

**课 程 实 验 报 告**

**课程名称： 大数据分析**

**专业班级： CS2008**

**学 号： U202015533**

**姓 名： 徐瑞达**

**指导教师： 王蔚**

**报告日期： 2022年12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。

## 1.3 实验过程

### 1.3.1 编程思路

（1）Map

由于共有9个源文件，因此需要利用多线程模拟9个Map节点，每个节点处理一个源文件。在每个节点中，利用Scanner和split()函数从源文件中读取单词；使用Map<String,Interger>对象存储每个单词出现的次数。

（2）Combine

在每个Map节点处理源文件时，可以使用TreeMap实现。当单词已存在于字典中时put(word,get(word)+1)，否则调用put(word,1)录入单词。

（3）Shuffle

在每个Map节点处理完成后，使用shuffle函数将统计结果分为三部分。将首字母为a~o的单词划分至shuffle2、首字母为p-z的单词划分至shuffle3、其余单词划分至shuffle3。最终输出结果时，每个节点将输出三个map文件。

（4）Reduce

本实验要求模拟3个Reduce节点，每个节点处理一个shuffle文件。以第1个Reduce节点为例，遍历每个Map节点输出的shuffle1文件，按照与Combine类似的逻辑统计单词个数，最终输出到reduce1文件中。

### 1.3.2 遇到的问题及解决方式

**问题**：在实现多线程时，Reduce节点应在所有Map节点完成后运行，许因此需要统一管理Map节点。

**解决方式**：使用Java提供的ExecutorService类统一管理Map节点，通过轮询所有线程是否结束，实现Map节点均完成后运行Map节点。

### 1.3.3 实验测试与结果分析

由于Map、Combine、Shuffle的功能均在线程类MapRunner中实现，因此这里只展示Shuffle后的文件。文件结构如图1.1所示，可见每个Map节点共输出3份文件，分别是part1、part2、part3且各部分文件大小基本相同。文件map01.part1内容如图1.2所示，可见经过Combine处理后，得到每个单词出现的次数。

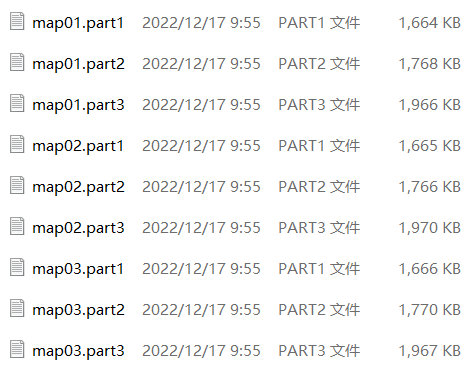


图1.1 Map处理后的文件结构

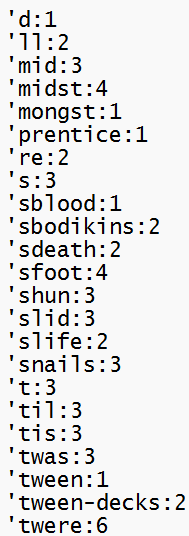


图1.2 Map处理后的文件内容

Reduce负责统计各个part中单词出现次数，最终合并3个Reduce节点的结果，文件内容如图1.3所示。

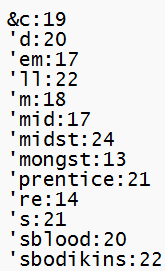


图1.3 Reduce处理后的文件内容

统计各个Map节点和Reduce节点运行时间，结果如图1.4所示。

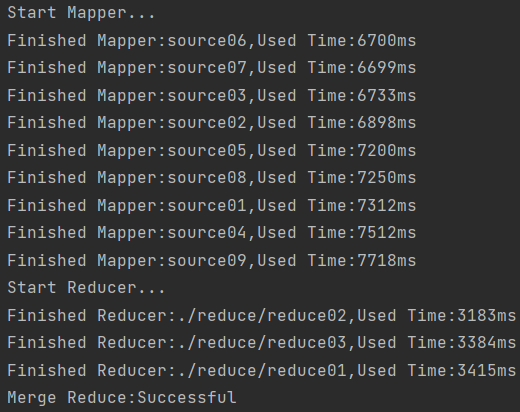


图1.4 各节点运行时间

## 1.4 实验总结

通过实现MapReduce算法，我认识到了分布式运算的强大之处——通过多线程处理提高运行效率。

# 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。

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。最后输出人名id及其对应的pagerank值。

## 2.3 实验内容

### 2.3.1 编程思路

1. 定义矩阵类Matrix；
2. 读取文件sent\_receive.csv，统计节点集nodes
3. 构建大小为nodesLength×nodesLength的邻接矩阵M（当存在i发送给j的邮件时，M[i,j]=1）；
4. 标准化矩阵M，使矩阵的每一列元素之和为1；
5. 初始化currentPr矩阵；
6. 代入迭代公式计算nextPr并标准化，计算每次迭代前后的误差率errorRate；
7. 迭代至误差率小于10-8时，终止迭代并输出最终结果。

