

**课 程 实 验 报 告**

**课程名称： 大数据分析**

**专业班级： 大数据2101班**

**学 号： U202115652**

**姓 名： 李嘉鹏**

**指导教师： 王蔚**

**报告日期： 2023年5月11日**

**计算机科学与技术学院**

**目录**

[实验三 关系挖掘实验 1](#_Toc135394670)

[**3.1 实验内容** 1](#_Toc135394671)

[**3.2 实验过程** 1](#_Toc135394672)

[3.2.1 编程思路 1](#_Toc135394673)

[3.2.2 函数模块定义 6](#_Toc135394674)

[3.2.3 数据结构与存储形式 7](#_Toc135394675)

[3.2.4 遇到的问题及解决方式 8](#_Toc135394676)

[3.2.5 实验测试与结果分析 8](#_Toc135394677)

[**3.3 实验总结** 13](#_Toc135394678)

[**3.4 参考文献** 14](#_Toc135394679)

# 实验三 关系挖掘实验

## **3.1 实验内容**

**必做：**

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

**加分项：**

1. 实验内容

在Apriori算法的基础上，要求使用pcy或pcy的几种变式multiHash、multiStage等算法对二阶频繁项集的计算阶段进行优化。

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

输出pcy或pcy变式算法中的vector的值，以bit位的形式输出

 参数不变，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

本次实验要求实现Apriori算法，以及其进阶的PCY算法（利用Multihash、Multistage完成）。两种算法的基本流程都是不断迭代生成候选频繁项集和频繁项集，最终生成关联规则，但在构造频繁项集的过程中，使用PCY算法可以优化计算效率。

**（一）Apriori算法编程思路**

首先，根据Apriori算法，我需要通过候选频繁一项集C1构造频繁一项集L1，并根据频繁一项集L1构造出候选频繁二项集C2，依次类推逐步构造出L2、C3、L3。最后，需要在L1、L2、L3中找出全部的关联规则。整体算法流程图如图3-1所示。

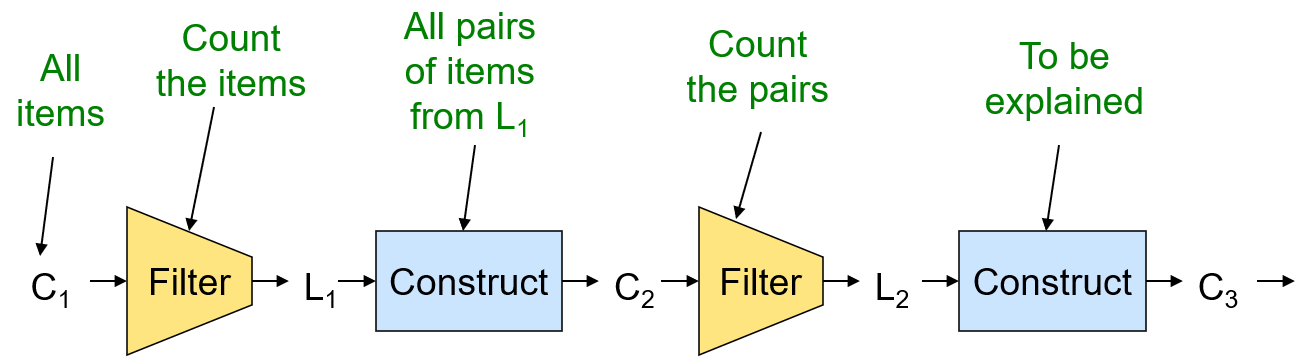


图3-1：Apriori算法流程图

在具体实现上，首先通过读取原始的Groceries.csv数据集（如图3-2所示）得到全部的C1，然后分别统计一项集中每个元素的标准化支持度，若support≥0.005（本次实验中规定的最小支持度），则将其视为频繁项集L1中的元素。对C1中每个元素遍历一次，并统计其support值即可。标准化支持度的计算方法是：

