

華中科技大學

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	1
1.1 实验目的	1
1.2 实验内容	1
1.3 实验过程	1
1.3.1 编程思路	1
1.3.2 遇到的问题及解决方式	2
1.3.3 实验测试与结果分析	2
1.4 实验总结	4
实验二 PageRank 算法及其实现	5
2.1 实验目的	5
2.2 实验内容	5
2.3 实验过程	5
2.3.1 编程思路	5
2.3.2 遇到的问题及解决方式	6
2.3.3 实验测试与结果分析	6
2.4 实验总结	7
实验三 关系挖掘实验	8
3.1 实验内容	8
3.2 实验过程	8
3.2.1 编程思路	8
3.2.2 遇到的问题及解决方式	9
3.2.3 实验测试与结果分析	10
3.3 实验总结	11
实验四 kmeans 算法及其实现	12
4.1 实验目的	12
4.2 实验内容	12
4.3 实验过程	13
4.3.1 编程思路	13
4.3.2 遇到的问题及解决方式	14
4.3.3 实验测试与结果分析	14
4.4 实验总结	15
实验五 推荐系统算法及其实现	16
5.1 实验目的	16

5.2 实验内容	16
5.3 实验过程	18
5.3.1 编程思路	18
5.3.2 遇到的问题及解决方式	21
5.3.3 实验测试与结果分析	22
5.4 实验总结	23

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

实现 9 个分布式节点内容的统计，使用的 map-reduce 方法包括 4 个主要环节，即 map、combine、shuffle 和 reduce。最后，将各统计结果组合为一个 outresult 文件。各环节的逻辑关系图如图 1.1 所示。

map 环节中，将 source 文件中的信息提取出来，形成<word, count>键值对，由于 map 环节只需要提取关键字，故 count 值置 1。source 文件中，以一行一行为间隔，每行的关键字以逗号‘,’分隔，每提取一个关键字(单词)后，以<word,1>的形式记录到对应的 map 文件中。其中的关键在于，9 个分布式节点的 map 环节可以使用并行操作，提高运行效率，如对 source01 文件内容的 map 操作进行

并行线程运行行为: `t1 = threading.Thread(target=run('source01', 'map01'), args=("t1",))`。

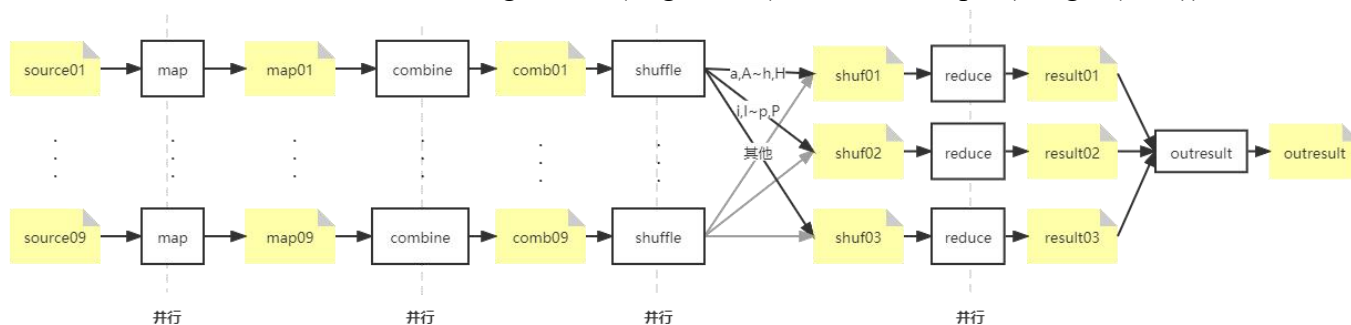


图 1.1 各环节的逻辑关系图

combine 环节中，则是对各 map 文件内预处理好的<word,1>键值对进行统计，将相同的关键字进行累加。实现十分简单，将首次出现的关键字记录到字典中，key 值为 word，value 值为 count，首次记录的 count 为 1。若在后续查找的关键字出现在字典里，则对应的<word,count>里的 count 加 1。同样的，并行各个 combine 操作，最终将字典导入相应的 comb 文件。

shuffle 环节中，则是对 comb 文件内的 word 值进行分类。可以采用首字母分类，a~h 和 A~H 首字母的<word,count>键值对放入 shuf01 文件内，i~p 和 I~P 首字母的<word,count>键值对放入 shuf02 文件内，其他则放入 shuf03 文件内。同样的，对 9 个 comb 文件进行并行 shuffle，最终生成 3 个 shuf 文件。

reduce 环节中，对 shuf 文件内<word,count>进行统计，统计方式同 combine 方式，且 reduce 环节依然能够同步进行。为了方便查看结果，将统计完的<word,count>键值对按 word 进行字典排序，将排序好的字典导入 result 文件。

最后，可以将 3 个 result 进行合并，生成最终文件 outresult，关键步骤同 combine，在此不再赘述。

1.3.2 遇到的问题及解决方式

第一个实验遇到最大的问题是，python 从入门到精通。为此，查阅了许多 python 学习网页，从学习 python 语法到对文件操作，再到多线程的使用，一套下来对 python 有了一知半解。

特别是 python 对文件的操作，与 C 语言有特别大的区别。查找资料时发现了 yield 的使用方法，便尝试用了用。不过现在看来有点小题大做。

1.3.3 实验测试与结果分析

1、测试 map 操作。运行 map.py 对 source 文件进行 map 操作，对每个单词生成<word,1>键值对的 map 文件。生成的 map01 文件部分如图 1.2 所示。



```

map01 - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
nonritualistic,1
freedoot,1
coleslaws,1
argalas,1
Essam,1
Vinnie,1
immaculateness,1
Listerised,1

```

图 1.2 map 操作测试

2、测试 combine 操作。运行 combine.py 对 map 文件进行 combine 操作，统计 word，生成<word,count>键值对的 comb 文件。生成的 comb01 文件部分如图 1.3 所示。



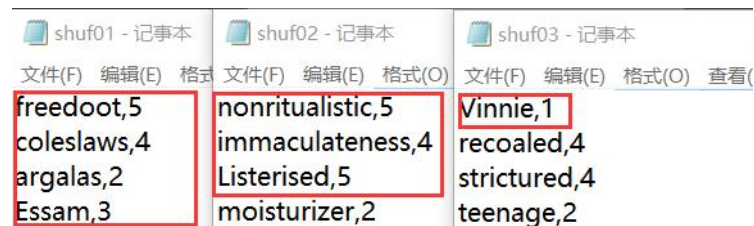
```

comb01 - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
nonritualistic,5
freedoot,5
coleslaws,4
argalas,2
Essam,3
Vinnie,1
immaculateness,4
Listerised,5

