

**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： 2022年1月8日**

**计算机科学与技术学院**

**目 录**

[实验一 wordCount算法及其实现 1](#_Toc92625594)

[**1.1 实验目的** 1](#_Toc92625595)

[**1.2 实验内容** 1](#_Toc92625596)

[**1.3 实验过程** 1](#_Toc92625597)

[1.3.1 编程思路 1](#_Toc92625598)

[1.3.2 遇到的问题及解决方式 2](#_Toc92625599)

[1.3.3 实验测试与结果分析 2](#_Toc92625600)

[**1.4 实验总结** 4](#_Toc92625601)

[实验二 PageRank算法及其实现 5](#_Toc92625602)

[**2.1 实验目的** 5](#_Toc92625603)

[**2.2 实验内容** 5](#_Toc92625604)

[**2.3 实验内容** 5](#_Toc92625605)

[2.3.1 编程思路 5](#_Toc92625606)

[2.3.2 遇到的问题及解决方式 6](#_Toc92625607)

[2.3.3 实验测试与结果分析 6](#_Toc92625608)

[**2.4 实验总结** 6](#_Toc92625609)

[实验三 关系挖掘实验 7](#_Toc92625610)

[**3.1 实验内容** 7](#_Toc92625611)

[**3.2 实验过程** 7](#_Toc92625612)

[3.2.1 编程思路 7](#_Toc92625613)

[3.2.2 遇到的问题及解决方式 7](#_Toc92625614)

[3.2.3 实验测试与结果分析 8](#_Toc92625615)

[**3.3 实验总结** 8](#_Toc92625616)

[实验四 kmeans算法及其实现 9](#_Toc92625617)

[**4.1 实验目的** 9](#_Toc92625618)

[**4.2 实验内容** 9](#_Toc92625619)

[**4.3 实验过程** 10](#_Toc92625620)

[4.3.1 编程思路 10](#_Toc92625621)

[4.3.2 遇到的问题及解决方式 10](#_Toc92625622)

[4.3.3 实验测试与结果分析 10](#_Toc92625623)

[**4.4 实验总结** 11](#_Toc92625624)

[实验五 推荐系统算法及其实现 12](#_Toc92625625)

[**5.1 实验目的** 12](#_Toc92625626)

[**5.2 实验内容** 12](#_Toc92625627)

[**5.3 实验过程** 14](#_Toc92625628)

[5.3.1 编程思路 14](#_Toc92625629)

[5.3.2 遇到的问题及解决方式 15](#_Toc92625630)

[5.3.3 实验测试与结果分析 16](#_Toc92625631)

[**5.4 实验总结** 17](#_Toc92625632)

# 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

map：利用split()函数从源文件中读取每个以逗号分隔的单词，以（单词，1）的形式保存在map文件中。设立九个线程完成上述操作。

combine：建立一个字典，遍历map文件中每一条记录，当单词不存在于字典中时，则新建条目；若单词已存在于字典中时，则数值加一。最后按顺序将字典中的内容输出至reduce文件中。设立九个线程完成上述操作。

shuffle：遍历reduce文件，按照首字母顺序将单词划分至3个文件中（大小写视为同一个字母），其中首字母A~G划分至shuffle1，J~R划分至shuffle2，其余划分至shuffle3。设立九个线程完成上述操作。

