# 



**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**

[实验三 关系挖掘实验 1](#_Toc58793746)

[**3.1 实验内容** 1](#_Toc58793747)

[**3.2 实验过程** 1](#_Toc58793748)

[3.2.1 编程思路 1](#_Toc58793749)

[3.2.2 遇到的问题及解决方式 1](#_Toc58793750)

[3.2.3 实验测试与结果分析 1](#_Toc58793751)

[**3.3 实验总结** 2](#_Toc58793752)

# 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1~4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

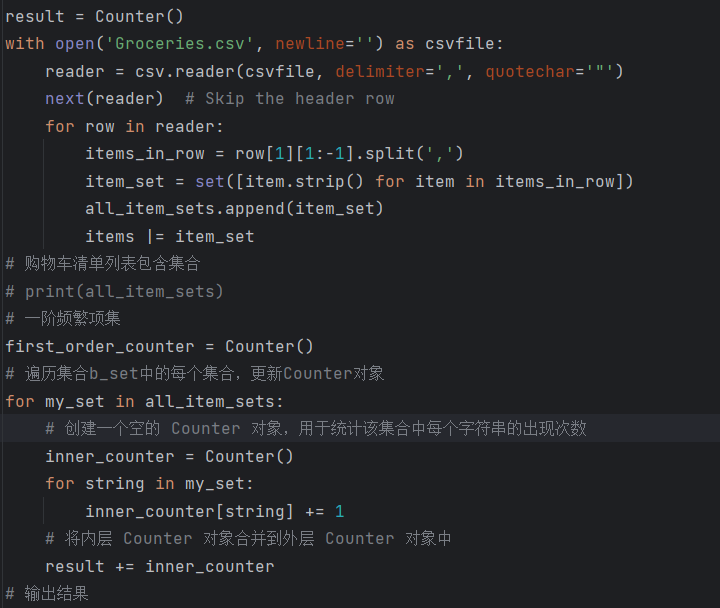
## **3.2 实验过程**

### 3.2.1 编程思路

本实验需要使用Apriori算法，根据Groceries.csv文件中的物品列表，找出数据集中集中频繁出现的数据集合以及可能会有关联的物品组。

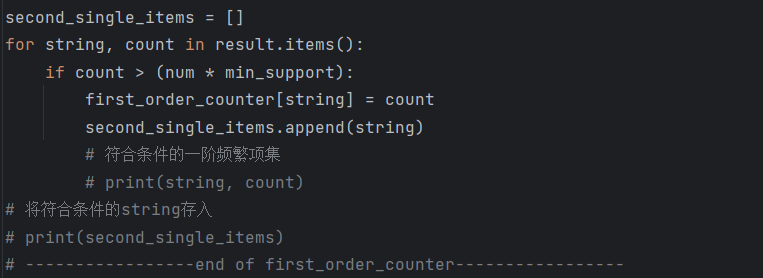
本实验设计到三个重要的概念：其一是关联规则，关联规则是形如X指向Y的表达式，并且X、Y互不相交，其可以通过支持度和置信度来衡量。其二是支持度，对于某个由物品构成的集合来说，其支持度等于该集合在所有购物清单集合中出现的次数，例如对于集合{A,B}，其在清单{A,B,C}中出现一次，而不在{A,C,D}中出现。其三是置信度，对于关联规则集合A指向集合B来说，置信度计算公式为，后两者是衡量频繁项集和关联规则的重要指标。值得注意的是，满足最小支持度是计算频繁项集和关联规则的前置条件！

本次实验，我们首先要实现csv格式文件的第二行中的购物清单进行读取，由于我们要得每个序号购物的集合以及出现的所有物品的集合，我们需要准备两个数据类型来保存这两组数据。观察到我们需要保存所有物品和所有物品清单，我们将每个清单视为一个集合，将所有清单集合保存在all\_item\_sets的集合当中，同时我们也将所有出现的物品保存在items的集合。同时观察到我们需要实时统计每个物品出现的次数，可以采用collections库中的Counter类型，其为字典dict的字类。值得注意的是对于dict类型的数据而言，其key为待统计的次数的数据，value为对应元素出现的次数。详细代码如下图所示：