```

图 1.3 combine 操作测试

3、测试 shuffle 操作。运行 shuffle.py 对 comb 文件进行 shuffle 操作，将相应的<word,count>键值对放入对应的 comb 文件。具体为，a~h 和 A~H 首字母的放入 shuf01 文件内，i~p 和 I~P 首字母的放入 shuf02 文件内，其他则放入 shuf03 文件。生成的各 shuf 文件如图 1.4 所示，方框内为图 1.3 的对应分类。



```

shuf01 - 记事本
文件(F) 编辑(E) 格式(O)
freedoot,5
coleslaws,4
argalas,2
Essam,3

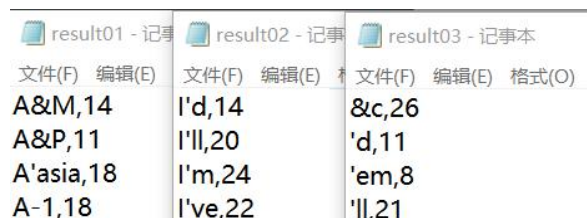
shuf02 - 记事本
文件(F) 编辑(E) 格式(O)
nonritualistic,5
immaculateness,4
Listerised,5
moisturizer,2

shuf03 - 记事本
文件(F) 编辑(E) 格式(O) 查看(V)
Vinnie,1
recoaled,4
strictured,4
teenage,2

```

图 1.4 shuffle 操作测试

4、测试 reduce 操作。运行 reduce.py 对 shuf 文件进行 reduce 操作，统计 word，生成<word,count>键值对的结果文件。生成的各 result 文件部分如图 1.5 所示。



```

result01 - 记事本
文件(F) 编辑(E)
A&M,14
A&P,11
A'asia,18
A-1,18

result02 - 记事本
文件(F) 编辑(E)
I'd,14
I'll,20
I'm,24
I've,22

result03 - 记事本
文件(F) 编辑(E) 格式(O)
&c,26
'd,11
'em,8
'll,21

```

图 1.5 reduce 操作测试

5、测试 outresult 操作。运行 outresult.py，对 3 各 result 文件进行合并，生成最终文件 outresult。生成的 outresult 文件部分如图 1.6 所示。



图 1.6 outresult 操作测试

结果分析：

各个操作结果均满足预期，且最终文件 outresult 按字典顺序排序，满足要求。

1.4 实验总结

本次实验实现了通过 map-reduce 的单词统计工作。map-reduce 方法中，增加了 combine 和 shuffle 操作，进一步分解步骤。让繁琐的统计步骤由多个并行线程完成，在多处理器环境下，大大提高了运行效率。

map-reduce 里的关键技术在于分解与并行，这在分布式数据处理中用途极大，可以大大减轻总数据处理的压力，且各个部件的协同合作又会减少数据的出错次数。

同时分解技术，也让复杂的工作分配到各个小环节上，减轻了单任务机的工作压力。显而易见的，可以进一步进行流水线工作，再次提高工作效率。

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；（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

由预处理完成的 sent_receive.csv 文件中，可以获取各个 sent 与 receive 之间的通信对 $\langle \text{sent_id}, \text{receive_id} \rangle$ 。可以通过信息流通有向图了解具体含义，如图 2.1 所示。其中有 A, B, C 三个用户，有 $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow A$ 的流向，那么可以建立一个归一化矩阵 M 。 M 的每列表示用户的消息流通情况，每一行表示用户接收的情况。

¹ 基本 pagerank 公式: $r = Mr$

² 进阶版 pagerank 公式: $r = \beta Mr + (1 - \beta) \begin{bmatrix} 1 \\ N \end{bmatrix}_{N \times N}$ ，其中 β 为阻尼系数，常见值为 0.85

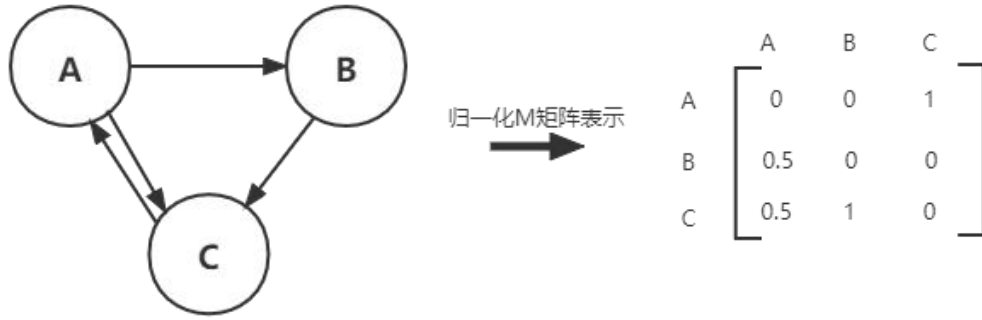


图 2.1 模型基础示例

那个发送方作为有向边的出端点，接收方作为有向边的入端点。首先，统计发送方-接收方的数据对 $\langle \text{sent_id}, \text{receive_id} \rangle$ ，用一个字典存放，在存放过程中判断当前数据对是否在字典内，若在则滤去。同时，将第一次出现的节点加入 `nodes` 列表内。然后根据 `nodes` 的大小 N 建立一个 $N \times N$ 的 0 矩阵。接着，遍历数据对 $\langle \text{sent_id}, \text{receive_id} \rangle$ ，将 $M[\text{nodes.index}(\text{receive_id}), \text{nodes.index}(\text{sent_id})]$ 置 1。然后对 M 矩阵进行归一化，即对 M 矩阵的每一列求和 $\text{sum}(M[:, \text{col}])$ ，再将每一列的元素除以该和数。

为了消除 `dead ends` 和 `spider trap` 带来的影响，引入 β 值，设置其为 0.85。然后构建矩阵 A ， $A = \beta M + (1 - \beta)[1/N]_{N \times N}$ 。建立初始 `pagerank` 矩阵 r 为 $\text{np.ones}((N, 1))/N$ ，即矩阵 $[1/N]_{N \times 1}$ 。

