华中科技大学

课程报告

课程名称：大数据分析

专业班级：CS2008

学　　号：U202015176

姓　　名：刘鉴之

指导教师：王蔚

报告日期：2022.12.15

计算机科学与技术学院

目 录

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. 使用 map 多线程并行处理 9 个 source 文件，每个节点通过 shuffle 函数生成 3 部分结果
2. 使用 reduce 多线程处理某一部分结果，并生成对应的 reduce 文件。
3. 合并 3 个 reduce 节点产生的结果为 result.txt。

1.3.1 编程思路

map：使用九个线程分别记录某个 source 文件中的单词数，以<单词,1>的形式存储在字典中，并输出到 map 文件。

combine：使用九个线程分别处理 9 个 map 文件，统计某个单词出现的次数，以<单词,n>的形式存储在字典中，并输出到 combine 文件。

shuffle：使用九个线程分别处理 9 个 combine 文件，按照首字母将单词划分至 3 个文件中，其中首字母 d-o 划分至 shuffle1，p-z 划分至 shuffle2，其余划分至 shuffle3。

reduce：使用 3 个线程分别处理一个 shuffle，并输出到文件中。

1.3.2 遇到的问题及解决方式

问题：reduce 操作时的单词记录次数仍是已存在加一。

解决方式：将存在时的记录规则改为加上原来记录好的数值。

1.3.3 实验测试与结果分析

map、combine、shuffle 操作可视为对原始文件中单词的处理与划分，操作完成后输出的文件列表如下：

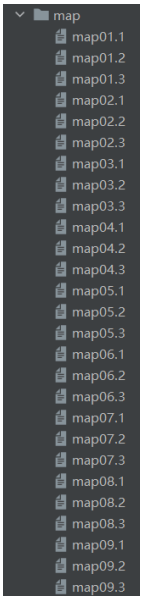

图 1.1 shuffle 操作后输出的文件列表

根据 shuffle 操作，首字母 d-o 的单词被划分至 shuffle1，其文件内容如下：

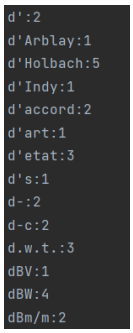

图 1.2 shuffle 操作后的文件内容

reduce 操作后输出的文件列表如下：

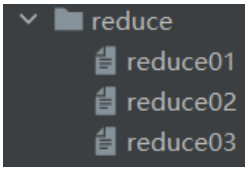

图 1.3 reduce 操作后的文件列表

1.4 实验总结

通过实现 MapReduce 算法，并结合之前学过的大数据处理课程，我深入掌握了该算法的运行原理，同时认识到了分布式计算在大数据分析领域的重要作用。

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]；（可选进阶版）理解阻尼系数[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

1. 读取 sent_receive.scv 文件内容，每一行划分为两个数字，存储在列表中。

2. 设人名 ID 数总计 m 个，构建 m×m 的邻接矩阵 M，当 i 向 j 寄过信时，M[i,j]=1。

3. 标准化矩阵 M 并构建初始 PageRank 矩阵 r。

---

[1] 基本 pagerank 公式 r=Mr

[2] 进阶版 pagerank 公式：r= $\beta Mr + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$，其中$\beta$为阻尼系数，常见值为 0.85