reduce：操作与combine一致，区别为只设立三个线程完成（对应shuffle1~3），以及输出前需将字典键值按字母顺序排序。

### 1.3.2 遇到的问题及解决方式

**问题**：在实现线程计时的过程中，使用time.clock()方法会报错。

**解决方式**：目前python中已将time.clock()移除，改用time.perf\_counter()代替即可。

### 1.3.3 实验测试与结果分析

map程序将单词输出为(单词,1)的格式，由于source中提供了1000000条记录，在生成的map文件也应有这些记录。如图1.1所示，map程序运行正确。

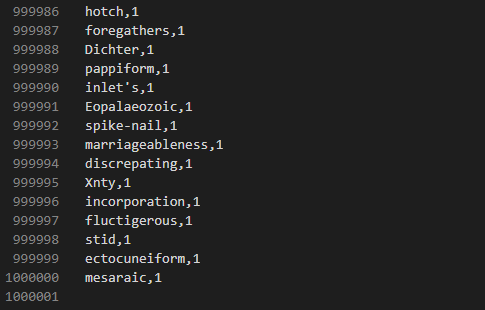


图 1.1 map程序的运行结果

combine程序将map文件中相同条目合并并输出到combine文件中。如图1.2可知，合并操作运行正确。

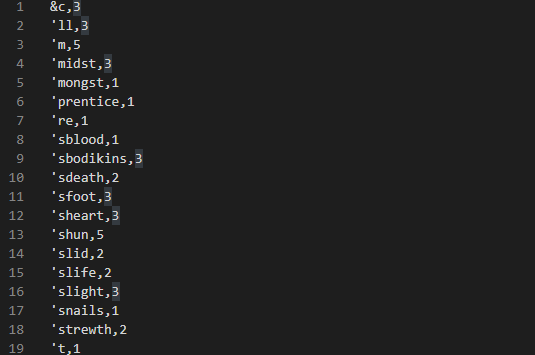


图 1.2 combine程序的与运行结果

shuffle程序将combine文件中的条目按首字母划分至三个文件中。由于1.3.1中的设计思路，shuffle1文件应只包含首字母A~G，如图1.3所示，shuffle程序运行正确。

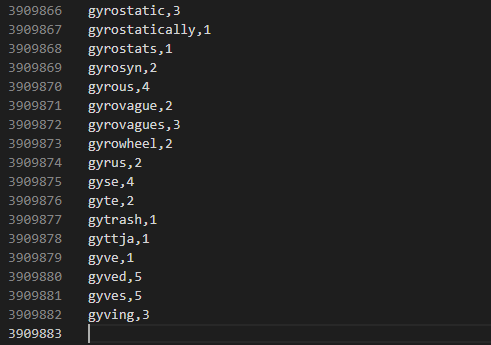


图 1.3 shuffle程序的运行结果

reduce程序与combine程序完全一致，测试略。

计时功能分别获取进程开始时的时间和结束时的时间，相减得到用时。如图所示，计时功能运行正确。

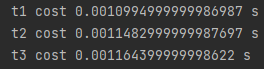


图 1.4 reduce程序中的计时程序

## 1.4 实验总结

通过对map-reduce算法的实现，使我能够深入认识到这一算法的运行过程和运行原理，同时理解多线程处理的优势以及这一简单算法在分布式节点中起到的重要作用。同时，由于这是我首次编写python程序，这次实验的编写过程也让我了解了python的基本语法和一些操作。