进行迭代过程， $r_{\text{next}} = A r$ ，对 r_{next} 进行归一化，即 $\text{sum_col} = \text{sum}(r_{\text{next}})$ ， $r_{\text{next}} = r_{\text{next}} / \text{sum_col}$ 。直到误差值 $\text{sum}(|r_{\text{next}} - r|) < 10^{-8}$ ，停止迭代。否则，将 r 替换为 r_{next} ，继续迭代。

2.3.2 遇到的问题及解决方式

在做该实验时，需要注意一些小细节。

1、对每个变量初始化的正确性的判断。由于该实验涉及的变量有些多，而且每个变量所表示的含义不同，比如 $[1/N]_{N \times N}$ 与 $[1/N]_{N \times 1}$ 的差异，一个在 A 中，而另一个是 r 的初始值。为了避免变量混乱，可以通过在过程中查看变量值的方法解决。

2、考虑对 r 计算后的 r_{next} 进行归一化操作。这是根据求取误差方法决定的，误差是 r_{next} 与 r 差值后矩阵的绝对值求和，因此需要对 r_{next} 与 r 进行统一规则设置，由 r 的初始化进行归一化可知，后续也需进行 r_{next} 归一化。

2.3.3 实验测试与结果分析

运行 `pageRank.py`，测试 `pagerank` 操作是否正确。

输出迭代次数和 `id: pagerank` 数值对，部分结果如图 2.2 所示。

```
迭代次数: 51
87 : [0.00097877]
80 : [0.10845779]
32 : [0.00195593]
81 : [0.00293308]
185 : [0.00097877]
77 : [0.00097877]
213 : [0.00097877]
194 : [0.00097877]
21 : [0.00097877]
22 : [0.00097877]
```

图 2.2 部分输出结果

结果分析:

最终的误差为 $9.70883494 \times 10^{-9}$, 满足要求。

2.4 实验总结

本次实验的思想分为基础和进阶部分。在基础的 pagerank 里, 只进行 $r = Mr$ 操作。当 M 不满秩时, 会导致出现 rank 为 0 的情况, 且该情况会影响绝大部分用户。导致 M 不满秩的原因则是, dead ends 和 spider trap 的出现, 这些节点的维度一般较低, 且将其加入 M 矩阵中极易导致与其他用户出现线性关系。由线性代数知识可知, 极易导致 M 不满秩。

解决这种方法则需要引入 teleporting, 对于每个结点有 $1 - \beta$ 的概率通过 teleporting 进行 rank 传递。这一机制的引入, 导致 M 矩阵中产生常量矩阵, 对这些 dead ends 和 spider trap 结点进行维度填补, 有极大概率使得 M 满秩, 或只有极小概率与合法用户有线性关系。这样一来, 解决了 dead ends 和 spider trap 吸收 rank 的问题。

上述就是对该实验的一个整体思想过程。在实现的过程中, 也需要对细节有把握, 不在小细节上出错也很十分关键。

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

该实验使用 Apriori 算法实现对给定数据关联规则的分析。要求获取一阶、二阶和三阶的频繁项集，并求出各频繁项集中可信的关联规则。同样梳理一条处理流程介绍编程思路，如图 3.1 所示。

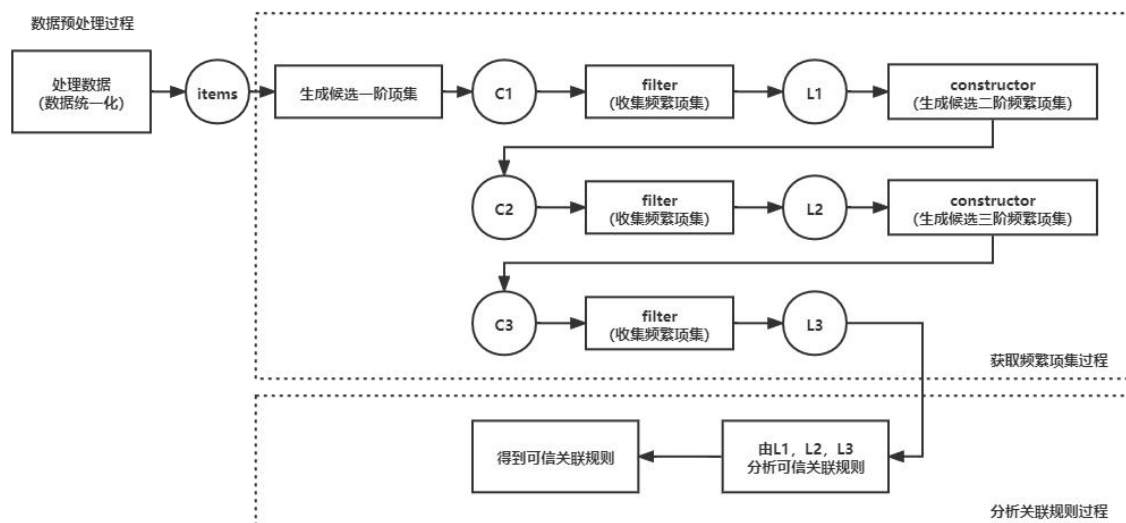


图 3.1 Apriori 算法思路流程

数据预处理阶段：

Groceries.csv 文件内包含所有项集信息，数据预处理阶段则是将其中的内容提取出，用一个统一结构表示出来。采用二级列表保存该数据，如有两个项集 {A,B} 和 {A,C}，那么其二级列表结构表示为[[项 A,项 B],[项 A,项 C]]。设总项集数为 N。

获取频繁项集阶段：

在该阶段中，首先做存放准备，拿 3 个字典存放各阶频繁项集和其支持度，存储结构为<frozenset{项集},支持度>，为后续分析关联规则做准备。值得说明的是，在此用了不可变集合，是为了将其作为字典关键字使用，否则处理会有些麻烦。由此，Ck 与 Lk 都视作集合，在后续不再重复说明。

接着遍历项集列表中每个项，将首次出现的项加入 C1，由此生成 C1。

将 filter 模块化，即通过 Ck 得到 Lk。具体则是对项集列表中各项集进行遍历，查看每个项集中是否含有与 Ck 内项集相同的子集。若有，则对<c,count>中的 count 进行加一，其中 c 为 Ck 中的项集。遍历完所有项集后，对每个<c,count>进行判断，若 $\text{count}/N \geq \text{最小支持度}$ ，则将 c 加入到 Lk 中，将<c,support>加入 Dk 频繁字典中。