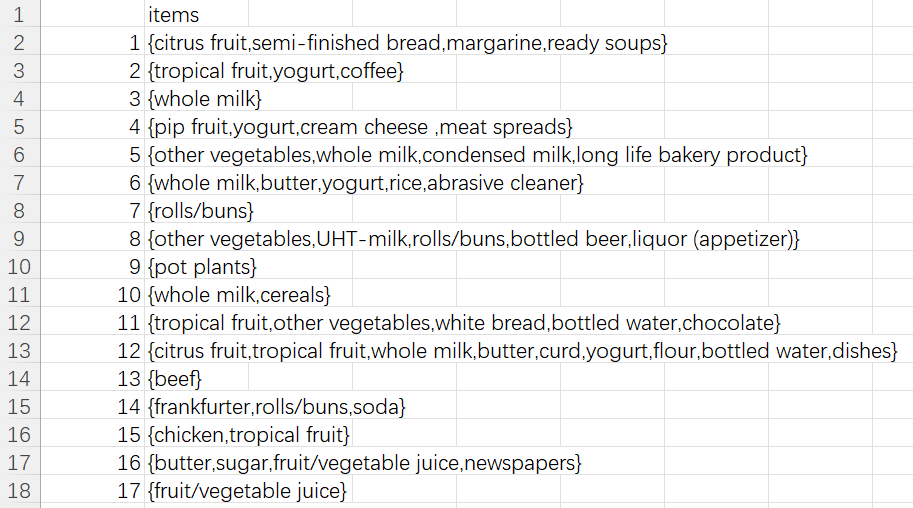


图3-2：原始数据集部分预览

接下来，要根据L1构造C2。我对L1中的元素进行两两组合（例如由得到新的集合），并对组合后的大量可能集合进行了检验，以确保进入C2的集合都是合法的。同理，对C2中的每个元素（集合）遍历统计其support值，并判断能否进入L2。

在根据L2构造C3的过程中，采用了比较直接的方式，即两两比较L2中的集合，若两个集合中只有一个元素不同，那么取这两个集合的并集，就能得到含有三个元素的集合。这个过程中需要采取剪枝策略提高算法的性能，主要是依据Ck+1中每一个集合的子集都在Lk中的原理，对每个三元组依次去除其中任意一个元素，并判断剩下的二元组是否在L2中。若出现一个二元组不在L2中，那么说明这个三元组不能加入C3。实现构造功能的代码如下。

|  |
| --- |
| **def** is\_apriori(ck\_item, Lk): *#在从Lk构造CK+1的过程中，判定一个项集ck\_item是否在Ck+1中* **for** item **in** ck\_item:  sub\_item = ck\_item - frozenset([item])  **if** sub\_item **not in** Lk:  **return False  return True**  **def** Create\_Ck(Lk, k): *# 通过合并Lk-1中的项，获得Ck中的项* Ck = set()  len\_Lk = len(Lk)  list\_Lk = list(Lk)  **for** i **in** range(len\_Lk):  **for** j **in** range(i + 1, len\_Lk):  l1 = list(list\_Lk[i])[0:k - 2]  l2 = list(list\_Lk[j])[0:k - 2]  l1.sort()  l2.sort()  **if** l1 == l2:  Ck\_item = list\_Lk[i] | list\_Lk[j]  **if** is\_apriori(Ck\_item, Lk): *#如果只含k-1个元素或含有大于k+1个元素，则会被直接筛掉* Ck.add(Ck\_item)  **return** Ck |

最后，利用关联规则的置信度公式，得出全部的有效规则。其中，置信度的计算方法是：



其中I是n元频繁项集，j是与I有关系的单个货物。

若某条规则满足conf≥0.5（本次实验规定的最小置信度），则将其视为有效规则并加入关联规则集合中。实现这部分功能的代码如下。

|  |
| --- |
| **def** get\_Rule(L, support\_data, min\_confidence):*#获取关系规则  # 参数：所有的频繁项目集（L1~L3），项目集-支持度dic，最小置信度* rule\_list = []  sub\_set\_list = []  **for** i **in** range(len(L)):  **for** frequent\_set **in** L[i]:  **for** sub\_set **in** sub\_set\_list:  **if** sub\_set.issubset(frequent\_set): *# 寻找上一层循环中出现的frequent\_set的子集* conf = support\_data[frequent\_set] / support\_data[sub\_set] *# conf(rule)=S(J)/S(J-j)* rule = (sub\_set, frequent\_set - sub\_set, conf)  **if** conf >= min\_confidence **and** rule **not in** rule\_list: *#找出大于最小置信度的规则* rule\_list.append(rule)  sub\_set\_list.append(frequent\_set)  **return** rule\_list |