### 2.3.2 遇到的问题及解决方式

**问题**：迭代次数过少，最终结果错误。

**解决方式**：邻接矩阵M构建错误，重新构建后结果正确。

### 2.3.3 实验测试与结果分析

迭代终止后，输出迭代次数与ID对应PageRank值，结果如图2.1所示。

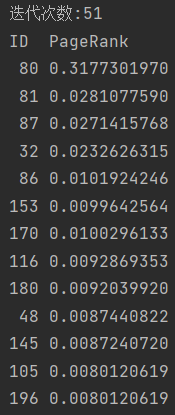


图2.1 PageRank输出结果

## 2.4 实验总结

通过本次实验，我掌握了PageRank算法，也了解了搜索引擎返回搜索结果的工作方式，深刻体会到了大数据算法在互联网中的重要作用。

# 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basket作为C1；
2. 遍历C1中的basket，统计每个元素出现的次数，同时统计所有可能的二项集出现的次数。遍历完成后，调用getSupportedItems方法排除出现频率小于于支持度support的元素以得到频繁一项集L1。使用hash函数将二项集统计结果哈希到桶中，并压缩为01数组；
3. 组合L1中元素得到二项集，并根据01数组进行剪枝，得到候选集C2。调用getSupportedItems方法得频繁二项集L2；
4. 依照C2、L2的生成方式得到C3、L3、C4、L4；
5. 遍历由L2、L3、L4生成的关联规则，根据置信度公式计算置信度，最后筛选出大于最小置信度的关联规则并输出。