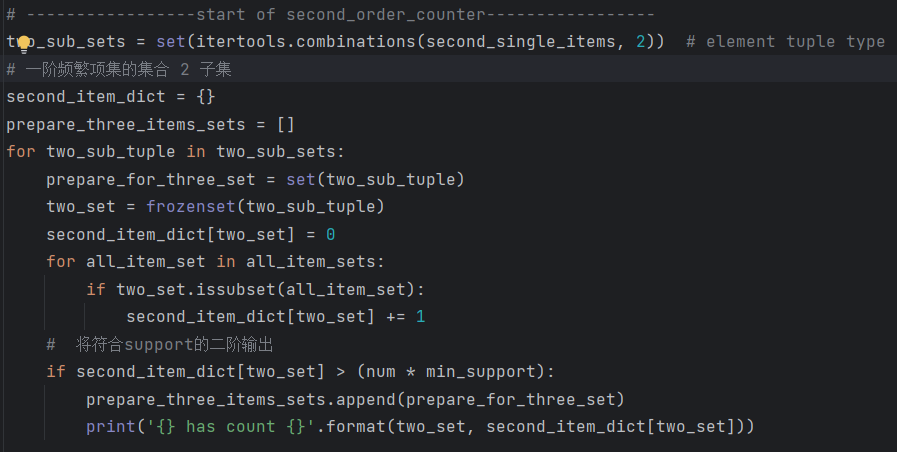
图一 读取文件部分

在读取好Groceries.csv文件后，我们要开始计算一阶频繁项集，并且为计算二姐频繁项集的备选集做准备。对与一阶频繁项集的筛选，我们初步只需要考虑最小支持度的要求即可。由于我们直接统计了次数，所以在筛选的过程中，我们只需要把Counter类型result中的value值和(物品数量\*min\_support)相比较即可。同时，作为符合最小支持度的物品存入sencond\_single\_items的集合作为二阶频繁项集的备选集。这一步的代码如下图所示：



图二 遍历result将符合最小支持度的存入备选集

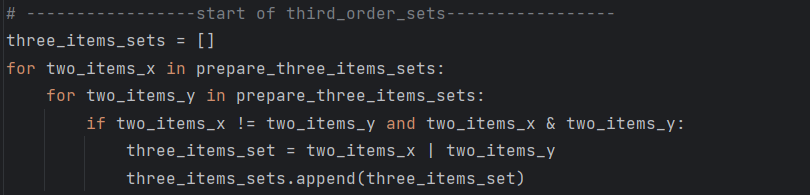
第三步，我们需要计算二阶频繁项集并且选择符合条件的作为三阶频繁项集的备选集。首先我们要从符合最小支持度条件second\_single\_items中选择两个元素作为二阶项集。这里我使用了itertools库中的combinations函数，从second\_single\_items集合中构造元素数为2的集合，并且存入second\_item\_dict中。并且在后续筛选符合条件的二阶频繁项集过程中，我们同时将符合条件的二阶频繁项集存入prepare\_three\_items\_sets中。相关详细代码如下图所示：



**找到元素数为2的子集**

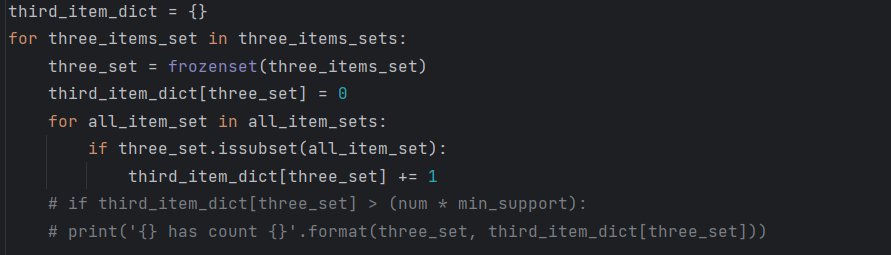
图三 得到二阶备选集，筛选二阶频繁项集

第四步，我们需要从三阶频繁项集的备选集prepare\_three\_items\_sets之中，获得元素数为3的三阶项集，然后去进行与上面相同的筛选。这里涉及到集合的运算，对于本次实验二阶频繁项集来说，若存在{A,B}和{A,C}那么可以得出一个备选集{A,B,C}。因此我们需要将相交的集合但不相等的集合进行合并，所以根据条件我们可以写出集合运算如下，假设这里的集合为set1、set2，则运算式为(set 1 != set2) and (set1 & set2)，然后再”new\_set = set1 | set2”即可得到三阶项集了。详细代码如下图所示：