**（二）PCY算法（Multihash）编程思路**

PCY算法相比Apriori算法最大的不同就是利用哈希桶的方式，快速过滤了大量不可能成为频繁项集的候选项集（它们被哈希到不频繁的桶中），同时也不会漏掉真正的频繁项集（它们被哈希到频繁的桶中）。在算法实现上，只需要对二阶频繁项集的计算阶段进行优化，此时需要建立两个哈希表（分别对应两个不同的哈希函数）。接下来分别利用两个哈希函数mmh3.hash()和hash()计算各个候选二项集的哈希值，并将这个哈希值所对应的vector矢量加1。其中，mmh3.hash()属于一种非加密型哈希算法MurmurHash3，它一般用于哈希检索操作，当前版本是MurmurHash3，能够产生出32-bit或128-bit的哈希值。这种哈希算法对于随机分布特征表现更加优良，因此这次我将其作为PCY算法中的一个哈希函数。在判断每个二项集能否进入L2时，只需要判断其所在的哈希桶的support值是否大于等于最小支持度即可。哈希桶的支持度计算方法是：

若某个二项集对应的两个哈希桶的support均大于最小支持度，则将其放入candidate集合中，否则该二项集不能放入candidate集合中。这是因为若该二项集要成为频繁二项集，那么必须满足三个条件：第一，二项集中的两个元素都是频繁的，这一点由L1可以保证；第二，二项集在第一个哈希函数下被分到的哈希桶是“频繁”的；第三，二项集在第二个哈希函数下被分到的哈希桶是“频繁”的。只要有一个哈希桶不是“频繁”的，则其中的所有二项集都不可能是频繁的。利用哈希桶更新candidate集合的代码如下（以哈希函数mmh3.hash()为例）。

|  |
| --- |
| **if** k == 2: *#用Multihash对二阶频繁项集的计算阶段进行优化  # 第一次哈希，使用mmh3.hash函数* hash\_table1 = [0] \* hash\_size  **for** transaction **in** data:  **for** itemset **in** candidates:  **if** set(itemset).issubset(set(transaction)):  hash\_value = abs(mmh3.hash(str(tuple(itemset)))) % hash\_size  *#将itemset转变成字符串并获取哈希值* hash\_table1[hash\_value] += 1  filtered\_candidates1 = [] *#记录可能为频繁项集的candidates* **for** itemset **in** candidates:  hash\_value = abs(mmh3.hash(str(tuple(itemset)),seed=12345)) % hash\_size  *#加入seed，但hash的效果没有区别* **if** hash\_table1[hash\_value]/len(data) >= min\_support: *#标准化* filtered\_candidates1.append(itemset)  candidates = filtered\_candidates1 *#更新candidates* **else**:  candidates = sorted(candidates) *#保持不变* |