4. 根据 pagerank 公式计算 nextr 并标准化，计算 r 和 nextr 之间的误差值，并更新 r

5. 迭代至误差值小于 10^{-8} 后，停止迭代，输出结果。

2.3.2 遇到的问题及解决方式

2.3.3 实验测试与结果分析

迭代结束时，程序输出迭代次数、pagerank 矩阵，如下所示：

```
迭代次数: 55
[0.02714158 0.31773019 0.02326263 0.02810776 0.00489376 0.00740632
 0.00489376 0.00489376 0.00489376 0.00571015 0.00489376 0.00489376
 0.00740632 0.00634661 0.00181417 0.00920399 0.00489376 0.00099778
 0.00099778 0.00099778 0.00489376 0.00608569 0.00099778 0.00201326
 0.00099778 0.00099778 0.00201326 0.00099778 0.00218971 0.00099778
 0.00099778 0.00218971 0.00181417 0.00218971 0.00489376 0.0030061
 0.00489376 0.00099778 0.00872407 0.00218971 0.00489376 0.00218971
 0.00181417 0.00099778 0.00099778 0.00218971 0.00099778 0.00099778
 0.00201326 0.00099778 0.0059228  0.00099778 0.00099778 0.00201326
 0.00099778 0.00150552 0.00150552 0.00099778 0.00099778 0.00099778
 0.00099778 0.00099778 0.00218971 0.00489376 0.00099778 0.002456
 0.00099778 0.00571015 0.00099778 0.00181417 0.00181417 0.00489376
 0.00218971 0.00099778 0.00150552 0.00181417 0.00774587 0.00150552
 0.00099778 0.00099778 0.00099778 0.00099778 0.00099778 0.00489376
 0.00099778 0.00608569 0.00216266 0.00181417 0.00489376 0.00099778
 0.00302874 0.00099778 0.00181417 0.00192096 0.00099778 0.00181417
 0.00201326 0.00493942 0.00099778 0.00099778 0.00302874 0.00099778
 0.00489376 0.00099778 0.00099778 0.00099778 0.00099778 0.00351035
 0.00099778 0.00099778 0.00099778 0.00099778 0.0032997  0.00099778
 0.00099778 0.00099778 0.00099778 0.00099778 0.00099778 0.00099778
 0.00489376 0.00099778 0.00489376 0.00099778 0.00181417 0.00489376]
```

图 2.1 PageRank 程序运行结果

2.4 实验总结

通过本次实验，我掌握了 PageRank 算法，并深入掌握了 python 中各种数据结构的使用方法。

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. 从 Groceries.csv 中读取所有元素作为 C1。
2. 遍历 C1，统计每个 item 出现的次数，计算其支持度并筛选掉支持度小于最小支持度的项，得到频繁一项集 L1。
3. 组合 L1 中的元素，得到候选集 C2。
4. 统计 C2 中二项集的次数并筛选掉支持度小于最小支持度的项，得到频繁二项集 L2。
5. 重复 3-4 步，得到候选集 C3、频繁二项集 L3
6. 计算由 L2、L3 得到的关联规则，计算置信度并筛选掉置信度小于最小置信度的关联规则，得到关联规则集并输出

3.2.2 遇到的问题及解决方式

3.2.3 实验测试与结果分析

运行程序，输出文件 L1，L2，L3 中元素数目、关联规则数目，结果如下：

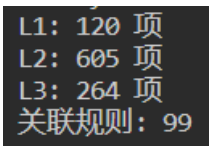

图 3.1 程序输出结果

	A	B	C	D
1	frozenset({'bottled water'	'bottled beer'	'soda'})	0.005083884087442806
2	frozenset({'root vegetables'	'whole milk'	'bottled water'})	0.007320793085917641
3	frozenset({'other vegetables'	'citrus fruit'	'whipped/sour cream'})	0.0056939501779359435
4	frozenset({'other vegetables'	'rolls/buns'	'pork'})	0.005592272496187087
5	frozenset({'other vegetables'	'yogurt'	'soda'})	0.008337569903406202
6	frozenset({'other vegetables'	'beef'	'rolls/buns'})	0.005795627859684799
7	frozenset({'citrus fruit'	'yogurt'	'rolls/buns'})	0.005795627859684799
8	frozenset({'other vegetables'	'whole milk'	'tropical fruit'})	0.01708185053380783
9	frozenset({'other vegetables'	'whole milk'	'bottled beer'})	0.007625826131164209
10	frozenset({'root vegetables'	'newspapers'	'whole milk'})	0.005795627859684799
11	frozenset({'fruit/vegetable juice'	'yogurt'	'soda'})	0.005083884087442806
12	frozenset({'whole milk'	'soda'	'chocolate'})	0.005083884087442806
13	frozenset({'whole milk'	'brown bread'	'tropical fruit'})	0.0056939501779359435

图 3.2 L3 文件的输出结果

1	frozenset({'baking powder'})	of	frozenset({'whole milk'})	: 0.5229885057471264
2	frozenset({'rolls/buns', 'pork'})	of	frozenset({'whole milk'})	: 0.5495495495495495
3	frozenset({'chicken', 'root vegetables'})	of	frozenset({'whole milk'})	: 0.5514018691588786
4	ozenset({'domestic eggs', 'whipped/sour cream'	of	frozenset({'whole milk'})	: 0.5714285714285715
5	frozenset({'beef', 'yogurt'})	of	frozenset({'whole milk'})	: 0.5217391304347826
6	frozenset({'rolls/buns', 'chicken'})	of	frozenset({'whole milk'})	: 0.5473684210526316
7	frozenset({'tropical fruit', 'root vegetables'})	of	frozenset({'other vegetables'})	: 0.5845410628019324
8	frozenset({'butter', 'tropical fruit'})	of	frozenset({'whole milk'})	: 0.6224489795918368
9	frozenset({'butter', 'domestic eggs'})	of	frozenset({'whole milk'})	: 0.6210526315789474
10	frozenset({'fruit/vegetable juice', 'yogurt'})	of	frozenset({'whole milk'})	: 0.5054347826086957
11	ozenset({'whipped/sour cream', 'root vegetables	of	frozenset({'other vegetables'})	: 0.5