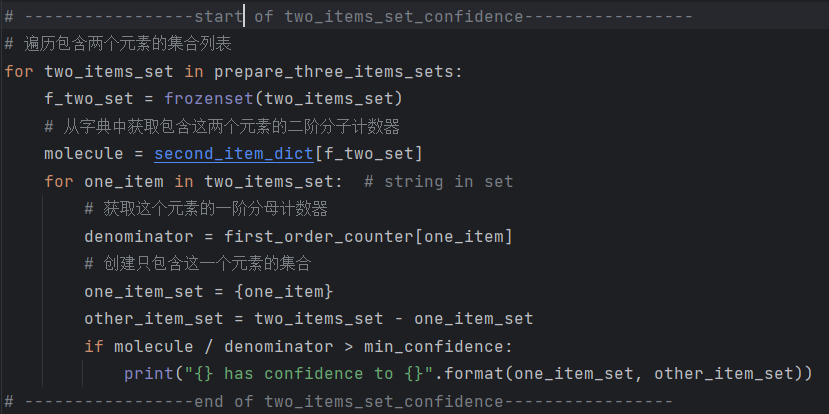
图四 对三阶频繁项集备选集进行集合运算得到三阶项集

接着为了得出结果我们只需要和上面相同的筛选过程即可。相关详细的代码如下图所示：



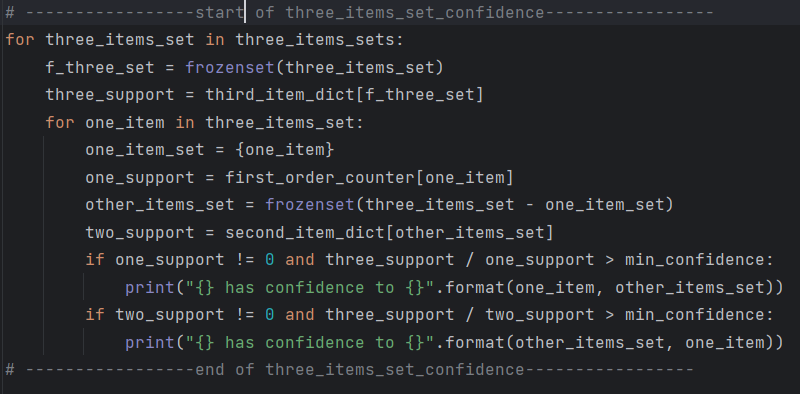
图五 筛选三阶项集，保存次数

到这里我们已经完成了频繁项集的计算，接下来只需要根据上面计算的结果去计算关联规则的置信度即可。由于置信度公式：（插图），所以我们得到其概率之比即为对应集合出现次数之比，而次数我们已经统计到各个字典型数据当中了，所以这一步操作较为简单。对于二阶频繁项集来说，比如项集set1中为{A,B}我们需要计算A对B的置信度，以及B对A的置信度。当我们知道二阶项集后，其实我们只需要判断A、B出现的次数是否小于(min\_convince \* set1\_cnt)即可，相关关键代码如下图所示：



图六 由二阶频繁项集计算一对一的关联规则

计算完一对一的关联规则，接下来就是计算一对二以及二对一的关联规则。这一部分相对于二阶频繁项集来说较为麻烦，我们需要先遍历所有的三阶频繁项集，然后在对三阶频繁项集进行遍历，将其拆分成两个集合，这里我将其命名为one\_item\_set以及other\_items\_set，我们将遍历过程的单个物品字符串转化为集合，再通过集合运算得到other\_items\_set。在进行筛选的过程中，我们需要排除support为0的一些特殊情况。相关关键代码如下图所示：



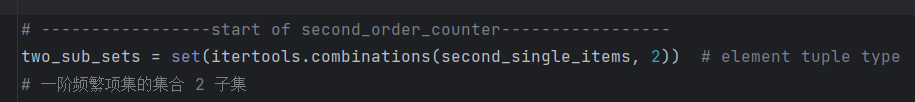
图七 由三阶频繁项集计算一对二、二对一的关联规则

### 3.2.2 遇到的问题及解决方式

**问题一，如何根据一阶频繁项集得到我们二阶项集的备选集？**

我一开始想通过直接遍历的方式，将一阶的这些字符串直接互相组合构成二阶项集的备选集，这样的可能会涉及两次for循环，导致该步骤的时间复杂度陡然增加，延缓代码运行。在网上查询集合的操作函数过后，可以选择itertools库中的combinations函数来进行该项操作，这个函数重在“组合”返回一个tuple元组类型的数据，每个元组中有2个元素。最后转化为集合类型方便后续处理。