最后，只需要再从candidate中选出L2，即可降低算法运行时间。整体的算法流程如图3-3所示。

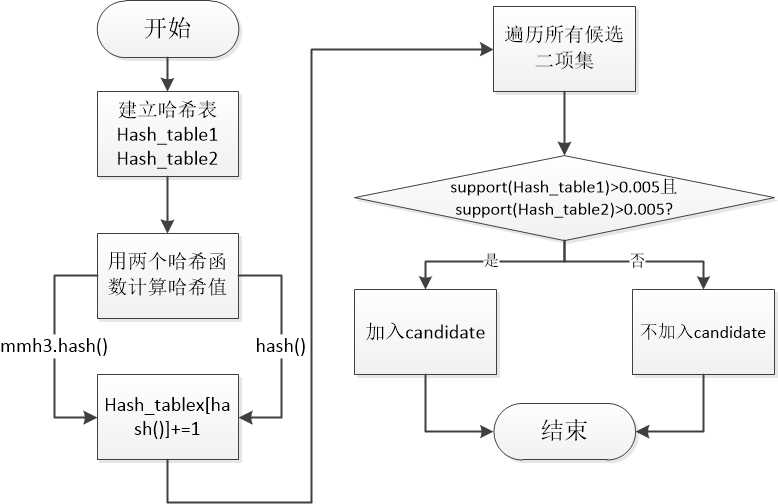


图3-3：PCY（Multihash）算法流程图

采用Multihash可以大幅度减少存储空间的需求，只需要保存两个vector矢量即可，如图3-4所示。在Pass 1中，包含了所有元素的总数以及两个哈希表。在Pass 2中，仅需存储所有的频繁项集、candidate集合中元素的总数以及两个哈希表对应的两个vector矢量，节省了大量内存空间。

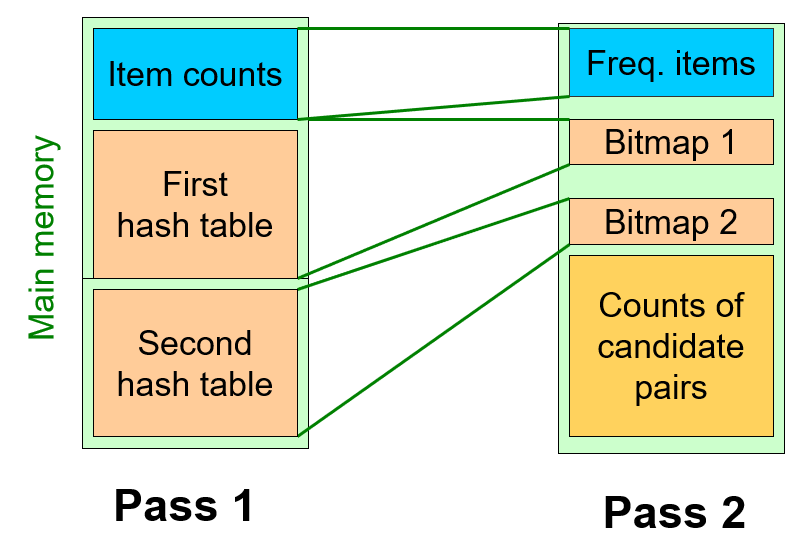


图3-4：Multihash算法的存储结构示意图

### 3.2.2 函数模块定义

**（一）Apriori算法函数模块**

**（1）data\_set**

①函数参数：无

②函数功能：读入Groceries.csv中的数据并存储。

**（2）Create\_C1**

①函数参数：data

②函数功能：根据data构造初始的候选频繁一项集C1。

**（3）is\_apriori**

①函数参数：ck\_item, Lk

②函数功能：在从Lk构造C(k+1)的过程中，判定一个项集ck\_item是否在C(k+1)中。这里依据的原理是C(k+1)每一个集合的子集都在Lk中。

**（4）Create\_Ck**

①函数参数：Lk, k

②函数功能：通过合并L(k-1)中的项，构造产生Ck中的项。

**（5）get\_Lk**

①函数参数：data\_set, Ck, min\_support, support\_data

②函数功能：从候选频繁k项集Ck中筛选出频繁k项集Lk。

**（6）get\_Rule**

①函数参数：L, support\_data, min\_confidence

②函数功能：获取最终的全部关系规则。

**（二）PCY算法（Multihash）函数模块**

**（1）generate\_candidates**

①函数参数：items, k

②函数功能：与前面的Create\_Ck函数一致，产生candidate集合。

**（2）get\_frequent\_itemsets\_pcy**

①函数参数：data, min\_support, hash\_size

②函数功能：利用两次哈希产生频繁项集，对二阶频繁项集的计算进行优化。

### 3.2.3 数据结构与存储形式

**（1）data**

①类型：二维列表，如[[item1, item2, …], [item3, item4, …], … ]

②作用：保存从Groceries.csv中读入的数据，每个一维列表保存初始数据集中一行中第二列的每个item。

**（2）C1、L1、C2、L2、C3、L3**

①类型：集合（frozenset）

②作用：Ck保存候选频繁k项集，用于筛选下一级的频繁多项集；Lk保存频繁k项集。实验中采用frozenset集合的形式存储C1、L1、C2、L2、C3、L3，其特点是一旦集合中的元素被确定后，就不能再对其进行增加或删除操作，从而防止在操作过程中对集合元素造成无意的修改。