# 实验二 PageRank算法及其实现

## 2.1 实验目的

1、学习pagerank算法并熟悉其推导过程；

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

3、将pagerank算法运用于实际，并对结果进行分析。

## 2.2 实验内容

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

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

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

输出人名id及其对应的pagerank值。

## 2.3 实验内容

### 2.3.1 编程思路

1. 从源文件sent\_receive.scv中读取每一封邮件的发件人和收件人，并按照读取到的顺序保存在一个列表中。

2. 计算这个列表中元素的个数len，创建一个大小为len×len的矩阵M（即邻接矩阵），当存在i发送给j的邮件时，M[i,j]=1。

3. 标准化矩阵M，即使矩阵的每一列元素之和为1。

4. 代入迭代公式进行计算，同时注意每一步均需要将矩阵进行标准化。

5. 迭代至两次结果矩阵元素之差的最大值小于10-8时，结束迭代并输出最终结果。

### 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、结果r矩阵以及r中元素之和。如图2.1所示，程序运行正确。

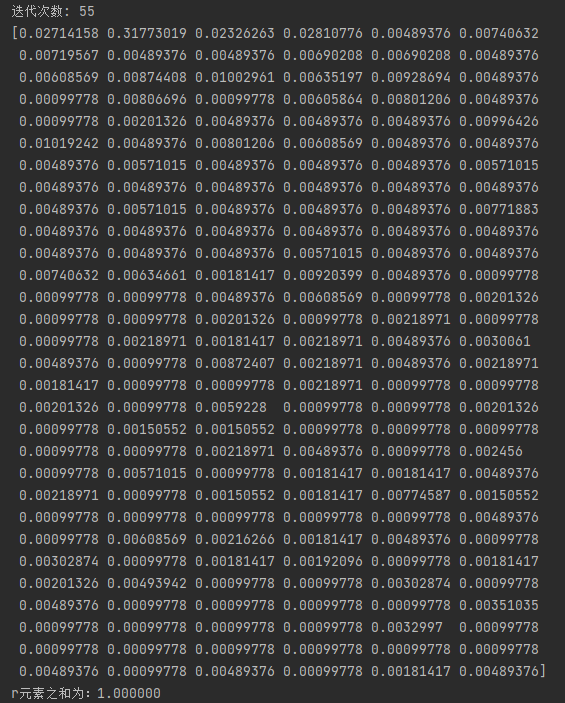


图 2.1 PageRank程序运行结果

## 2.4 实验总结

通过本次实验使我了解了PageRank算法的基本思路，也借此了解了Google搜索结果排序的原理，深刻感受到大数据方法在互联网应用中的巨大应用和重大作用。

# 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，计算每个项目的的支持度，根据规定的最小支持度排除不满足项，得频繁一项集L1。

3. 对L1中元素进行任意组合，并采取剪枝策略，得到候选集C2。计算C2中元素的支持度并根据最小支持度排除不满足项，得频繁二项集L2。

4. 对L2中元素，当两集合仅有最后一个元素不同时，取两集合的并集，并采取剪枝策略判断，得到候选集C3，计算C3中元素的支持度并根据最小支持度排除不满足项，得频繁二项集L3。

5. 利用置信度公式，计算关联规则得置信度，根据最小置信度筛选出符合条件的关联规则

### 3.2.2 遇到的问题及解决方式

**问题**：由Ln生成Cn+1、Ln+1的速度过慢。

**解决方式**：在由Ln生成Cn+1的过程中，采取剪枝策略，即只有当元素的所有子集都在Ln中时，该元素才能被作为Cn+1中的元素。这样可以大大减少

### 3.2.3 实验测试与结果分析

运行程序，输出文件L1，L2，L3中包含项目的数量和最终获得的关联规则，与提供的参考结果对比可知结果正确。

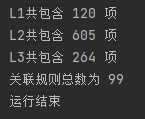


图 3.1 程序输出结果

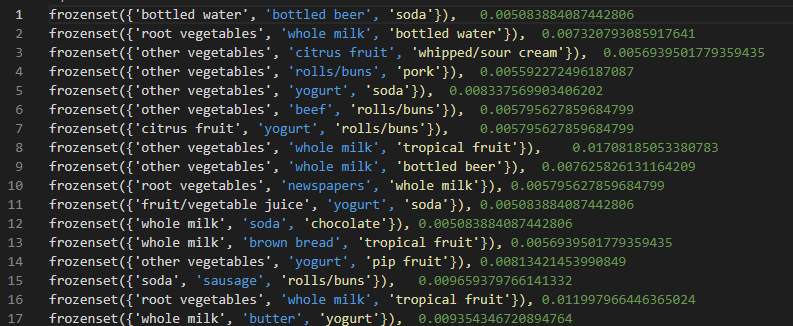


图 3.2 L3文件的输出结果

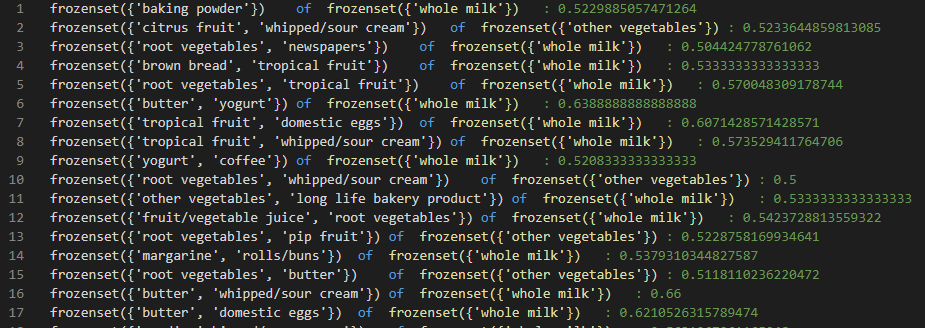


图 3.3 程序生成的关联规则

## 3.3 实验总结

本次实验涉及到了关联规则的挖掘，是大数据在生活中的重要应用，理解了经典的“啤酒与纸尿裤”的实现。另外，利用剪枝策略对运算过程进行优化也使我理解到在程序运行中随时剔除无效数据对大规模计算速度的显著提升效果。