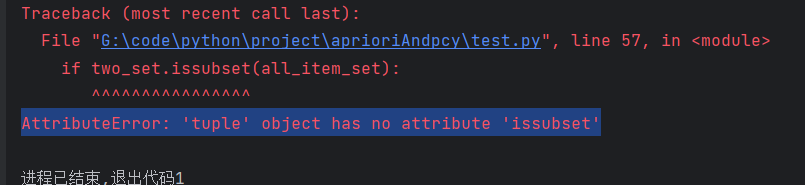
相关程序处理如图所示：



图八 由一节频繁项集获得二阶项集备选集

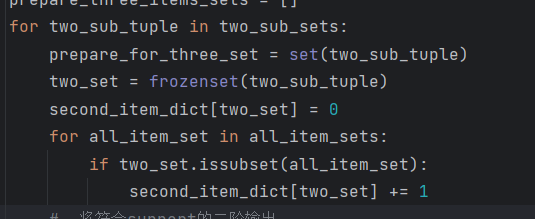
**问题二，如何将集合作为列表查询的索引来存储出现次数？**

我一开始没有细想写代码的时候直接将上一阶项集集合作为列表的索引，后来发现报错如下所示：



图九 索引出现错误

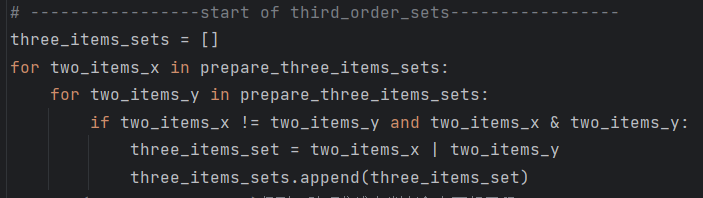
这个问题出现的原因是，作为一个集合来说，它可以自由的添加、删改元素。但是对于列表的索引来说，却不能是动态的，必须要求是固定的。所以这里我们需要将这个集合“固定”，这一步我们可以通过forzenset()的函数来实现。这个函数会返回一个冻结的集合，冻结后的集合不能再添加或者删除任何元素。另设一个作为索引的冻结集合two\_set，在遍历中将其赋值two\_set = forzenset(two\_sub\_tuple)，程序正常运行，相关代码如下所示：



图十 将集合进行frozenset()冻结操作

**问题三，如何根据二阶频繁项集得到我们三阶项集的备选集？**

这个问题我没有很好的处理办法，想着就是通过遍历一个个去生成三阶项集。通过网上查询，我得到了一个比较好的生成办法：在二阶频繁项集，将有两个元素重复的项集可以进行合并生成两个三阶项集，对于只有一个相同元素的，就无需考虑了。这一步设计到几个集合运算规则：首先我们假设两个集合set1和set2，(set1 & set2)是判断两个集合是否有交集，如果有就返回1，没有就返回0；(set1 | set 2)是将两个集合进行合并。所以对于本操作，只需要将不相等且有交集的进行合并的操作即可，相关关键代码如下图所示：

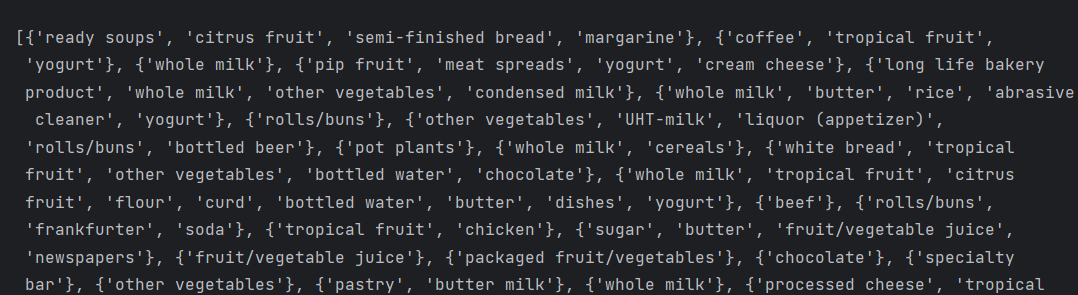


图十一 由二阶频繁项集得到三阶项集的备选集

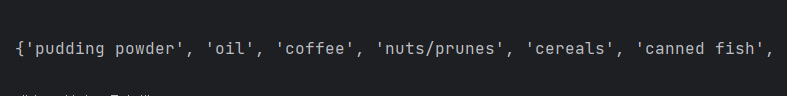
### 3.2.3 实验测试与结果分析

在实验测试步骤中，先将所有的输出语句注释掉，在代码最后逐个输出涉及到的所有数据：购物车清单列表all\_item\_sets、物品列表items、一阶频繁项集first\_order\_counter、二阶频繁项集second\_item\_dict、三阶频繁项集third\_item\_dict、

