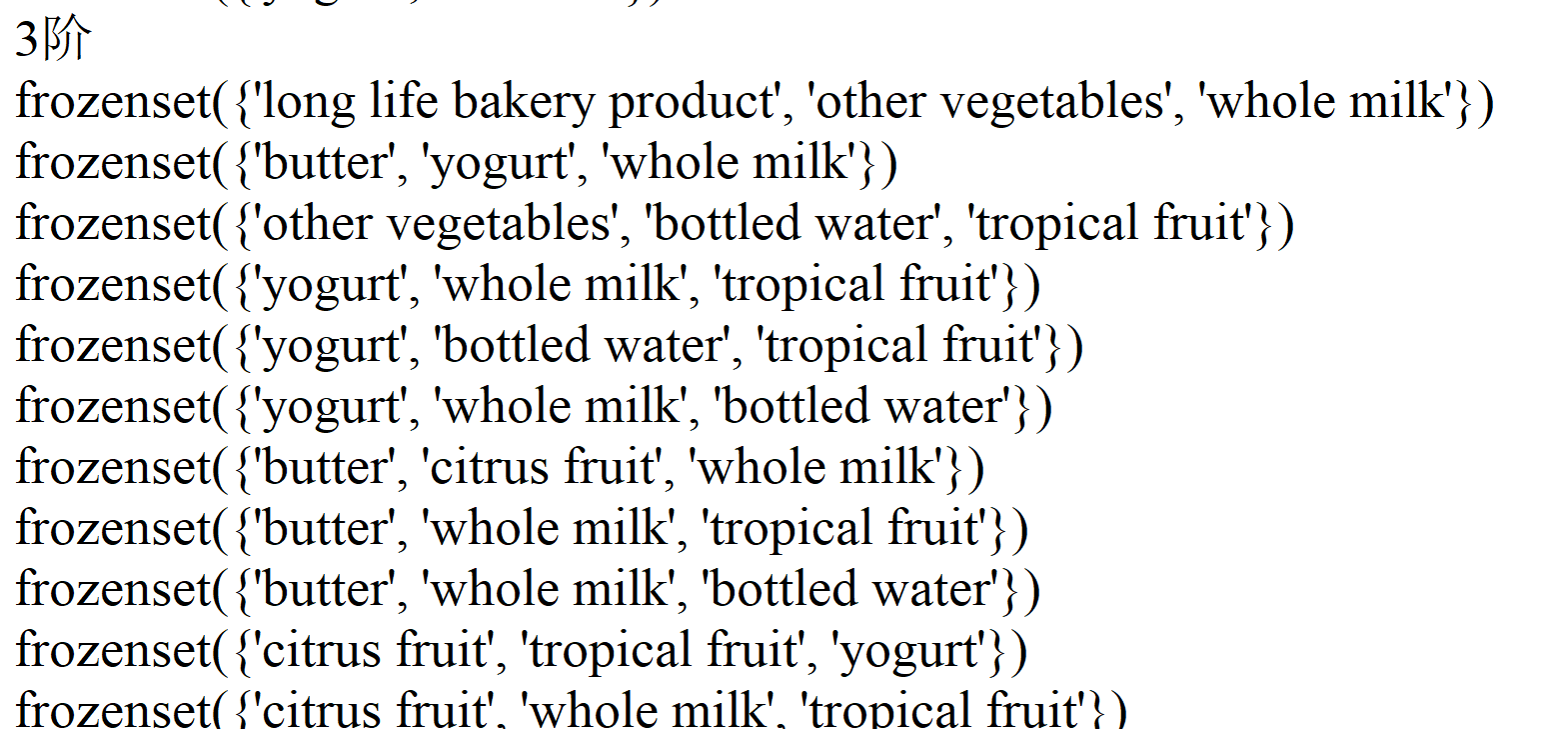


图3-4：结果文件（3）

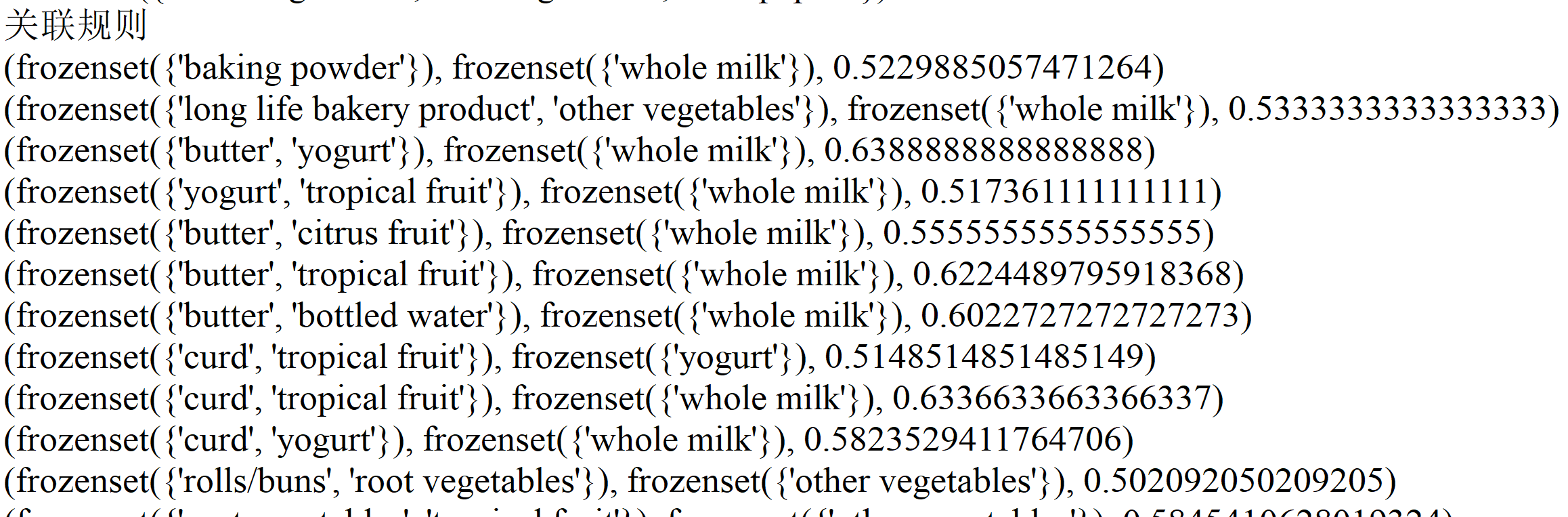


图3-5：结果文件（4）

（2）结果分析：

由上图可知：1阶频繁项集：120个；2阶频繁项集：605个；3阶频繁项集：264个；关联规则总数：99个。

## **3.3 实验总结**

实验三应该是几个实验里最不好做的一个，主要在于数据处理以及一些关键点的理解上面，好在网上开源资料丰富，最后顺利的实现了要求。