# 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. 从normalizedwinedata.csv文件中读取各项数据，每项数据以列表的形式存储13维数据。

2. 任选三个中心点（可用随机数的方法），计算每个点到这三个中心点的距离，保存最小距离和对应中心点，即按中心点分为3个聚类。

3. 在每一个聚类中，计算各个样本点的重心，以此作为新的中心点，得到三个新的中心点。

4. 计算每个点到这三个中心点的距离，保存最小距离和对应中心点，即按中心点分为3个聚类。

5. 重复3、4，直到两次迭代中，三个中心点的坐标未发生变化，即得三个聚类。

6. 计算准确度，准确度。

7. 利用python库matplotlib.pyplot绘制散点图，得到数据可视化结果。

### 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

程序运行结果如图4.2所示，选择镁和黄酮两个指标作为绘制图形的依据。有图表展示结果可知，聚类程序运行正确。

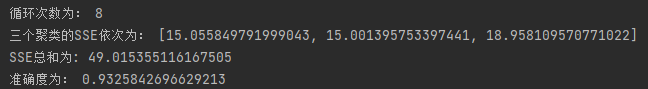


图4.2 程序计算结果

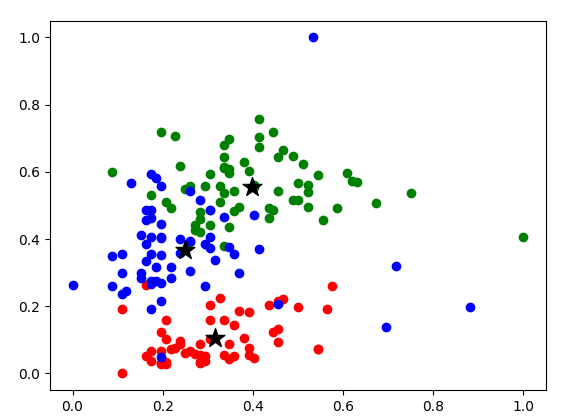


图4.3 以镁和黄酮为横、纵坐标绘制的图形

## 4.4 实验总结

实验实现了kmeans聚类算法，体现了逐次修正以逼近正确结果的思想。另外，程序中涉及了调用第三方库实现绘制图表的操作，体现了python语言实现此类操作的便捷性和优越性。

# 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：

**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. 基于用户的协同过滤推荐算法**

（1）从文件movies.csv中读取电影的信息：电影名、电影ID、电影分类，这是为了后续推荐是可以展示电影的完整信息。

（2）从文件ratings.csv中读取用户的评分信息：用户ID、被评价的电影ID和用户评分。这是为了查找相似用户以及计算推荐电影的预计评分。根据读取到的信息建立列表保存每部电影的评分用户ID。

（3）获取用户数目m，建立一个m×m的矩阵用于存储用户间的Pearson相关系数。计算用户间的Pearson相关系数，公式如下：

其中，xi表示用户x对电影i的评分，yi表示用户y对电影i的评分，n为电影总数。

（4）对于一个用户，取相关系数矩阵对应列的所有元素排序，取得与其相似度最高的k个用户（k由程序使用者指定），保存至一个列表中。

（5）计算这k个用户评分过、但被推荐者未评分过的所有电影的预计评分。预计评分由两部分组成，一部分为用户对其已看过电影的平均评分，一部分为补充评分。补充评分为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，权重为相似用户与当前用户的相关系数。具体公式为：