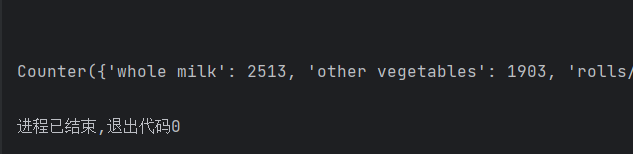
三阶备选集three\_items\_sets。相关的输出如下图所示：



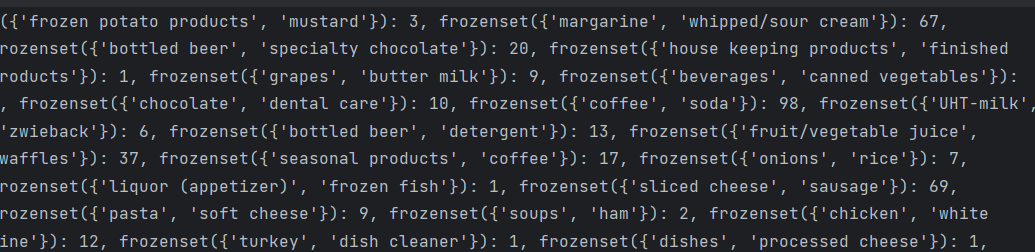
图十二 部分all\_item\_sets内容输出



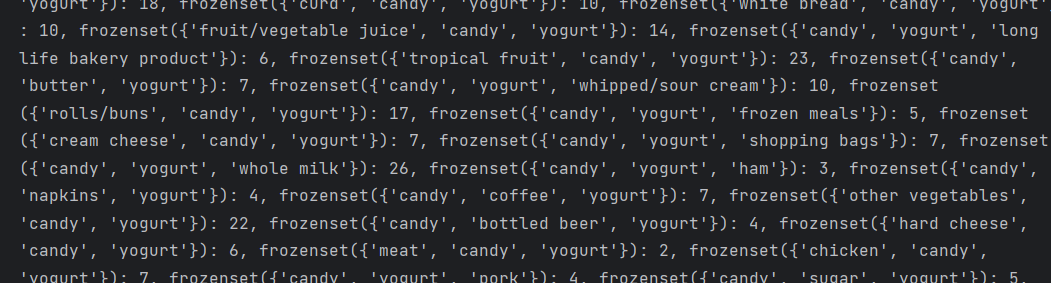
图十三 部分items内容输出



图十四 部分first\_order\_counter内容输出



图十五 部分second\_item\_dict内容输出



图十六 部分third\_item\_dict内容输出

对结果分析发现：一是，在所有关联规则中，‘whole milk’ ‘vegetables’ ‘yogurt’三词出现频率较高，这反映出三者是购物中经常会和其他商品一起购买的物品，这个结论**可以帮助商家更好的了解市场需求，从而优化产品组合**；二是，这些出**现次数多的物品，更容易成为关联规则的前提**，可以帮助我们挖掘出更多的关联规则；三是，由于这些物品在很多交易中都出现过，这个物品也**可能不具有区分度**，和其他物品出现频率相近，那么**可能挖掘出来的关联规则没有意义**。

整体代码思想、步骤正确，能够得到最终的正确结果。

对本次实验进行结果分析，与其他同学商讨算法的编写步骤，以及在助教指出相关问题后，我对我的代码有了如下分析：

第一项，**数据命名不规范还可以重新改进**，以一个固定的规范来进行命名。比如本实验涉及到的几个频繁项集，我没有统一命名，会显得有些杂乱，在结果输出时也会让代码显得不整洁。在中间的算法过程中，也会出现一些数据命名混乱的问题，最后的课设过程中需要加以改进。同时部分数据的类型也有混乱的情况，还需要统一进行类型转换，或者使用type()的函数来判断其类型，来帮助后续代码的编写。

第二项，可以采用在MapReduce中出现的**多线程函数来加快求频繁项集的备选集**的过程。由于我们在处理备选集的时候，是对一个有很多数据的文件同时处理的，所以可以考虑使用之前涉及到的threading库进行多线程处理。在本实验的代码中，处理数据速度较慢，因为里面有较多次的for循环以及for循环的嵌套，导致数据处理慢。