### 3.2.2 遇到的问题及解决方式

**问题**：实现之前，没有正确理解PCY算法对于Aprior算法的改进方式，导致频繁二项集L2统计错误。

**解决方式**：PCY算法的改进之处在于，在遍历C1中的basket时，统计所有可能的二项集出现的次数，并哈希到桶中。

### 3.2.3 实验测试与结果分析

在使用PCY算法之前，运行结果如图3.1所示。

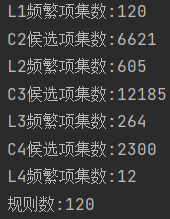


图3.1 Aprior算法运行结果

使用PCY算法之后，运行结果如图3.2所示，可见C2候选项集数缩减为一半左右，大大提高了效率。其中第一行是展示的部分01数组。

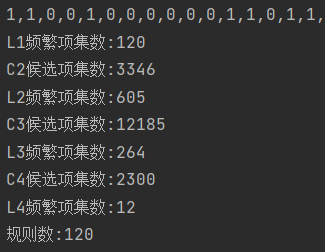


图3.2 PCY算法运行结果

输出的L4文件如图3.3所示。

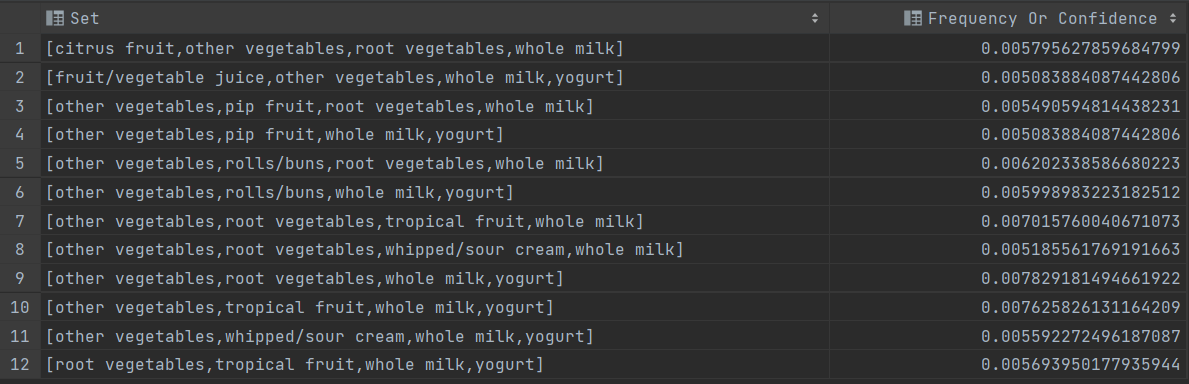


图3.3 输出的L4文件

输出的关联规则文件如图3.4所示。

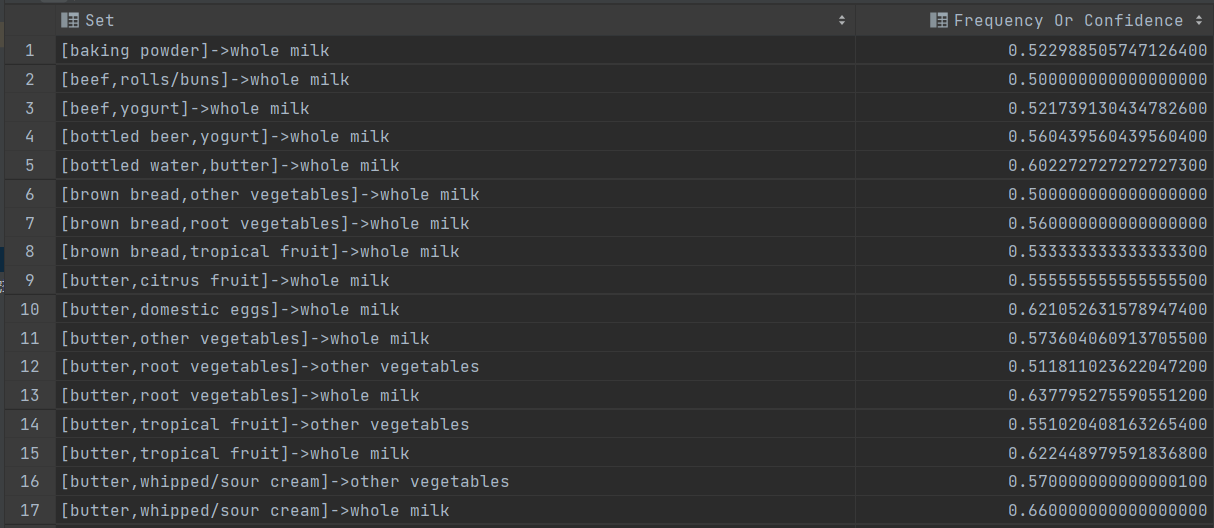


图3.4 输出的关联规则文件

## 3.3 实验总结

在本次实验中，我实现了关联规则挖掘Aprior算法，并使用PCY算法显著提高了算法的运行效率，这让我认识到通过剪枝策略在运行时剔除无效数据对提高大数据运算性能的重要作用。

# 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。

## 4.3 实验过程

### 4.3.1 编程思路

1. 从归一化数据文件中读取各项数据，每项数据包含13维特征，存储在Point对象中；
2. 随机选取三个点作为初始中心点，计算每个点到这三个中心点的距离，选取距离最小对应的中心点并根据每个点距离最小的中心点划分为三个聚类Cluster；
3. 根据欧式距离公式，计算每个聚类的新质心，重复2、3步；
4. 迭代直至迭代前后中心点的坐标不再变化，即得最终结果；
5. 分别统计标准结果、实际结果的SSE，并计算准确度，准确度公式如下