其中，Ri表示用户i对其看过电影的平均评分，Sim(i,j)表示用户i与用户j的相关系数，Ri,m表示用户i对电影m的评分。

（6）将这些评分结果排序，取最大的n个结果（n由程序使用者指定），按步骤（1）读取到的信息进行输出。

（7）运行测试集，对测试集中用户的评分进行预测，并与实际值进行比较，误差为预测值与实际值差的平方和，即

**2. 基于内容的推荐算法**

（1）从文件movies.csv中读取电影信息：电影名、电影ID、电影分类，这是为了计算电影之间的相关系数和便于展示最终结果。

（2）从ratings.csv中读取用户评分相关信息：用户ID、被评价的电影、用户评分，建立字典存储每个用户评过分的电影，这是为了通过用户评过分电影来估计用户对未评分电影的评分。

（3）获取电影总数m和分类总数n，建立一个m×n的矩阵作为TF-IDF矩阵，计算每个电影的每个分类的TF-IDF值。TF是词频，即一个单词在文件中的出现次数，在此问题中，电影包含此分类为1，否则为0。IDF是反文档频率，值为文档总数与包含该单词的文档总数加一的比值的对数，其中加一是为了避免分母为0（但在此例下分母不会为0，因为未出现的分类不会出现在分类列表中）。TF与IDF之积即为TF-IDF值。

（4）电影总数为m，建立一个m×m的矩阵保存电影间的余弦相关系数，根据TF-IDF值可计算电影间的余弦相关系数，具体公式为：

其中，向量x，y为电影x和电影y的TF-IDF向量。

（5）对每一部用户用户没有看过的电影，利用用户已评分数据计算预计评分，若电影相似度为负数则不参与计算。若计算出的预计评分为0，则取用户已评分电影的均值作为预测结果。计算公式为：

其中，si为用户对看过的第i部电影的评分，sim(i)为用户看过的第i部电影与当前电影的相似度。

（6）将这些电影的预计评分由高到低排序，取最大的n个结果（n由程序使用者指定），将这些电影的信息（由步骤1中读取）输出。

（7）运行测试集，用步骤5中的方法计算预计评分，与用户实际评分进行比较，计算SSE值。

### 5.3.2 遇到的问题及解决方式

1. 运行测试集时，若复用推荐算法来计算预期评分，会导致计算时间过长。这是由于推荐算法会计算所有电影来进行推荐，而运行测试集是只需计算目标电影即可。因此可重新构建一个测试集用函数，相比推荐算法去除非目标电影的计算过程。

2. 计算pearson相似度和余弦相似度时，使用自己编写的算法进行运算时间较长。可直接调用相关库中的函数进行运算，有助于加快程序运行速度。

3. 在基于用户的协同过滤算法中，采用原始公式计算时，可能出现误差较大的情况，这是由于与目标用户相近的用户中，可能很多人未看过某部电影，使预计评分过于接近那些看过此电影的人的评分。改进措施是将预期评分分为基础评分和补充评分两部分，具体如5.3.1中相关内容所示。这样的优点是可以充分利用各个相似用户的数据，而不会导致结果有明显偏向性。

4. 在基于内容的推荐算法中，使用公式计算预期评分时，有可能出现相似电影的相似度之和为0的情况，即所有相似电影的相似度均为负值而被舍弃。可以在求和运算结束时进行判定，若相似度之和为0，则预期评分指定为目标用户所评分的平均值。