# 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,探究聚类算法院里；

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所示。



图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

（1）主要逻辑如下图4-1所示：

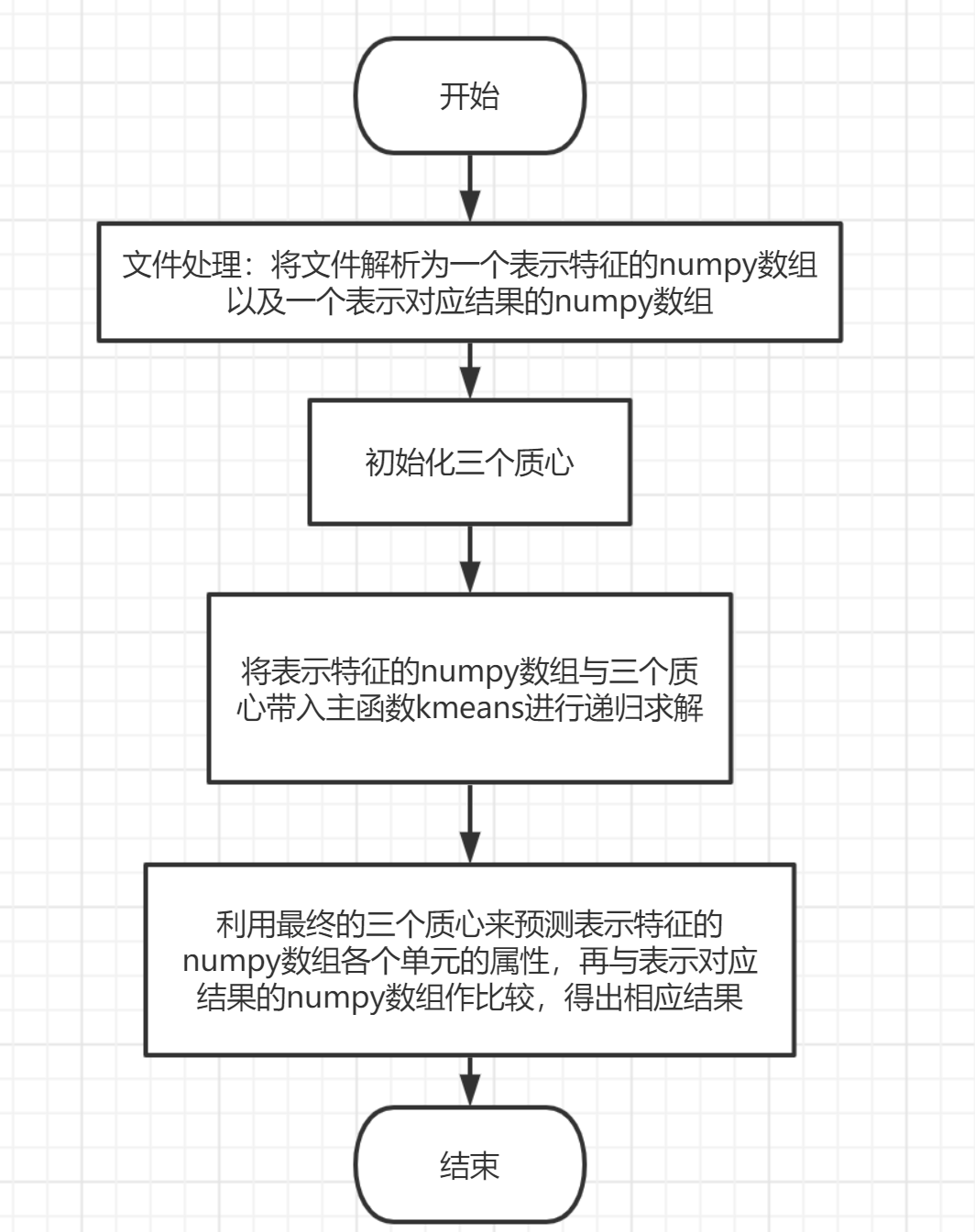


图4-1：主要逻辑

（2）kmeans函数逻辑如下图4-2所示：

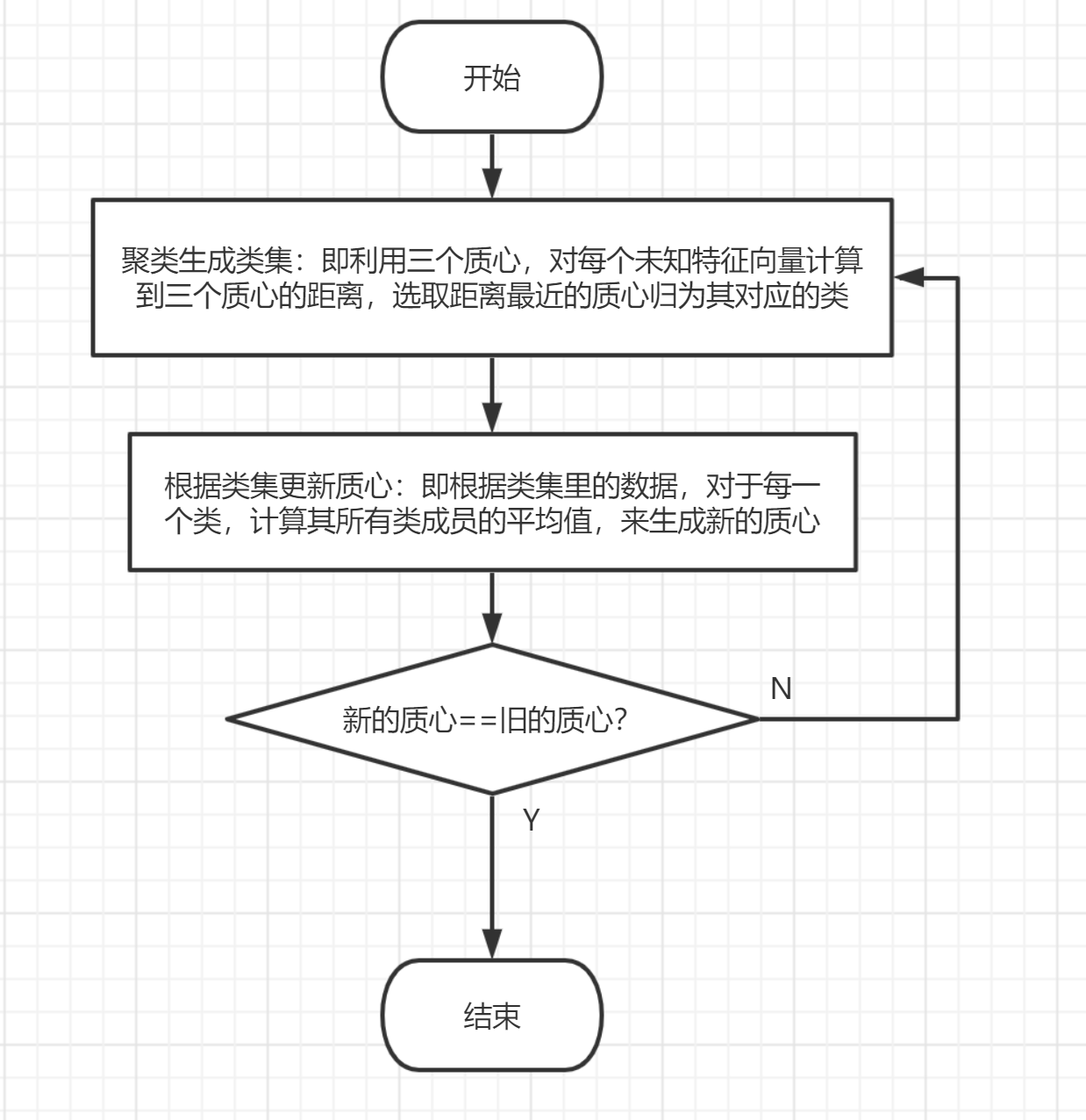


图4-2：kmeans函数逻辑

### 4.3.2 遇到的问题及解决方式

（1）kmeans函数递归爆栈：由于我是按照递归的方式来进行迭代，所以在跑了1000多次之后就出现了栈空间不足的问题，解决方案一就是设置一个迭代次数，在迭代次数超过某个数之后就来到递归边界，然后我发现在不同的迭代次数设置下正确率总会到达一个阈值，所以我觉得应该要换一个思路，于是就换了一个递归边界的方式。原来我使用的递归边界是类集里同样的类到质心的距离平方和不变，可能是精确度的问题或者是编码中的bug导致一直达不到递归边界。学长告诉我正常迭代100多次就能得出结果，所以我将递归边界换成了比较质心更新后与更新前的值，如果相等就直接退出，结果就没遇到这个爆栈的问题，但是正确率不尽人意；