**（3）L**

①类型：列表

②作用：存储全部的频繁k项集，即L1、L2、L3。

**（4）item\_count**

①类型：列表

②作用：存储各个item出现的次数，为后续计算支持度做好记录。

**（5）support\_data**

①类型：字典

②作用：保存每个k阶项的支持度，其中键为k阶项，值为支持度。

**（6）rule\_list**

①类型：列表

②作用：存储最终的关联规则，每一项包含关联规则左侧的item、右侧的item以及置信度。

**（7）candidate**

①类型：列表

②作用：PCY算法中经过两次哈希后得到的候选频繁项集。

**（8）hash\_table1、hash\_table2**

①类型：列表

②作用：PCY算法中存储两个哈希表的vector矢量，便于后续计算每个桶的支持度。

### 3.2.4 遇到的问题及解决方式

本次实验中主要遇到了两个问题。第一是在基础的Apriori算法中，在计算support值时没有除以购物车的总数，导致计算出来的L1、L2、L3三个频繁项集的元素总数都和标准答案不一致，这是比较粗心的一个问题。

第二是在PCY算法中，我使用了两个哈希函数，分别是mmh3.hash()以及hash()，然而这两个函数得到的哈希表的vector矢量均显得非常不均匀，大量的hash值出现了0次，而只有很少量的hash值出现了很多次。后来我发现，我是将每个项集中的集合转变成字符串并获取其hash值，但是整个Groceries文件中的货物种类总共只有169种，在单词上有高度的重叠，因此最后得到的hash值也是分布不均的，不过这不会影响最终的结果。

### 3.2.5 实验测试与结果分析

**（一）Apriori算法实验结果分析**

运行apriori.py，分别输出了C1、L1、C2、L2、C3、L3的总项数，以及L1、L2、L3三个项集所含的元素以及相对应的支持度support。最后输出了关联规则的总数目、具体条目以及每条规则的置信度confidence。如图3-5~图3-8所示。