### 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10，取与其相近的前50个用户，为其推荐10部电影的运行结果如下图所示，为保留计算出的原始数据，未将超过5的预计评分化为5。

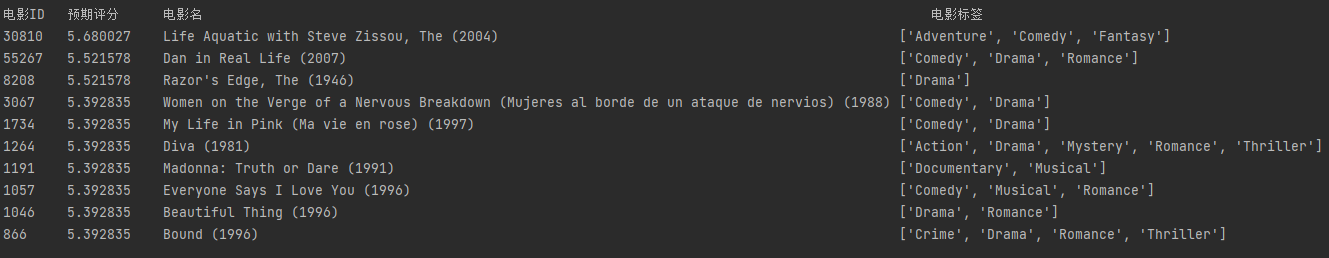
****

图 5.1 对用户10的推荐结果

对于测试集，取相似用户数量为50，运行结果如下图所示，预期评分与实际评分较为接近。



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10推荐10部电影，运行结果如下图所示

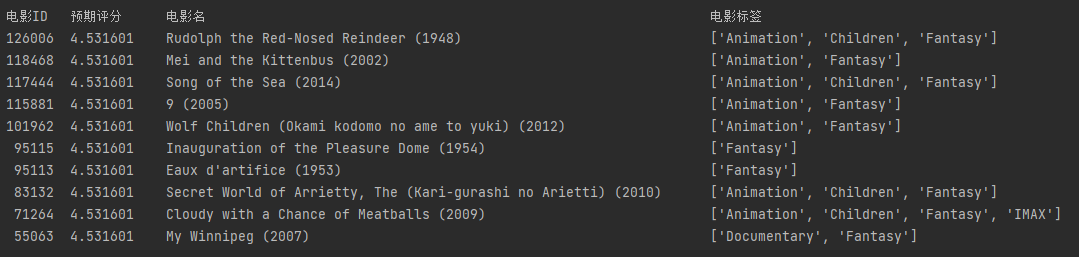


图 5.3 对用户10的推荐结果

运行测试集，部分结果和SSE结果如图5.4所示，预计评分与实际评分较为符合。

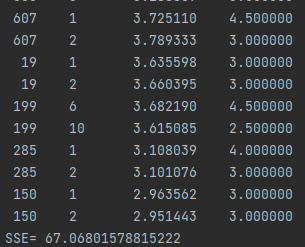


图5.4测试集运行结果（基于内容的推荐算法）

## 5.4 实验总结

本次实验内容是推荐系统的实现，分为基于用户的协同过滤算法和基于内容的推荐算法，分别以用户间的相似性和电影间的相似性为出发点，来对目标用户的喜好进行预测，这也是实际中推荐系统的重要方法。

但是在实现过程中，我也发现了这两种算法存在的问题。对于基于用户的协同过滤算法，如果用户的相似用户中看过某部电影的人较少，那么在不对公式进行修正的情况下，对于这部电影的评分就会被这少数人极大影响，从而造成结果的失真。而基于内容的推荐算法存在的问题是，电影的分类并不能很好地作为用户评分的参考标准，即用户评分时不会以分类作为单一标准，还可能有电影情节、特效制作等方面的标准，仅以电影分类为预测标准是有失偏颇的。因此，在真实的推荐系统中，我们不仅需要增加推荐的相关指标，也需要将基于用户、基于内容两种方式进行结合，以期找到更精准的推荐结果。

在程序的编写中，虽然我的推荐模块能够计算电影的预计评分，但是在运行测试集时，简单复用这一模块会导致运算时间大大延长，这是由于这一过程中进行了大量的无用计算。因此还是应该编写一个新的测试集模块，在推荐模块基础上去除无用计算的部分，也可以将对某部电影预期评分计算单独作为一个模块运行。这样虽然使代码长度增加，却也加快了程序运行速度，这是在程序编写中需要注意的问题。

在pearson相似度和余弦相似度计算的过程中，如果使用自己编写的算法进行运算，也会导致运行时间偏长。调用第三方库中的计算函数即可优化这一过程。可见，在实际程序编写过程中，我们并不需要自己实现所有的运算过程，合理运用第三方库提供的功能不仅有助于简化程序编写，有时也能加快程序的运行速度。

这次实验过程中，运用了较多的循环过程，涉及到很多数据结构如列表和字典的迭代，这些内容在python程序中实现起来极为容易，这也是python语言在机器学习、大数据等领域大放异彩的原因。

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