将 constructor 模块化，即通过 Lk 得到 Ck+1。过程中，使用两个关键策略：连接策略和剪枝策略。对于连接策略有定理：如果 Lk 中某两个的项集 itemset1 和 itemset2 的前(k-1)个项是相同的，则称 itemset1 和 itemset2 是可连接的。对于剪枝策略有定理：任何非频繁的 k 项集都不是频繁(k+1)项集的子集。可知其逆否命题为任何频繁项集的子集都是频繁项集。若过程生成的 Ck+1_item 中，存在 k 项子集不在 Lk 内，则可将 Ck+1_item 剪掉，不将其加入最后的 Ck+1 中。判断 k 项集是不是(k+1)项子集的做法为：sub_item = temp_item - frozenset([item])，对每个 k 项子集判断 sub_item 是否在 Lk 中，若不在，可直接将其剪掉；若在则继续判断，直到所有子集都是在 Lk 中为止。

在进行 3 次 filter 和 3 次 construct 后，可以获得 L1，L2 和 L3，将其打印。还获得了 L1，L2 和 L3 的支持度字典，用于关联规则分析过程。

分析关联规则阶段：

分析二阶项集和三阶项集的关联规则。对于各阶项集，对项集内的每个项进行判断，若 I 为一个 k 阶项集，i 为项集内的任意一个项，则对每个项判断 $\text{support}(I)/\text{support}(I-\{i\})$ 判断是否不小于最小置信度。若不小于则将关联规则和置信度加入置信关系列表中。最后打印该置信关系列表。

3.2.2 遇到的问题及解决方式

此次实验应该是最难的一次，过程中遇到了两个较大的问题。

1、选择列表结构存放 Ck 和 Lk，还是选择集合存放。最初选择列表结构存

放，但由于 filter 和 constructor 含大量的判断子集过程，列表相对与集合会消耗更多的时间。造成运行时间延长。且在字典存放支持度时，列表无法充当 key 值，而集合中，可以使用 frozenset 不可变集合作为字典 key 值，大大提高了编程效率和运行效率。

2、至关重要的，则是剪枝操作。最初并没有考虑剪枝情况，造成运行时间非常长，再查阅了相关资料后，得知了剪枝的操作过程。大大减小了多余项集的处理时间，这也是 Apriori 算法的关键所在。

3.2.3 实验测试与结果分析

运行 Apriori_improve.py，生成各阶频繁项集 L1, L2, L3，以及频繁项中的关联规则 rule。

L1 中的部分频繁项集如图 3.2 所示。总项集数为 120。



```
L1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
frequent-itemsets, support
['semi-finished bread'], 0.017691916624300967
['margarine'], 0.05856634468734113
['citrus fruit'], 0.08276563294356888
['tropical fruit'], 0.10493136756481952
['yogurt'], 0.13950177935943062
['coffee'], 0.05805795627859685
```

图 3.2 L1 中的部分项集

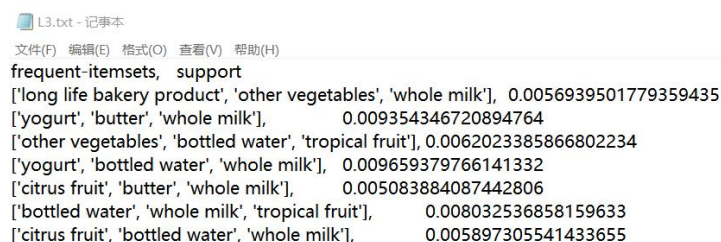
L2 中的部分频繁项集如图 3.3 所示。总项集数为 605。



```
L2.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
frequent-itemsets, support
['citrus fruit', 'margarine'], 0.007930859176410779
['coffee', 'tropical fruit'], 0.0071174377224199285
['yogurt', 'coffee'], 0.009761057447890189
['yogurt', 'tropical fruit'], 0.029283172343670564
['yogurt', 'cream cheese'], 0.012404677173360447
['cream cheese', 'pip fruit'], 0.006100660904931368
```

图 3.3 L2 中的部分项集

L3 中的部分频繁项集如图 3.4 所示。总项集数为 264。



```
L3.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
frequent-itemsets, support
['long life bakery product', 'other vegetables', 'whole milk'], 0.0056939501779359435
['yogurt', 'butter', 'whole milk'], 0.009354346720894764
['other vegetables', 'bottled water', 'tropical fruit'], 0.0062023385866802234
['yogurt', 'bottled water', 'whole milk'], 0.009659379766141332
['citrus fruit', 'butter', 'whole milk'], 0.005083884087442806
['bottled water', 'whole milk', 'tropical fruit'], 0.008032536858159633
['citrus fruit', 'bottled water', 'whole milk'], 0.005897305541433655
```

图 3.4 L3 中的部分项集

满足最小置信度的关联规则存放在 rule 文件中。部分关联规则和置信关联规则总数如图 3.5 所示。置信关联规则总数为 99。

```
['pastry', 'root vegetables'] ==> ['other vegetables']: 0.537037037037037
['yogurt', 'frankfurter'] ==> ['whole milk']: 0.5545454545454546
['yogurt', 'fruit/vegetable juice'] ==> ['whole milk']: 0.5054347826086957
['root vegetables', 'sausage'] ==> ['whole milk']: 0.5170068027210885
['other vegetables', 'whipped/sour cream'] ==> ['whole milk']: 0.5070422535211268
['onions', 'root vegetables'] ==> ['other vegetables']: 0.6021505376344086
['yogurt', 'whipped/sour cream'] ==> ['whole milk']: 0.5245098039215685
['baking powder'] ==> ['whole milk']: 0.5229885057471264
total:99
```

图 3.5 部分关联规则和置信关联规则总数

结果分析：

生成的 L1, L2 和 L3 中的频繁项集的支持度均不小于 0.005。生成的关联规则 rule 的置信度均不小于 0.5。满足要求。

3.3 实验总结

对于此次实验，有很大的感悟，感受到了剪枝的强大之处。在剪枝前，运行时间十分漫长，剪枝后，非常顺畅。剪枝也是 Apriori 算法的灵魂所在。

除此之外，也学习到了关联规则的生成过程。在此之前，只是感官上的感受啤酒和面包的关系，而不知道是如何得到其中关联规则的。而通过此实验，学习了一个关联方法，之后就可以处理许多类似的事件。这种有规则的数据处理比起直觉而言，更具有信服度。