（2）聚类后属性的认定：在进行迭代之后虽然分成了三个簇，但是它们的属性都是未知的，于是就按照簇内原属性占比最大的属性的方法来定义该簇属性。

### 4.3.3 实验测试与结果分析

（1）结果如图4-3所示：

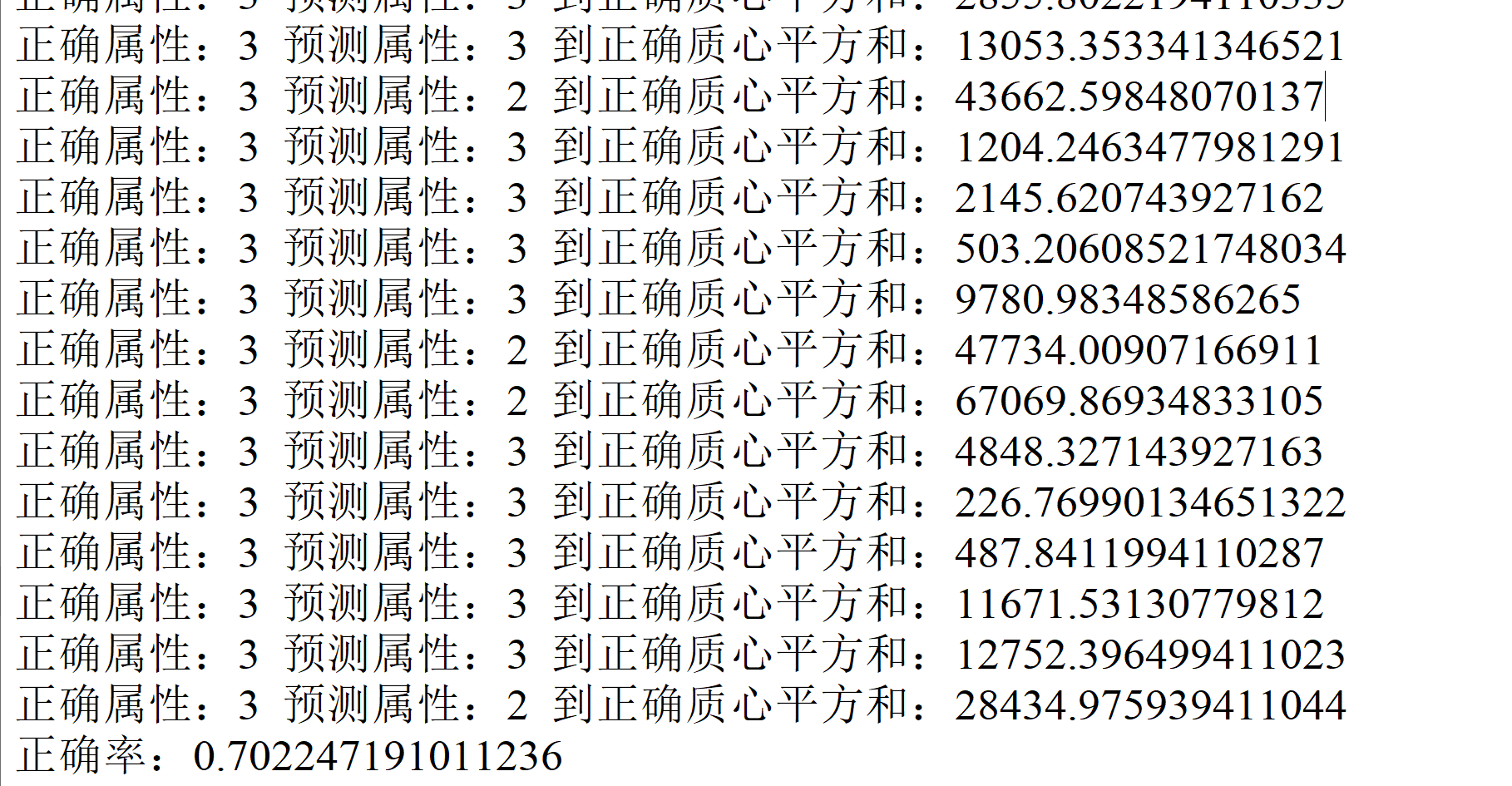


图4-3：final.txt文件部分内容

## **4.4 实验总结**

Kmeans算法在之前机器学习课程中已经学习过了，所以再次上手没有觉得有太大障碍。可能是由于写的有点仓促以及python中一些数值转化时的精度影响，最后的正确率只有百分之七十多，不是很理想。

# 实验五 推荐系统算法及其实现

## **1.1 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**

## **1.2 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。**

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**。

**选做（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。**

## **1.3 实验过程**

### 1.3.1 编程思路

1.基于用户的协同过滤推荐算法

（1）处理训练集中的评分数据，忽略文件中的观看时间这一列，构造用户-电影效用矩阵；

（2）编写pearson相似度计算逻辑，主要包括评分向量的构建以及计算公式的实现，评分向量的构建是找出两个user电影列表里的相同电影来实现，计算公式为：；

（3）编写预测评分逻辑，对于特定user，找出user列表中与其相似度最高的k个user，然后在这k个user中找出目标电影的评分，计算平均值；

（4）编写测试函数，解析测试文件，将userId和movieId带入预测评分函数，得出的结果与正确结果进行误差分析，最后统计SSE即可。

2.基于内容推荐

首先要得到tf-idf矩阵，创建一个1925\*19的矩阵，代表一共19种电影，一种1925个电影。第j列的idf值，为：电影的总数(1925)/该类下的电影数，第i行第j列的tf值为：1/该第i个电影拥有的电影种类数目。

得到tf-idf矩阵后可以根据这个矩阵得到相似度矩阵：

首先是根据余弦相似度的计算方法得到电影之间的相似度矩阵，余弦相似度的计算方法为，对任意两行i,j运用该公式就可以得到一个相似度矩阵中(i,j)处的值：



然后是根据minhash计算电影之间的相似度矩阵。首先得到01矩阵，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，根据01矩阵。为了将维数降到1925×5，自定义5个hash函数，这五个hash函数可以将行数打乱：h1(x) = (x+1)mod19, h2(x) = (2x+1)mod19, h3(x) = (3x+1)mod19, h4(x) = (4x+1)mod19, h5(x) = (5x+1)mod19。根据hash函数得到签名矩阵，该签名矩阵的维数是1925\*5，1925行中任意两行i,j之间进行jarcard操作，得到相似度矩阵中(i,j)处的值，遍历该1925行，得到相似度矩阵的所有值。

得到相似度矩阵(余弦相似度或minhash)后，要预测某movieId的分数，先得到用户已打分的电影Id，得到与movieId的相似度，根据任务书中的score计算方法，得到该movieId的预测分数。

最后打印不同userId的movieId的预测分数，将预测分数与真实分数求出SSE。

### 1.3.2 遇到的问题及解决方式

1.基于用户的协同过滤推荐算法

（1）文件解析问题：给定的训练集中包含timestamp这一项，最后认定为无关项，直接在解析文件时忽略掉即可；

（2）特殊测试案例问题：当k值取得不是很大时，存在某一个测试集里的用户-电影对，与该用户相似度最高的k个用户里都没有看过这个电影，所以无法预测其评分，我的处理方式是显式的告知该预测评分缺失，直接不纳入最后的SSE统计，当然最后肯定要统计一些可预测的数目；另一种方式就是找到满足所有测试集可预测的最小k值，经过不断测试，发现k值为66。