### 4.3.2 遇到的问题及解决方式

**问题**：误以为准确度计算公式为实际结果SSE/标准结果SSE。

**解决方式**：按照正确的准确度计算公式，统计正确分类的项目数。

### 4.3.3 实验测试与结果分析

程序运行结果如图4.1所示。其中准确度约为0.9428，程序结果符合预期。

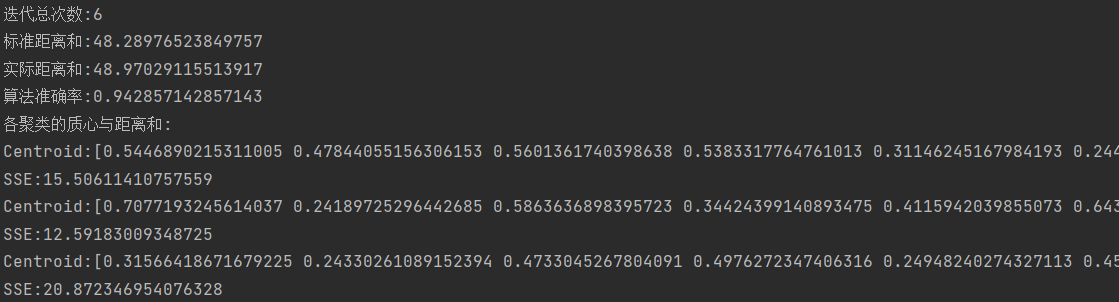


图4.1 Kmeans聚类算法输出结果

## 4.4 实验总结

本次实验中，我实现了Kmeans聚类算法，掌握了通过迭代修正逼近标准结果的算法思想。同时，由于我采用Java语言编写的程序，不方便绘制准确度图表，让我认识到了Python语言在大数据处理、机器学习等领域的重要性。

# 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

基础内容一：**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**。

基础内容二：**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**。

## 5.3 实验过程

## 5.3.1 编程思路

**1. 基于用户的协同过滤算法**

该算法的核心在于计算用户之间的相似度，以根据邻居评分情况推荐电影。

1. 读取数据

* 从test\_set.csv文件中读取所有评分记录：读取用户ID、电影ID、评分存储在Record对象中，获得原始评分数据集；
* 从movies.csv文件中读取所有电影：读取电影ID、电影名称、电影类型存储在Movie对象中，获得电影信息集以便于计算电影之间的相似度；
* 根据评分记录，统计出每个用户的评分列表、看过的电影列表，同时统计出看过某部电影的用户列表。

1. 计算用户之间的相似度

* 构建User-Movie矩阵：设用户数为M，电影数为N，构建M×N的矩阵，Matrix[i,j]表示用户i对电影j的评分；
* 计算Pearson相似度：根据User-Movie矩阵，计算两个用户之间的相似度，构建M×M的相似度矩阵，计算公式如下：

1. 推荐电影

* 选取邻居：对于某个用户，根据相似度矩阵获取与该用户相似度最高的K个用户；
* 计算预期评分：计算得到的K个邻居看过但是该用户未看过的电影的预期评分。预期评分的计算公式如下，其中一部分为该用户对已看过电影的评分平均值，一部分为K个邻居对当前电影的评分与该邻居平均评分之差的加权值（加权系数为相似度）；
* 对所有预期评分结果排序，选取评分最高的N个结果推荐给该用户。

1. 运行测试集

读取测试集，计算每部电影的预期评分，与实际评分比较，计算得到SSE值。

**2. 基于内容的推荐算法**

该算法的核心在于计算电影之间的相似度，以根据用户评分情况推荐电影。

1. 读取数据

* 从test\_set.csv文件中读取所有评分记录：读取用户ID、电影ID、评分存储在Record对象中，获得原始评分数据集；
* 从movies.csv文件中读取所有电影：读取电影ID、电影名称、电影类型存储在Movie对象中，获得电影信息集以便于计算电影之间的相似度；
* 根据评分记录，统计出每个用户的评分列表、看过的电影列表。

1. 计算电影之间的相似度

* 计算TF-IDF矩阵：根据电影信息集，统计电影总数M和类型总数N，构建M×N的TF-IDF矩阵。首先分别计算每个电影的每个类型标签的TF值和IDF值；TF表示词频，即单词在文件中出现的次数，当电影具有某个标签时，TF为1/N，否则为0；IDF表示反文档频率，其公式如下。TF矩阵和IDF矩阵对应相乘即为TF-IDF矩阵。
* 计算余弦相似度：构建M×M的相似度矩阵。遍历TF-IDF矩阵中的某两行，根据余弦公式（如下）计算得到这两行对应电影的相似度，写入相似度矩阵中。Matrix[i,j]表示电影i和电影j之间的相似度。

1. 推荐电影

* 对于某一用户，计算其未看过电影的预期评分：根据用户的评分记录和电影间的相似度计算预期评分，电影相似度为负数时不参与计算。计算公式如下：
* 对所有预期评分结果排序，选取评分最高的N个结果推荐给该用户