另外，对于 python 的各存储结构的区别有了更加深刻的理解。此前对列表和字典使用较多，此实验则使用到了集合结构，有了更加深刻的理解。列表对于相同的类型数据处理方便，而集合则对数据及包含在数据内的属性之间的联系的处理更加方便，如子集，包含等。

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 (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 csv 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。

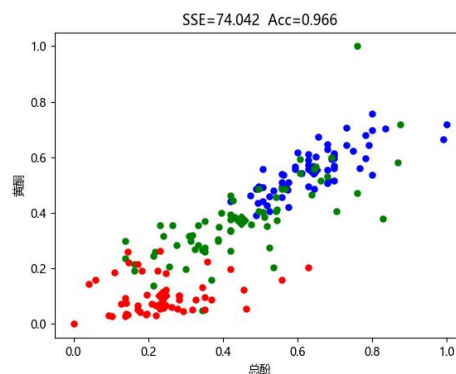


图 4.1 葡萄酒数据集在黄酮和总酚维度下聚类图像 (SSE 为距离平方和, Acc 为准确率)

4.3 实验过程

4.3.1 编程思路

该实验使用 `kmeans` 算法，进行聚类 and 测试。进行 3 个质心的聚类操作，具体流程图如图 4.2 所示。

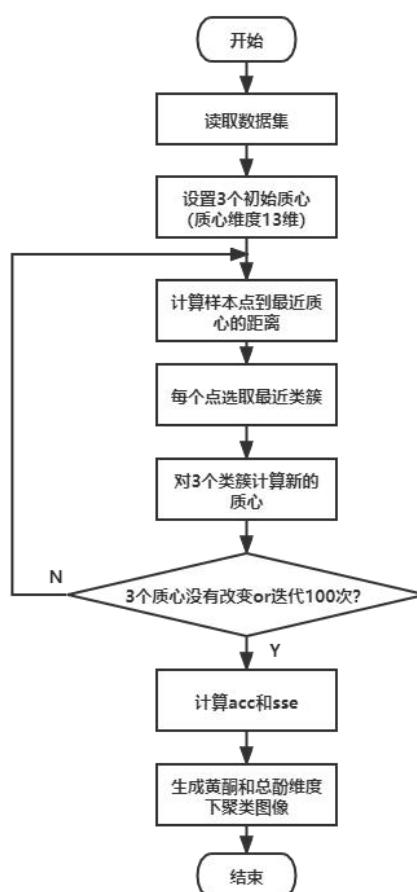


图 4.2 `kmeans` 聚类算法的具体实现流程图

首先进行数据集的处理，使用二维列表存放每个样本点的数据，具体存放结构为[[类型,属性 1,属性 2,...,属性 13],[类型,属性 1,属性 2,...,属性 13],...], 存放类型是为了方便后续计算准确度。

设置 `k=3`，即初始化 3 个随机质心，使用 `random` 进行 13 维数据的获取。由于，原始数据已经进行了归一化，故可将质心各维度数据随机在 0~1 间，这也是归一化的目的所在。

随后进行类簇迭代循环。在循环前，建立二维数组，存放每个样本点的距所属质心的最小距离平方和以及所属类簇编号。

循环中，设置 `flag` 标志，每次循环置 `flag` 为 0，若在出现修改二维数组的情况，则表明质心还在改变，置 `flag` 为 1。同时也设置计数器 `count`。当 `count` 大于

100 或 flag 为 0 时，退出迭代循环。

接着计算 acc 和 sse。计算 sse 思路简单，因为上述二维数组中记录了相关每个质心的所属类簇和最小距离平方和，故可直接算出 3 个类簇中样本点的距离平方和以及所有样本点的距离平方和。

对于 acc 的计算，基于如图 4.3 所示的思想。



图 4.3 计算 acc 的具体思想

其中每个圈代表一个类簇，类簇中不同的图形代表样本点的实际标签。将类簇中最多的那类图形视作该类簇的掌控者。进而计算该类簇中掌控者的比例为该类簇的准确度 acc，统计 3 个类簇的 acc 即可得到总的准确度。

最后进行进阶部分的画图。使用 matplotlib.pyplot 库函数中的 scatter 函数进行绘制，由于准备了保存每个样本点所属类簇的二维数组。为每个类簇内样本点分配相同的颜色，共 3 种颜色。横坐标为样本点的总酚数据，纵坐标为样本点的黄酮数据。

4.3.2 遇到的问题及解决方式

该实验比较简单，不过仍然考察各种细节。比如在进行类簇中心质心的计算时，需要获取类簇样本点，在进行双重循环时注意 i, j 所在循环层级。由于粗心导致了一些错误。

另外，在处理数据中的类型和所属类簇时，容易将二者混淆，弄混两者的含义，因此要足够的仔细。特别是在计算 acc 时，所属类簇为外层大圈，类型则为判断依据。通过画图的方式就很好理解了。

4.3.3 实验测试与结果分析

运行 kmeans.py，计算各类簇中样本点的距离平方和，总距离平方和 sse 和准确度 acc，如图 4.4 所示。结果表明迭代了 8 次，总 sse 为 48.97，acc 为 0.94。

```
run times: 8
the 1 12.59183009348725
the 2 20.872346954076328
the 3 15.50611410757559
ALL 48.97029115513917
acc: 0.9438202247191011
DONE
```