2.基于内容的推荐算法

最大的问题是理解tf-idf的意思，本题中的文本文档比较特殊，是一个电影Id，而单词是电影的类别。这点是问了助教的。

第二个问题是计算相似度矩阵时用时过多，最初是通过遍历所有行得到相似度矩阵中的值的，然后改进了算法，利用得到的tf-idf矩阵，先对每一个元素除以该元素所在行的模，得到新的if-idf矩阵，将这个矩阵×矩阵的转置，正好得到相似度矩阵，通过实验，处理后的if-idf矩阵的行×if-idf矩阵转置后的列正好是余弦相似度的计算公式。

其他也有问题但是都很快解决了。

### 1.3.3 实验测试与结果分析

1.基于用户的协同过滤推荐算法

（1）k值取6时，测试结果文件如下图所示：

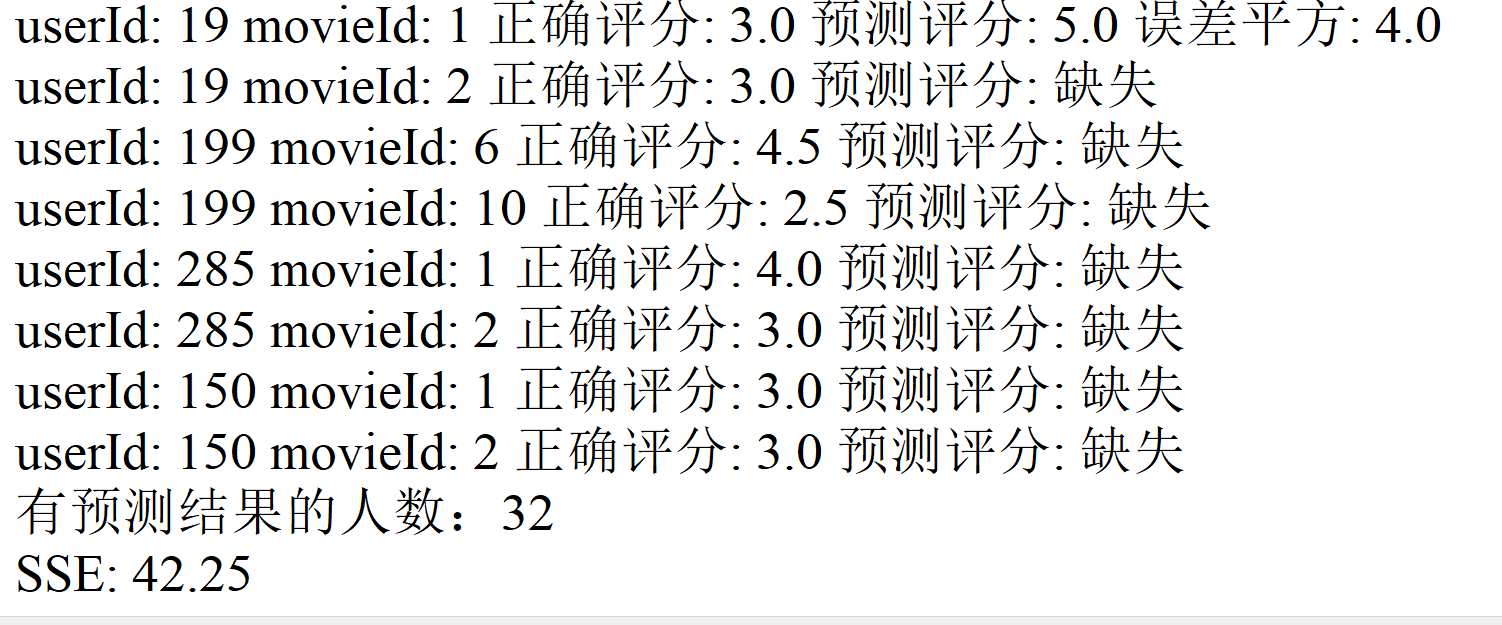


图5-1 k=6测试结果

（2）k值取20时，测试结果文件如下图所示：

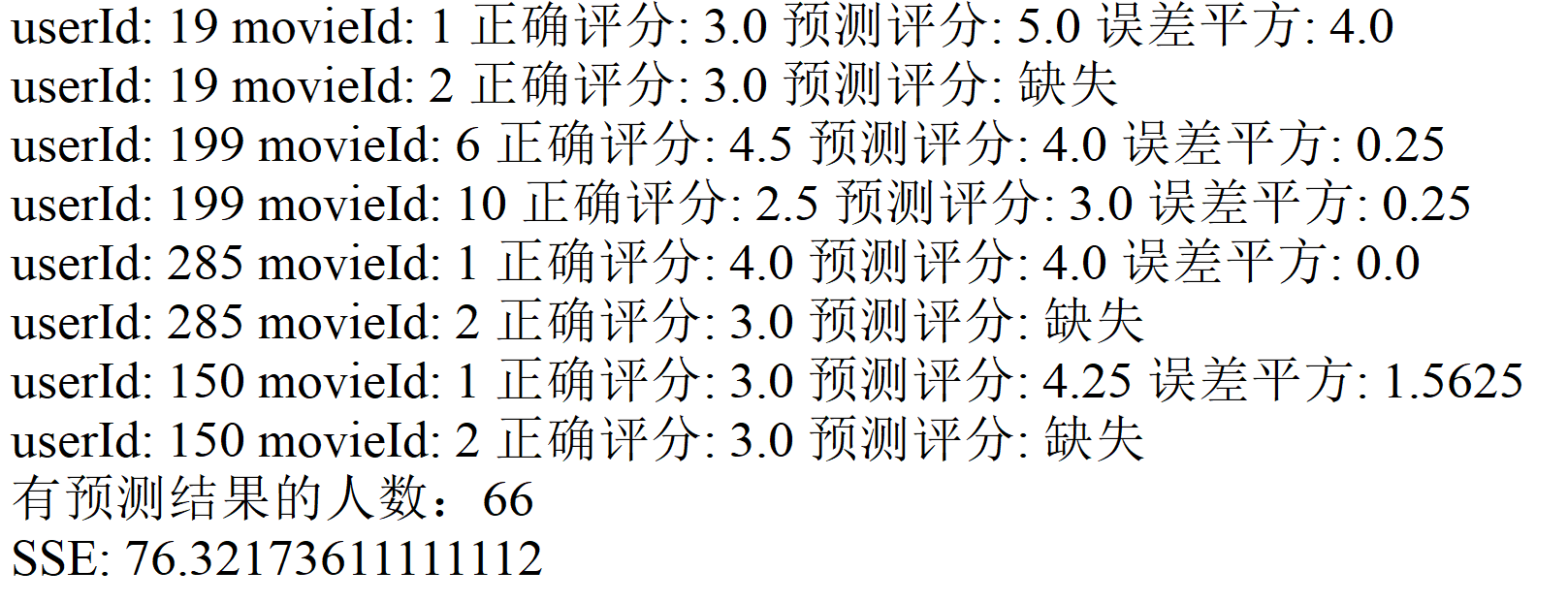


图5-2 k=20测试结果

（3）k取40时，测试结果文件如下图所示：

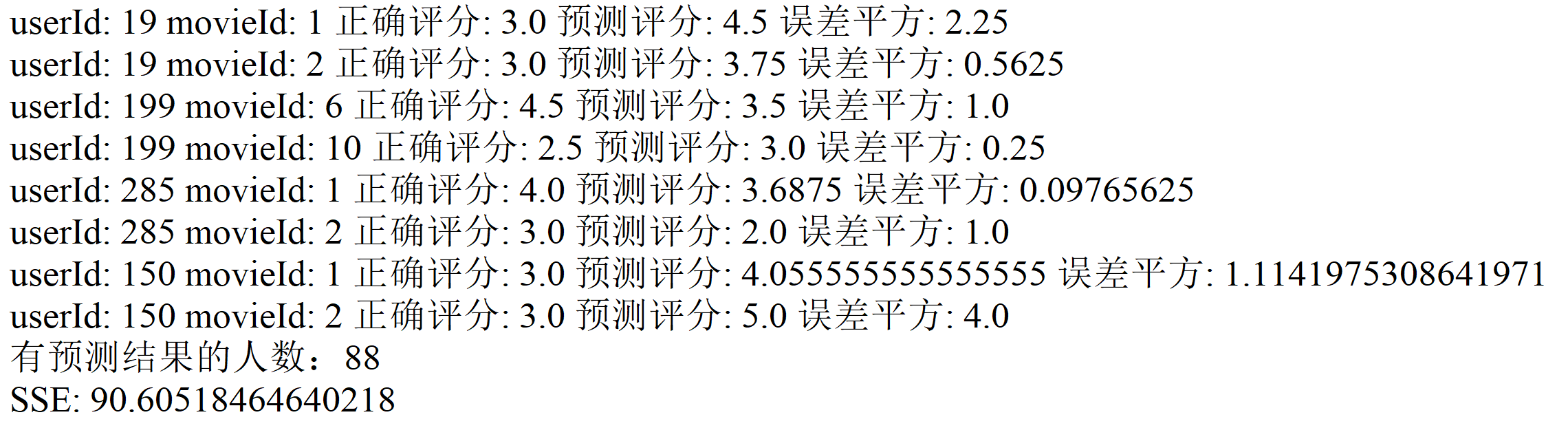


图5-3 k=40测试结果

（4）k取66时，测试结果文件如下图所示：

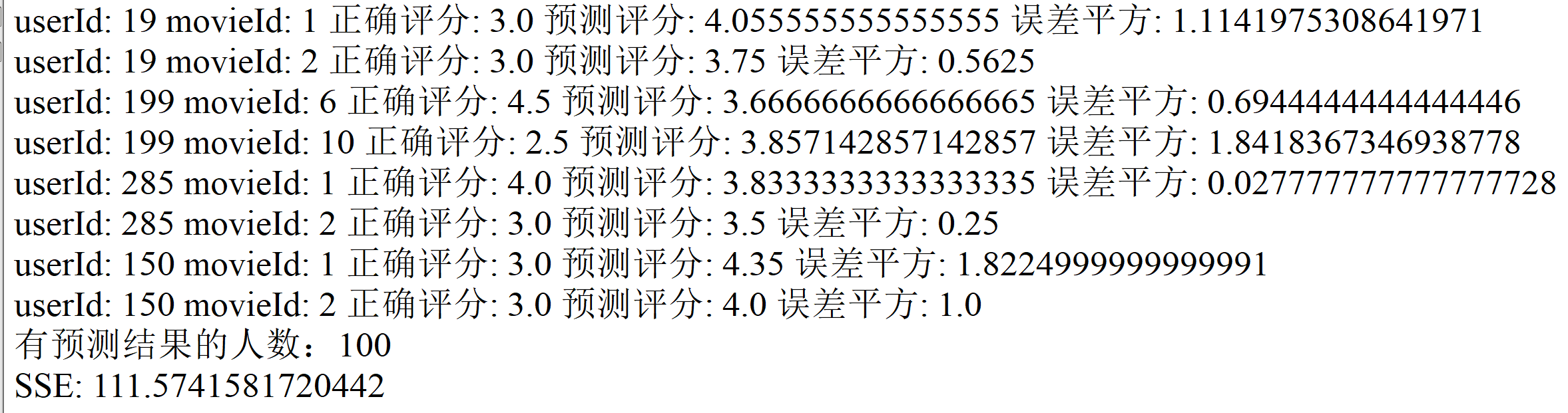


图5-4 k=66测试结果

（5）结果分析：