图 3.3 程序生成的关联规则

3.3 实验总结

通过本次实验，我掌握了 Aprior 算法，认识到了关联规则挖掘通过频繁项集发现关联项的原理，也初步了解了其在推荐系统中的应用。

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(WineData.csv)，数据集已经被归一化(normalizedwinedata.csv)。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 4.1 所示。

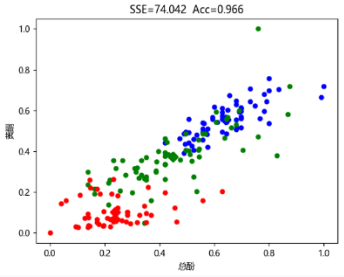

图 4.1 葡萄酒数据集在黄酮和总酚维度下聚类图像（SSE 为距离平方和，Acc 为准确率）

4.3 实验过程

4.3.1 编程思路

1. 读取归一化数据.csv 文件，每一行表示一个点存储在列表中。
2. 随机选取三个中心点，并根据其他点到中心点的最短距离将点集划分为三个聚类。
3. 重新计算每个聚类的质心，并重新计算其他点到中心点的最短距离以重新划分聚类。
4. 重复第三步，直到迭代前后中心点的坐标不再发生变化，由此得到最终的三个聚类。
5. 计算准确度，准确度计算公式为正确分类的项目数/总项目数。

4.3.2 遇到的问题及解决方式

4.3.3 实验测试与结果分析

运行程序算法准确度为 0.949，符合预期结果，如下所示：

```
迭代次数:  7
每个聚类的SSE:  [20.790425213135396, 15.001395753397441, 13.193593993511787]
实际SSE: 48.98541496004462
准确度:  0.949438202247191
```

图 4.2 程序计算结果

4.4 实验总结

通过本次实验，我掌握了 kmeans 算法，认识到了聚类算法在数据统计分析领域的重要应用。

实验五 推荐系统算法及其实现

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 minhash 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。

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 minhash 算法对效用矩阵进行降维处理，从而得到相似度矩阵，注意 minhash 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：

$$\text{score} = \frac{\sum_{i=1}^{n} score^{'}(i) * sim(i)}{\sum_{i=1}^{n} sim(i)}$$

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 minhash 算法对特征矩阵进行降维处理，从而得到相似度矩阵，注意 minhash 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（MinHash）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

1. 基于用户的协同滤波推荐算法

（1）读取电影信息、用户评分信息，并处理数据得到每个用户看过的电影、每个用户的评分信息、每个电影的用户列表。

（2）构建 user-movie 矩阵；设用户数目为 m，根据 user-movie 矩阵构建 m×m 的用户相似度矩阵。Pearson 相关系数，公式如下：

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

实现时，可以使用 numpy 库函数 corrcoef 计算 Pearson 相似度。

（3）根据用户相似度矩阵，得到与某个用户相似度最高的 k 个用户，称为该用户的 k 个邻居。

（4）遍历 k 个邻居看过且该用户未看过的的电影，计算预期评分。预期评分由两部分组成，一部分为用户对其已看过电影的平均评分，一部分为 k 个相似用户对当前电影的评分与用户平均评分之差的加权平均（权重为两者之间的相似度），其具体公式如下：

$$\widehat{R}_{i,m} = \overline{R_i} + \frac{\sum_{j=1}^{k} Sim(i,j)\left(R_{j,m} - \overline{R_j}\right)}{\sum_{j=1}^{k} Sim(i,j)}$$