图 4.4 计算 sse 和 acc 结果

葡萄酒数据集在黄酮和总酚维度下聚类图像如图 4.5 所示。

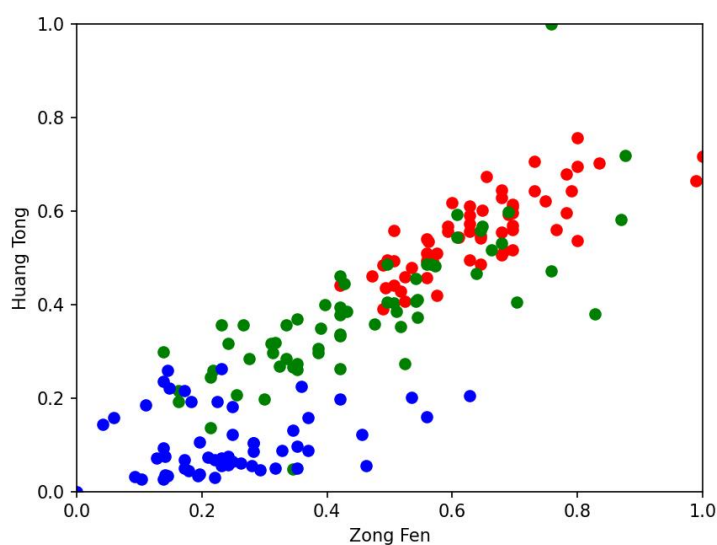


图 4.5 黄酮和总酚维度下聚类图像

结果分析：

根据最终的结果可知，sse 小于 60，acc 大于 0.90，sse 和 acc 达到要求。根据黄酮和总酚维度下聚类图像与示例图比较，相似度高，满足要求。

4.4 实验总结

首先分析为什么需要归一化的问题。第一点，在随机分配质心时，需要随机维度数据，若进行归一化，则可明确在 0~1 之间随机，若不进行归一化，则数据间隔太大，随机维度数据的间隔不好把控，具有极大的误差；第二点，不进行归一化，若选取最大数据与最小数据间为随机间隔，一般间隔很大，那么在迭代循环时，需要很多次迭代才能保证质心稳定，效率太低。

在 python 中运用了二维数组这一结构，对 python 的各结构都熟悉了一遍，完成了 python 的入门，并且还了解和使用 python 强大的画图功能。

对于 kmeans 算法，了解了聚类这一概念和具体的运用范畴。在计算 acc 时，学会了如何使用标签进行精确度的分析。

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 **jaccard** 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：

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

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， $\text{score}'(i)$ 为计算集合中第 i 个电影的分数， $\text{sim}(i)$ 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 **jaccard** 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。

备注：此实验报告作为最终结课报告，要求至少 4000 字

5.3 实验过程

5.3.1 编程思路

对于推荐系统，有两种具体分析思路，即以用户为中心和以内容为中心，两种方式的区别示例图如图 5.1 所示。其中，以用户为中心则是根据不同用户喜欢物品的情况，寻找相似的用户，将对应喜欢的物品推荐给目标用户。而以内容为中心，则是根据物品所带有的标签属性，寻找相似的物品，根据目标用户喜爱的物品推荐给其对应的相似物品。

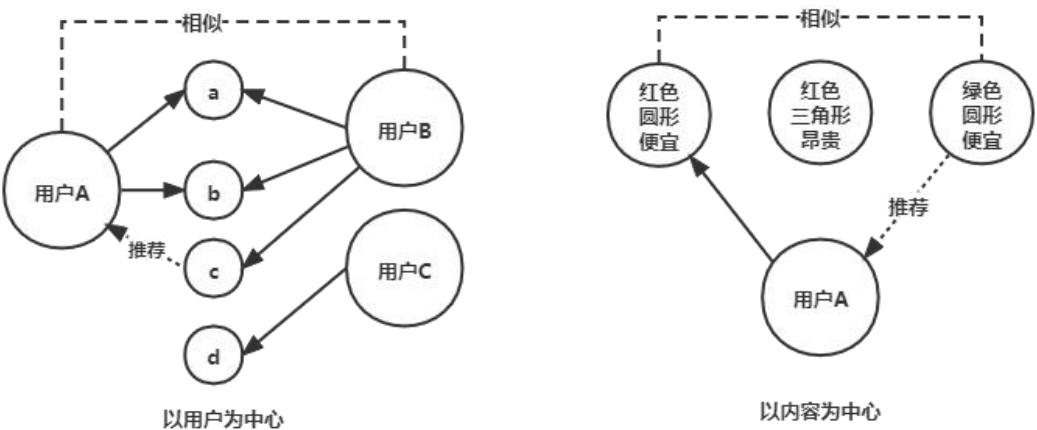


图 5.1 以用户为中心和以内容为中心的示例图

而此次实验包含的两个内容涵盖了这两种方面。基于用户的协同过滤推荐系统是以用户为中心，查找相似用户，对目标用户的未看电影进行预测评分，推荐给目标用户分高的电影。基于内容的推荐系统则是以内容为中心，查找电影间的相似关系，根据既定标签：电影的类型进行相似度分析，由目标用户已看电影对未看电影进行预测评分。两者的具体思路如下：

1. 基于用户的协同过滤推荐系统

由上述分析，可以将思路梳理为如图 5.2 所示的流程图。

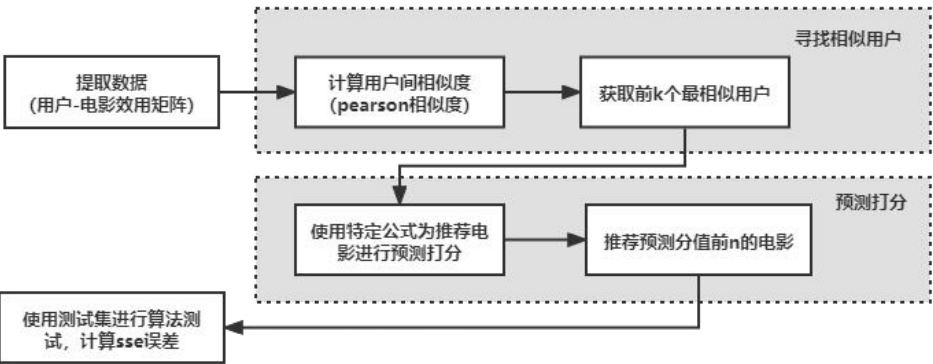


图 5.2 基于用户的协同过滤推荐系统流程图

首先进行数据的提取,目的是生成用户-电影效用矩阵。对训练集 train_set.csv 进行提取,训练集中的数据属性分为 4 类 userId, movieId, rating 和 timestamp, 其中有效信息为 userId, movieId 和 rating。每一行数据为一个用户-电影效用对, 效用对的数值为 rating 值。那么据此,遍历每一行,使用列表记录首次出现的用户与电影,由于用户 ID 是顺序存放的,故 users 列表中的索引值与用户 ID 有线性对应关系,后续可直接使用索引即可;而电影 ID 出现次序不定,故存放在 movies 列表内的电影 ID 与索引值无特定关系,故在后续查找中,需要使用 index 函数进行查找。设 users 长度为 N, movies 长度为 M, 建立 N×M 的初始化矩阵 user2movie 作为用户-电影效用矩阵。而用户-电影效用对的分值 rating, 则作为 user2movie[userId-1,movies.index(movieId)]的数值。