1）对上述测试的综合分析，k值与有预测结果人数的关系，如下图所示：

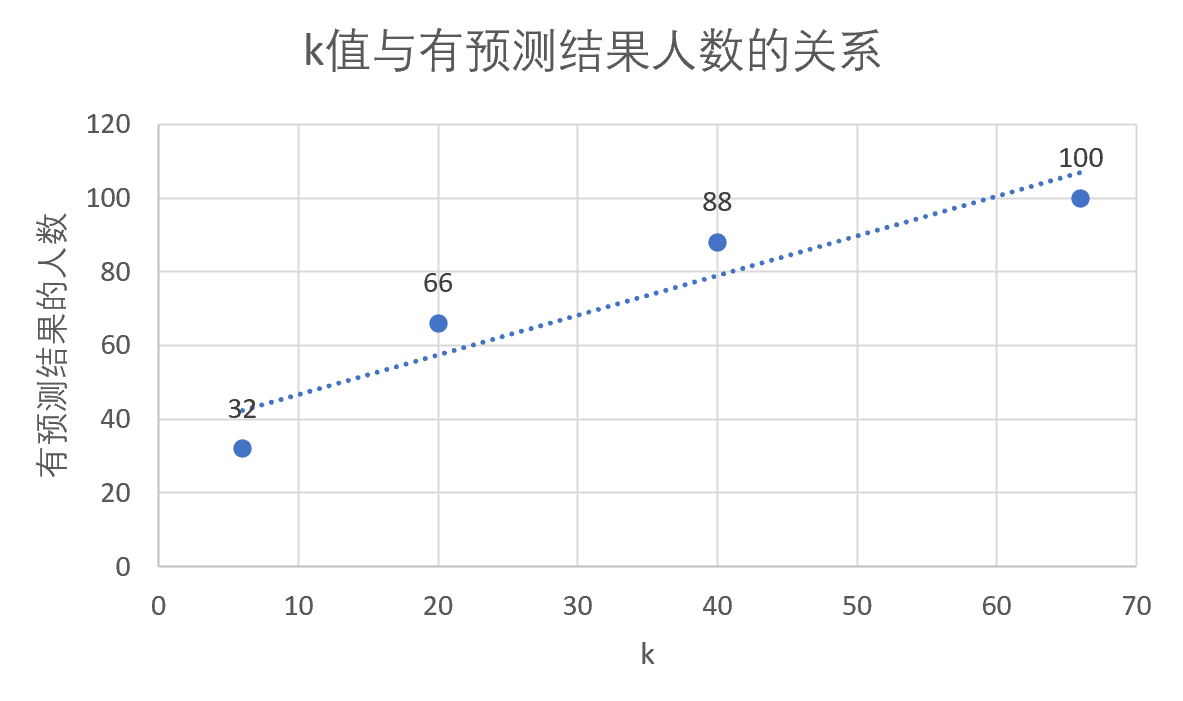


图5-5 k值与有预测结果人数的关系

由上图可知，k值越大，有预测结果的人数越多，人数最大为100（测试集人数为100）。从概率上来讲，相似的人越多对于某一电影是否被人看过来讲概率也会更高，经过测试，当k达到66时，测试集里的所有用户-电影对均有测试结果。

2）对上述测试的综合分析，k值与平均误差的关系，如下图所示：

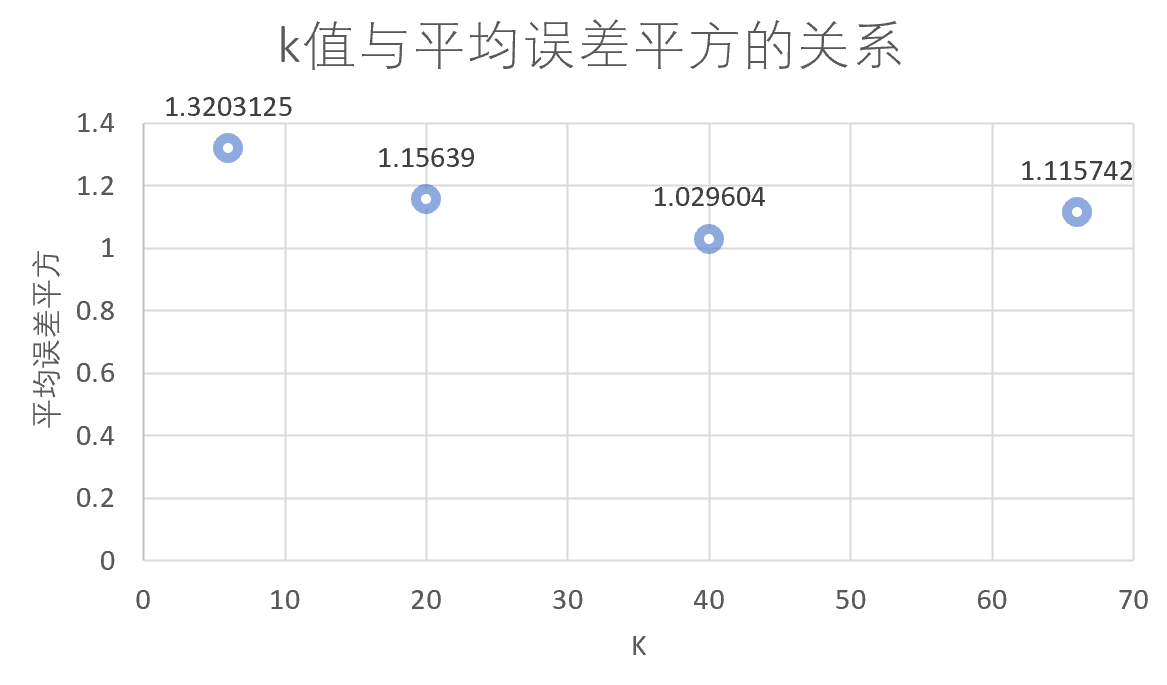
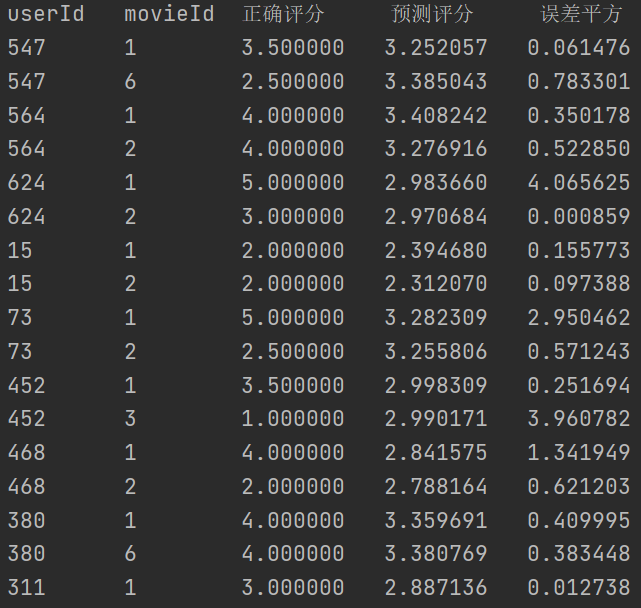