可以看出，L1有120项，L2有605项，L3有264项，与课上给出的参考答案一致，说明算法运行正确。



图3-5：C1、L1具体信息



图3-6：C2、L2具体信息



图3-7：C3、L3具体信息

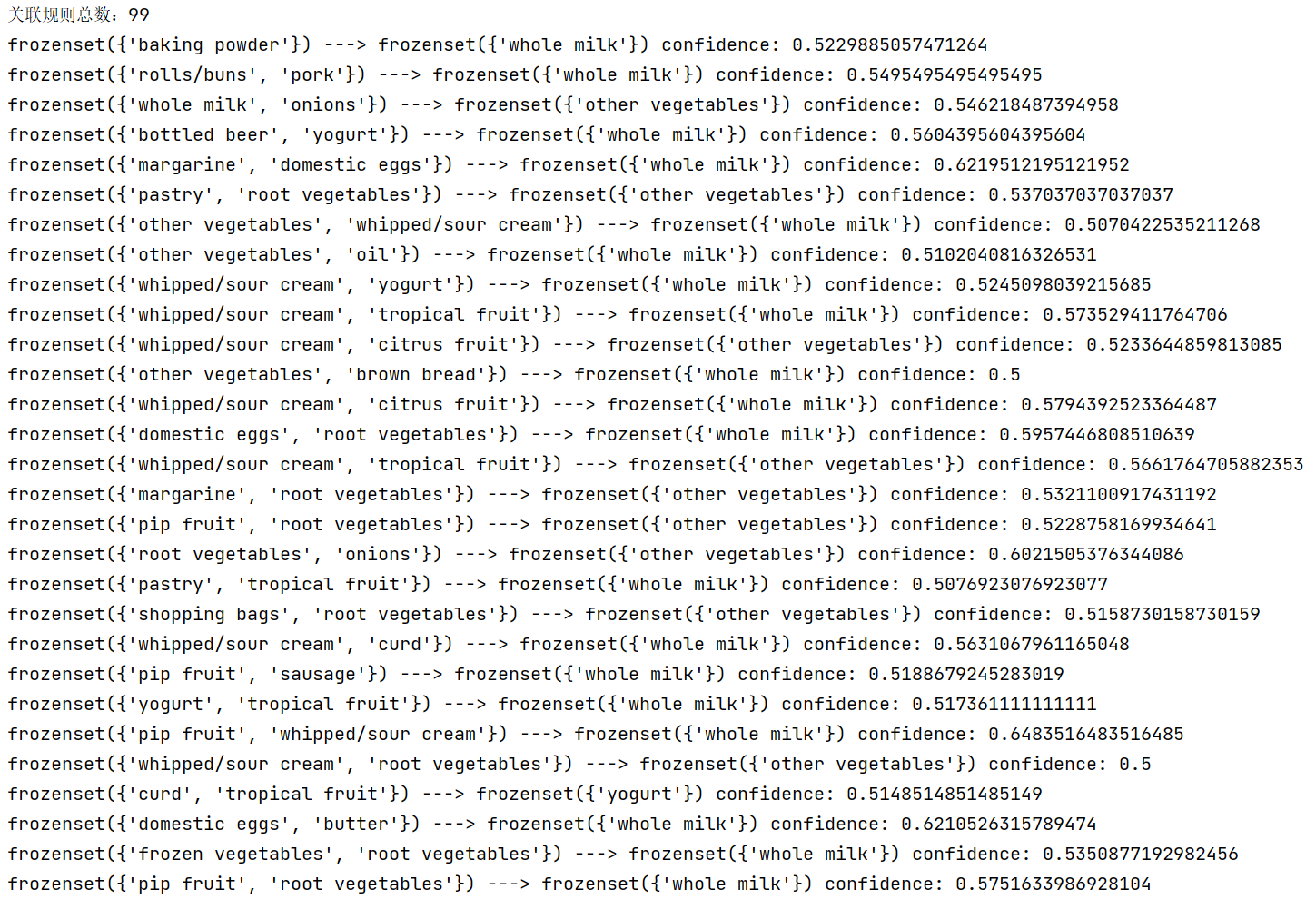


图3-8：关联规则信息

**（二）PCY算法实验结果分析**

运行pcy.py，输出了两个哈希表各自的vector矢量（以bit位的形式表示），以及L1、L2、L3三个项集所含的元素以及相对应的支持度support。最后输出了关联规则的总数目、具体条目以及每条规则的置信度confidence。如图3-9~图3-13所示。