1. 运行测试集

读取测试集，计算每部电影的预期评分，与实际评分比较，计算得到SSE值。

### 遇到的问题及解决方式

**问题：**在基于用户的协同过滤算法中，采用原始公式计算时，由于在某用户的邻居中，可能存在很多人未看过电影A，因此预期评分会偏向于看过电影A的少部分邻居的评分，存在较大误差。

**解决措施：**将预期评分分为两部分，其中一部分为该用户对已看过电影的评分平均值，一部分为K个邻居对当前电影的评分与该邻居平均评分之差的加权值。这样可以充分利用各邻居的评分，不会导致太大的误差。

### 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12，选择10个邻居，为其推荐5部电影。运行结果如图5.1所示。

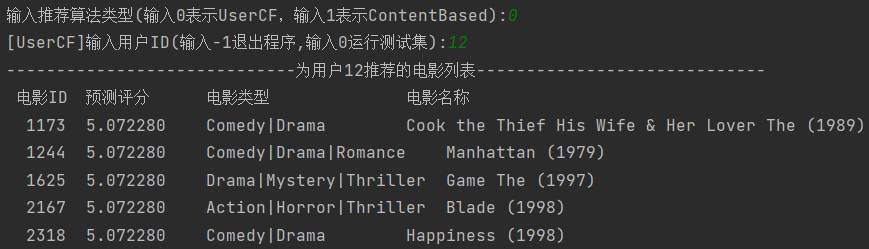


图5.1 基于用户的协同过滤算法推荐结果

运行测试集的结果如图5.2所示。

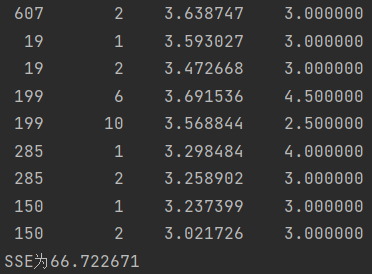


图5.2 基于用户的协同过滤算法运行测试集结果

**2. 基于内容的推荐算法**

对于用户12，为其推荐5部电影。运行结果如图5.3所示。

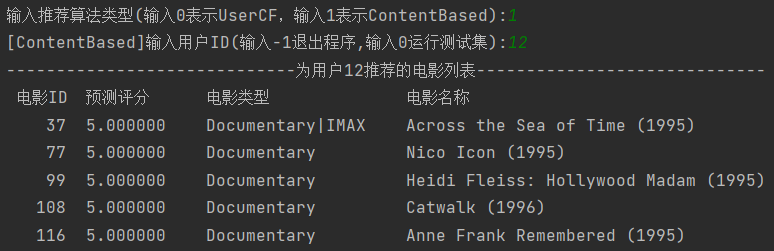


图5.3 基于内容的推荐算法推荐结果

运行测试集的结果如图5.4所示。

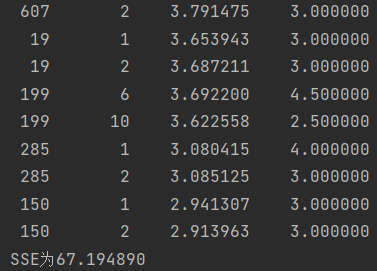


图5.4 基于内容的推荐算法运行测试集结果

## 5.4 实验总结

本次实验中，我掌握了基于用户的协同滤波和基于内容两种推荐算法。这两种算法是实际推荐系统中的重要算法，也各有其优缺点。

基于用户的协同滤波算法基于相似用户推荐电影，能够涵盖不同的喜好，适用于个性化需求不高的场景，也不会导致始终给用户推荐某种内容的极端情况。但是其又有冷启动问题，即当某推荐系统积累数据量过少时将无法向新用户进行个性化推荐。在实际应用中，可以通过根据用户个人信息进行粗粒度的个性化推荐、要求用户填写个人喜好、先基于内容推荐然后根据基于用户推荐等方式解决该问题。

基于内容的推荐算法基于相似电影推荐电影，能够符合用户的个人喜好，适用于个性化需求强烈的场景。该算法不存在冷启动问题，当某个用户产生对某个物品的新行为时，即可根据该行为进行个性化推荐。然而，使用这种推荐算法将会导致推荐内容过于专一，用户只能获得与其自身画像中的项目类似的推荐结果。同时，如何精准刻画内容之间的相似度也是十分重要的。在向用户推荐之前，推荐系统需要存储丰富的内容描述，如推荐电影时，不仅需要考虑电影类型，还需要考虑导演、演员、情节、台词等相关指标。

可见，不同的推荐算法各有优缺点，在实际应用中，需要通过结合不同的推荐算法，进行不断地测试调整优化相关参数来达到更理想的推荐效果。

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)