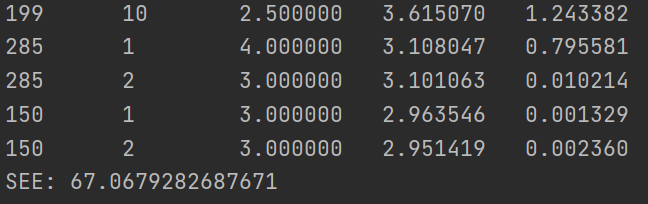
图5-6 k值与平均误差的关系

由上图可知，平均误差先减小，再增大；先减小的原因应该是样本具有一定规模后具备了普遍性，k值取得太小样本量较小，不利于预测；再增大的原因应该是当k不断增大后，一些原本跟目标用户相似度并不是那么高的用户也纳入了预测，会对预测产生干扰。

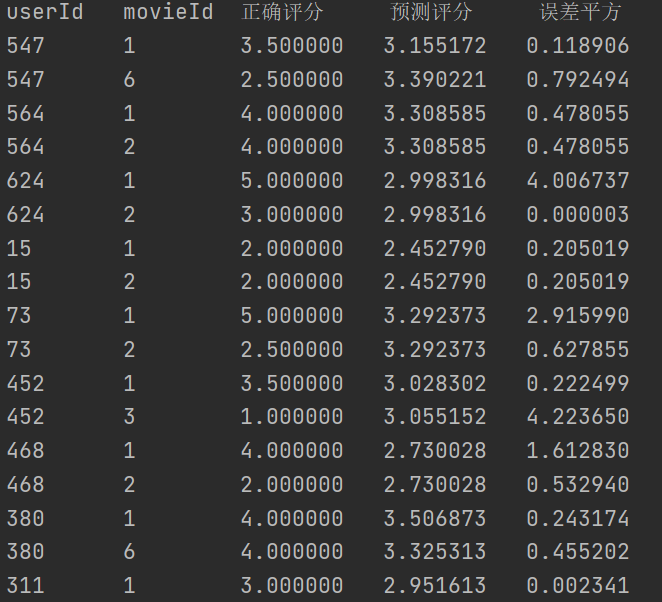
2.基于内容的推荐算法



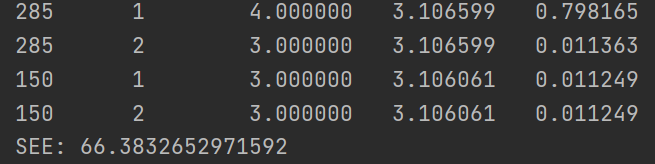
**图5-7 余弦相似度实验结果**



**图5-8 余弦相似度的SSE**

****

**图5-9 minhash的实验结果**

****

**图5-10 minhash的SSE**

基于minhash在本实验中得到的SSE的更小，虽然降低了维度，但是SSE确实更小了。此外余弦相似度的实验中，不计算tf-idf矩阵，直接用01矩阵代替tf-idf矩阵得到的结果与使用tf-idf矩阵得到的结果相近，所以本次实验中使用tf-idf矩阵并没有体现出优越性。

两个算法比较分析：

由上述实验结果可知，基于内容的推荐算法要明显优于基于用户的推荐算法。在基于用户的推荐算法中，由于k个相似的用户不一定也看过测试集中的电影，所以导致k值选取总会偏大，从而受到多个并不是那么相似的用户的干扰。

由以上结果也可以得知，在实际操作中，基于内容的推荐算法往往比基于用户的推荐算法更可靠。

## **1.4 实验总结**

1.基于用户的协同过滤推荐算法

左振：最后的大作业我完成的是基于用户的协同过滤算法，相对于基于内容来说代码量较少。大作业的难度相对来说比较低，实现的较为顺利，主要的工作就是在对于算法的理解与结果的分析上。

最后感谢老师和助教的指导。

2.基于内容的推荐算法

本实验中实现了推荐系统。基于内容的总体思路是通过所有电影的不同类别，通过某种合理的算法得到电影之间的相似度，并得到相似度矩阵。这样在推断用户对电影的喜好时，根据用户已打分电影，可以利用相似度矩阵找出用户对当前推测电影的喜好程度。本实验中对庞杂数据的处理比较困难，需要建立很多字典和列表保存数据，对矩阵的处理也不是简单粗暴的。此外只是根据电影的类别一个特征去预测结果也不是很好，比如100个预测数据的SSE就达到了67。

1. [↑](#footnote-ref-1)
2. [↑](#endnote-ref-1)