构造完用户-电影效用矩阵 user2movie 后,对用户进行相似度分析,使用 pearson 相似度,公式如下:

设用户 A 对 m 部电影的评分为 x_1, x_2, \dots, x_m ; 用户 B 对电影的评分为 y_1, y_2, \dots, y_m , 用户 A 与用户 B 的相似度:

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

计算后获得一个 N×N 的 pearson 相似度矩阵,用来表明各用户间的相似度关系,矩阵中的[x,y]表示用户 x 与用户 y 的相似度。通过设定目标用户的 My_userid, 为到相似度矩阵中的对应的行,即第 My_userid-1 行,剔除掉用户自己,进行排序。选取排名前 k 个用户,称其为“邻居”。

接着通过邻居的电影打分值,对目标用户未看电影进行预测评分。但出现问题,当对一个未看电影进行评分时,若邻居也未看该电影,该如何进行预测呢? 我们考虑将邻居的分值进行平均偏移调整,即获取各邻居已看的电影分值的平均值,对已看电影评分进行改变,即用已看的电影分值减去平均分,为目标分值。则有略喜欢看的电影分值为正,未看电影分值为 0, 略不喜欢看的电影分值为负,进而实现对分值预测的合理性。相关公式如下:

设用户 x 对未看电影 m 的预测评分为 $\tilde{R}_{x,m}$, 而用户 x 已看电影的平均分值为 \overline{R}_x , 用户 x 的 k 个邻居为 x_1, x_2, \dots, x_k , $\text{sim}(x, x_i)$ 表示 x 与 x_i 的相似度,假设所有邻居都看过该电影,则对 x-m 效用对分值的预测公式如下:

$$\tilde{R}_{x,m} = \overline{R}_x + \frac{\sum \text{sim}(x, x_i)(R_{x_i,m} - \overline{R}_{x_i})}{\sum \text{sim}(x, x_i)}$$

特别的,当 $R_{x_i,m}$ 为 0 时,表明 x_i 用户并未看过该电影,根据上述平均偏差规则, $R_{x_i,m} - \overline{R}_{x_i}$ 为 0, 即舍弃未看过该电影的邻居。

最终,对用户 My_userid 进行所有未看电影的预测评分,将预测评分进行排

序。选取前 n 部电影，即为推荐电影。另外，为了将电影 ID 与电影名称对应，需要读取 movies.csv，进行查找，再次不加赘述。

最后，进行测试集的 sse 计算。sse 计算公式如下：

$$sse = \sum (r_{x,i} - \tilde{r}_{x,i})^2$$

为了便于测试，在原查找前 n 部电影的基础上进行修改。由于测试机里每个用户只需要预测两部电影评分，则直接对两部电影进行 x-m 效用对分值的预测公式计算即可，则无需对测试用户的所有未看电影进行评分了。

2. 基于内容的推荐系统

基于内容的推荐系统则是以电影内容为中心，即分析电影标签：电影的类别。获取电影间的相似度，根据目标用户已看电影，进行对未看电影的预测打分。思路梳理为如图 5.3 所示的流程图。

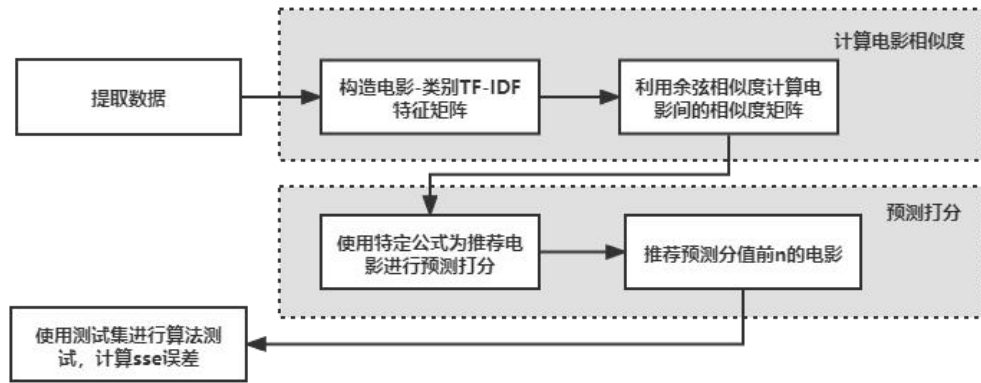


图 5.3 基于内容的推荐系统流程图

首先对 movies.csv 进行数据的提取，获得 movie-genres 效用对列表 movie_genres，其中记录着每个电影对应的类型，于此同时也获取电影列表 movies 和类型列表 genres，存放出现的电影与类型。设 movies 长度为 M ，genres 长度为 N 。

接着进行 TF-IDF 特征矩阵的构建。首先构建一个 $M \times N$ 的初始化矩阵。TF 表示一个电影中，各类型出现的次数占比。由于一部电影中，只会含有不重复的类型，故可先置[movies,genre]为 1，再除以该电影的标签数，则为每部电影中所含标签的 TF 值。接着进行 IDF 值的计算，对于每个标签的 IDF 值，为总电影数除以电影中出现该标签的次数，再取对数。对每个标签，即一行，乘以对应的 IDF 值，即可得到最终的 TF-IDF 特征矩阵。

使用余弦相似度计算各电影间的相似度，构成 $M \times M$ 的相似度矩阵 sim。具体公式如下：

设有电影 A 的每个标签的 TF-IDF 值为 x_1, x_2, \dots, x_n ；电影 B 的每个标签的 TF-IDF 值为 y_1, y_2, \dots, y_n 。那么 A 与 B 的相似度计算公式为：

$$cossim(A, B) = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2} \sqrt{\sum y_i^2}}$$

接着进行预测打分。对未看电影以权值进行打分，此时权值为电影间的相似度。设用户 A 对未看电影 y 打分，其中已看电影为 x_1, x_2, \dots, x_n 。 $cossim(y, x_i)$ 为权值， r_{x_i} 为 A 对 x_i 的评分， \tilde{r}_y 为对未看电影 y 的预测评分，有以下预测公式：

$$\tilde{r}_y = \frac{\sum cossim(y, x_i) \cdot r_{x_i}}{\sum cossim(y, x_i)}$$

特别的，当 $\sum cossim(y, x_i)$ 为 0 时，不予考虑推荐该电影。求和的相似度为 0，大概率是用户 A 已看电影中没有与 y 电影相关的电影，即 $cossim(y, x_i)$ 为 0，那么既然不相关，则可不作推荐考虑。