可以看出，L1有120项，L2有605项，L3有264项，说明算法运行正确。

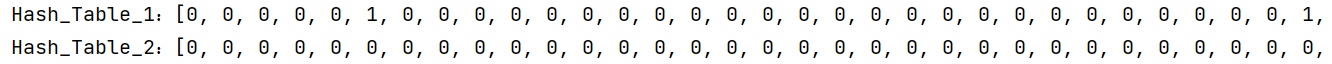


图3-9：Hash\_Table\_1和Hash\_Table\_2的bit vector信息（节选）

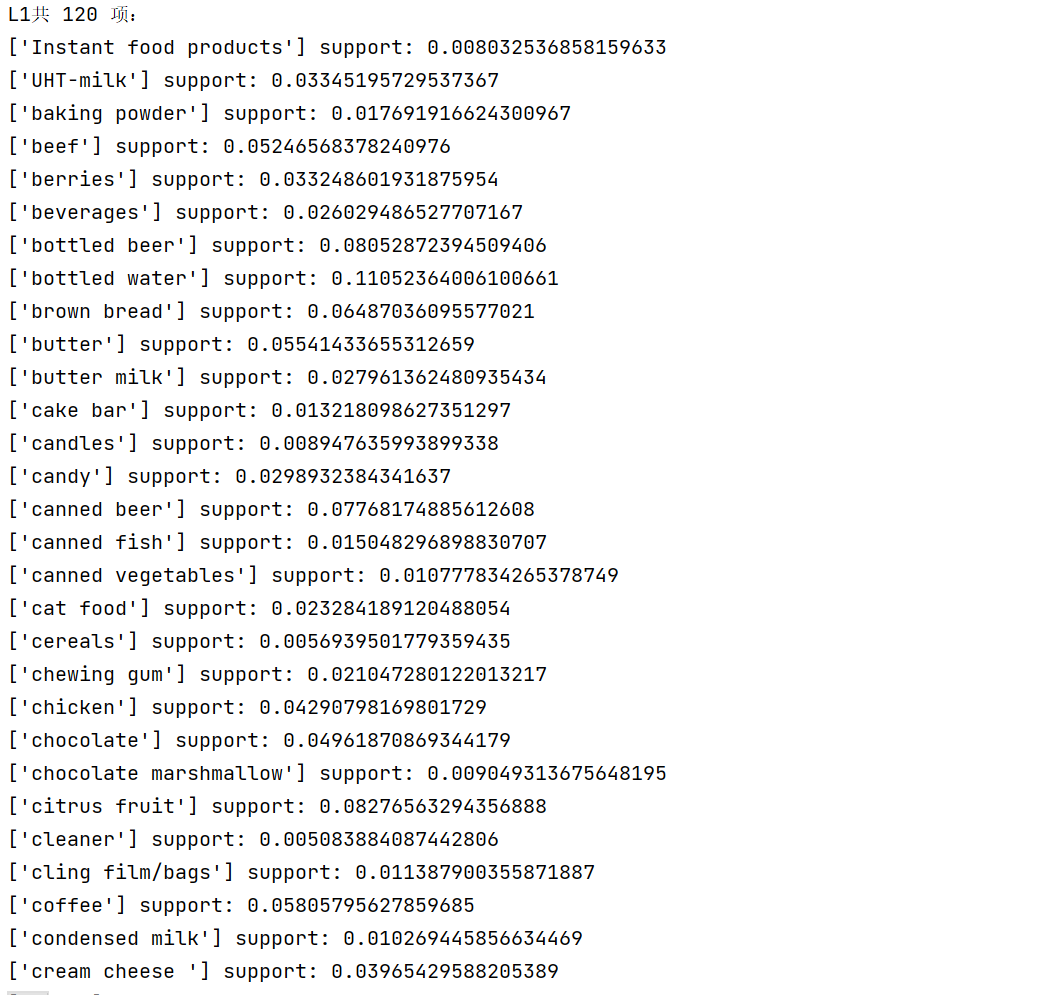


图3-10：L1具体信息



图3-11：L2具体信息



图3-12：L3具体信息



图3-13：关联规则信息

## **3.3 实验总结**

通过对Apriori算法的实现，我进一步掌握了迭代生成频繁n项集的方法，理解了项集的支持度、关联规则的置信度的计算方法，并得到了频繁的一至三项集以及全部的关联规则。这个算法主要是利用逐层搜索的迭代过程，找出各阶的频繁项集，其过程由连接与剪枝组成。实际上，这个算法在现实生活中有着十分重要的作用，例如分析消费者购买行为模式，并使商家制定更合理的销售策略。

除了完成基础的Apriori算法之外，我还使用基于Multihash的PCY算法进行了一定的改进。通过使用两个哈希函数、建立两个哈希表，我将全部的候选二项集分配到了不同的哈希桶中，并根据候选二项集成为频繁二项集的必要条件将其有选择性地放入candidate集合中，优化了二阶频繁项集的计算效率。无论是哪种算法，最后得到的结果均与参考结果一致。

值得一提的是，无论是Apriori还是PCY算法中都涉及到了剪枝的重要思想，在数据量很大的时候，全部遍历并获得每一种可能的情况是十分繁杂且没有必要的。在计算过程中，只需要时刻检查当前情形是否合法，若不合法则直接停止后续构造，即可大幅度减少可能性的总数，提高计算效率。

## **3.4 参考文献**

[1] 关联规则挖掘 <https://blog.csdn.net/yinlili2010/article/details/40193457>

[2] Apriori算法详解 <https://zhuanlan.zhihu.com/p/341882260?utm_id=0>

[3] 频繁项集与关联规则Frequent Itemset Mining and Association Rules、A-Priori算法、PCY算法 <https://blog.csdn.net/qq_39852142/article/details/111586236>