（5）将预期评分进行排序，取评分最高的 N 部电影推荐给该用户。运行测试集时，预测每部电影的预期评分，并与标准评分比较计算 SSE。

2. 基于内容的推荐算法

（1）读取电影信息、用户评分信息，并处理数据得到每个用户看过的电影、每个用户的评分信息。

（2）构建 TF-IDF 矩阵。设电影数为 M，电影标签数为 N，构建 M×N 的 TF-IDF 矩阵，也即词频×反文档频率矩阵。

（3）根据 TF-IDF 矩阵构建 M×M 的电影相似度矩阵，使用余弦相似度进行计算，其公式如下：

$$Cossim(x,y) = \frac{\vec{x}\cdot\vec{y}}{|\vec{x}|\cdot|\vec{y}|}$$

实现时，可以使用 sklearn 库函数 cosine_similarity 计算余弦相似度。

（4）遍历该用户未看过的的电影，计算预期评分。根据用户已评分电影和相似度矩阵，预测该用户对未看过电影的预期评分，具体公式如下：

$$\hat{S} = \frac{\sum_{i=1}^{n} S_i \cdot sim(i)}{\sum_{i=1}^{n} sim(i)}$$

（5）将预期评分进行排序，取评分最高的 N 部电影推荐给该用户。运行测试集时，预测每部电影的预期评分，并与标准评分比较计算 SSE。

5.3.2 遇到的问题及解决方式

5.3.3 实验测试与结果分析

1. 基于用户的协同过滤算法

对于用户 1，选取 10 个邻居，为其推荐 5 部电影的运行结果如下：

```
输入用户ID（0表示运行测试集）:1
电影ID          预期评分        电影名                                  电影类型
27478           5.745652        Ali G Indahouse (2002)                  ['Comedy']
69              5.745652        Friday (1995)                           ['Comedy']
69122           4.745652        Hangover, The (2009)                    ['Comedy', 'Crime']
35836           4.745652        40-Year-Old Virgin, The (2005)          ['Comedy', 'Romance']
3871            4.603830        Shane (1953)                            ['Drama', 'Western']
```

图 5.1 对用户 1 的推荐结果

运行测试集结果如下所示：

```
481  1 4.246785 4.00
481 16 4.303768 4.00
665  2 3.255997 3.00
665  3 3.225330 3.00
607  1 3.751639 4.50
607  2 3.593530 3.00
 19  1 3.740396 3.00
 19  2 3.465037 3.00
199  6 3.747933 4.50
199 10 3.579544 2.50
285  1 3.215255 4.00
285  2 3.123716 3.00
150  1 3.251866 3.00
150  2 3.061459 3.00

SSE为 64.02314317647144
```

图 5.2 测试集运行结果（基于用户的协同过滤算法）

2. 基于内容的推荐算法

对用户 1 推荐 5 部电影，运行结果如下图所示

```
输入用户ID(输入0运行测试集): 1
电影ID          预期评分        电影名                                  电影类型
50912   3.500000        Paris, I Love You (Paris, je t'aime) (2006)     ['Romance']
 8911   3.500000        Raise Your Voice (2004)                         ['Romance']
 7320   3.500000        Dirty Dancing: Havana Nights (2004)             ['Romance']
 4503   3.500000        Everybody's All-American (1988)                 ['Romance']
 4024   3.500000        House of Mirth, The (2000)                      ['Romance']
```

图 5.3 对用户 1 的推荐结果

运行测试集结果如下所示：

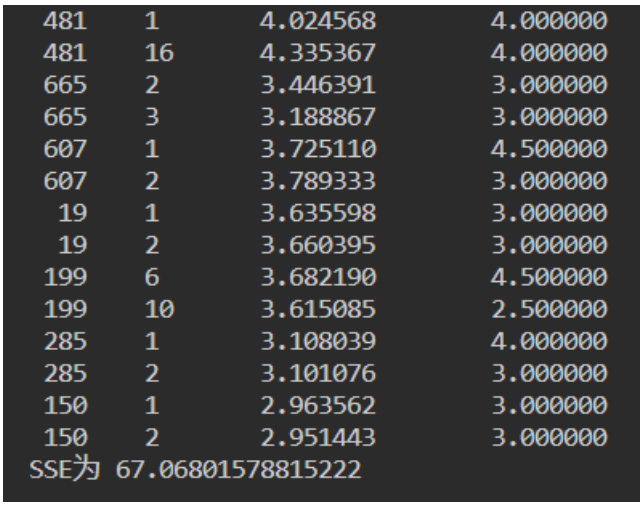

图 5.4 测试集运行结果（基于内容的推荐算法）

5.4 实验总结

通过本次实验，我掌握了分为基于用户的协同过滤算法和基于内容的推荐算法这两种在推荐系统中常用的推荐算法。