对目标用户 My_userid 所有未看电影进行预测。值得注意的一点是，为了便于搜索已看电影与未看电影，还需建立用户-电影效用矩阵，方便对未看电影的查找。对预测分值进行排序，排名前 n 的电影即为推荐电影。为了将电影 ID 与电影名称对应，查找 movies 与 movie_title 和 movie_genres 的对应。

最后，进行测试集的 sse 计算。同样的，为了便于测试，在原查找前 n 部电影的基础上进行修改。由于测试机里每个用户只需要预测两部电影评分，则直接对两部电影进行预测公式计算即可，则无需对测试用户的所有未看电影进行评分了。

5.3.2 遇到的问题及解决方式

1. 基于用户的协同过滤推荐系统

在进行基于用户的协同过滤推荐系统的编写时，由于画出了流程图，思路十分顺畅，所以整体无太大问题。

在细节处，仍有许多值得注意的几点：

(1) 由于用户-电影效用矩阵中， movies.csv 中的电影并非全部出现过，那么在遍历训练集获取的 moviesId 必然是杂乱的，且获取到的 movies 列表的索引与 movies.csv 中各电影的索引也必然不相同。这就会出现 moviesId 与 moviesIndex 的混淆，不过通过画图自然而然就可以解决这一问题。

(2) 在对未看电影进行分值预测时，由于存在部分邻居未看的情况，然后考虑到使用平均偏移的思想，处理完后的分值有正有负还有 0，这里就需要对 0 进行了处理，0 表示未看，那么可以对该用户进行忽略，不算进权值。但若考虑算入权值中，影响也不太大。

(3) 测试集测试的优化。由于测试集有 100 个测试用户，那么最多需要运行 100 次推荐系统。这样一来，若采用原推荐 n 电影方式，会对每个用户计算未看电影的预测分值，但绝大部分分值我们是不需要的。故考虑对推荐模块做微小

修改，只进行需要测试的电影推荐即可。

2. 基于内容的推荐系统

基于内容的推荐系统相对于基于用户的协同过滤推荐系统编写起来就更为简单了，因为通过流程图可以看出，两者的编码风格很相似，只是要求的相似对象不同。在原问题基础上，还需要注意以下几点：

(1) 基于内容的推荐系统使用权值预测，其中分母为相似度之和，会出现分母为 0 的情况。大概率而言，分母为 0 表示求和项 $\text{cossim}(y, x_i)$ 都为 0，即已看电影与未看电影不相关，那么可以不做推荐考虑。

(2) 在进行相似度计算时，推荐使用库函数相似度计算，使用自己编写的计算模块，运行很慢，效率很低，那么合理运用高效的资源不失为一种有效的方法。

5.3.3 实验测试与结果分析

1. 基于用户的协同过滤推荐系统测试

考虑用户 id 为 29，相似邻居数 k 为 30，推荐电影数 n 为 10 的测试用例，测试结果如图 5.4 所示。

```
1 Fight Club (1999) Action|Crime|Drama|Thriller
2 Shawshank Redemption, The (1994) Crime|Drama
3 Memento (2000) Mystery|Thriller
4 Godfather, The (1972) Crime|Drama
5 Star Wars: Episode IV - A New Hope (1977) Action|Adventure|Sci-Fi
6 Forrest Gump (1994) Comedy|Drama|Romance|War
7 Lord of the Rings: The Two Towers, The (2002) Adventure|Fantasy
8 Pulp Fiction (1994) Comedy|Crime|Drama|Thriller
9 Snatch (2000) Comedy|Crime|Thriller
10 American Beauty (1999) Drama|Romance
sse: 64.32525187703463
```

图 5.4 基于用户的协同过滤推荐系统基础测试截图

结果分析：

推荐的前 10 部电影符合预期。测试集的 sse 误差平方和为 64.33，满足要求。

2. 基于内容的推荐系统测试

考虑用户 id 为 29，推荐电影数 n 为 10 的测试用例，测试结果如图 5.5 所示。

```
1 Death on the Staircase (Soupçons) (2004) ['Crime', 'Documentary'] 5.000000000000001
2 Cocaine Cowboys II: Hustlin' With the Godmother (2008) ['Crime', 'Documentary'] 5.000000000000001
3 Two Escobars, The (2010) ['Crime', 'Documentary'] 5.000000000000001
4 Bus 174 (Ônibus 174) (2002) ['Crime', 'Documentary'] 5.000000000000001
5 Art of the Steal, The (2013) ['Crime'] 5.0
6 American Crime, An (2007) ['Crime'] 5.0
7 Chan Is Missing (1982) ['Crime'] 5.0
8 Killers, The (1946) ['Crime', 'Film-Noir'] 5.0
9 You Only Live Once (1937) ['Crime', 'Film-Noir'] 5.0
10 Enforcer, The (1976) ['Crime'] 5.0
sse: 67.0679282687671
```

图 5.5 基于内容的推荐系统基础测试截图

结果分析：

推荐的前 10 部电影符合预期。测试集的 sse 误差平方和为 67.07，满足要求。

5.4 实验总结

通过此次实验的编写，让我了解了推荐系统的两种实现方法。其一则是基于用户，其二则是基于内容。对两个进行实际编写，感觉基于内容似乎更加可靠，因为对内容的把握一定程度上具有客观性，物品的属性是不可变的，当然可以通过物品之间的相似属性进行推荐也具有一定的客观性。但基于用户而言，用户是人，多多少少具有多样性、复杂性和主观性，里面包含的因素更加复杂，需要考虑的因素也十分繁多，且多重因素的干扰后做出的推荐可能也会因为主观性的介入，显得有些不尽人意。

另外，实现一个推荐系统并进行推荐十分有趣，在日常生活中十分常见且有用。通过该实验，学习到了对数据集的收集和处理，通过有关算法进行相似度分析，根据特定公式计算获取推荐分数。并且可扩展性极强，不仅可以用于推荐系统，还可以用于一些评分系统等。

这次实验也可算作一个大杂烩，里面包含对文件的操作，数据的处理，以及各种结构数据与索引的关系等，都有所考察。通过此次实验的编写，我对 python 的了解又上了一层楼。