第三项，**有一些计算步骤是重复的**，比如根据备选集去挑选符合最小支持度的项集这一步，可以加以改进。比如建立一个函数select\_support\_set(set1,set2,set3)这里的set1是备选集，set2是购物车清单，set3是将符合要求的元素存入的集合。这样可以减少代码的重复过程，减少代码长度，节省空间。

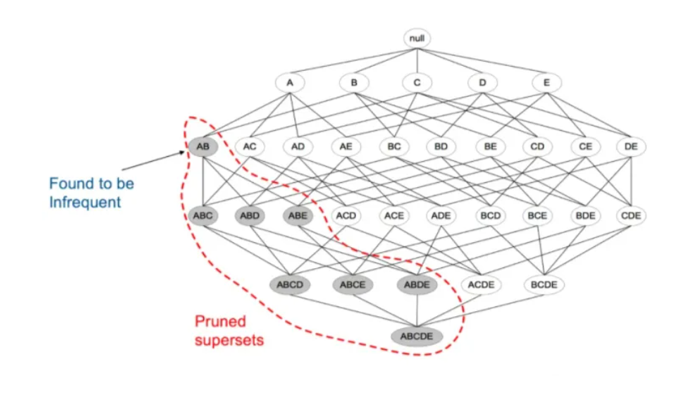
## **3.3 实验总结**

首先是针对3.2.3中提到的“这个物品也**可能不具有区分度**，**可能挖掘出来的关联规则没有意义**”的问题，我们可以在A-priori算法中最小置信度和最小支持度的基础上添加提升度（lift）来衡量一个关联规则的是否有意义。假设这个关联规则涉及购买物品集合A和购买物品集合B，我们可以知道A对B的置信度和B对A的置信度分别为：

那么可以求出提升度L（lift）：

如果提升度=1说明A和B没有任何关联，如果<1，说明A事务和B事务是排斥的，>1，我们认为A和B是有关联的，但是在具体的应用之中，一般认为提升度>3才算作值得认可的关联，但是在实际应用过程中受购物总单数的影响比较大，需要多方面来考虑关联规则是否具有现实意义。

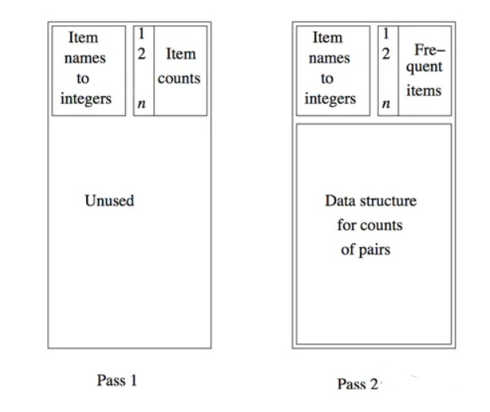
接下来对A-priori算法进行分析，本次A-priori算法有以下的优点：一是算法实现较为简单，只需要按照最小置信度、最小支持度的要求遍历求解，将符合要求的存入作为备选集进行下一阶挑选即可；二是数据结构较为简单，实验中我统一采用了集合结构，这种组织方式不仅易于处理，还可以有效的减少内存的消耗；三是处理过程中我们采用类似于剪枝的操作，即根据原理如果一个物品集合不频繁，那么它的子集也不会频繁，可以根据这个原理来减少备选集的数量，提升算法的效率；四是A-priori算法可以快速处理稀疏的数据集以进行数据挖掘，由于三中采用了剪枝的操作，所以我们加快处理速度，如下图所示：



图十七 剪枝操作

本次A-priori算法有以下的缺点：一是对数据扫描次数过多，由于算法中多次出现for循环来遍历查询数据导致算法的时间复杂度比较大，会占用大量的资源；二是算法可能会产生很多候选集，这些候选集会导致算法的内存占用量较大，降低运行效率；三是算法适应面窄，只采用支持度和置信度来衡量关联规则可能会存在实际意义不大的结果，这一点上面也提到了。

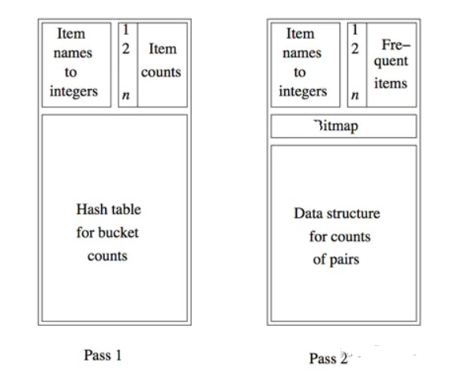
针对A-priori算法内存使用率不够这一缺点我们可以改进使用PCY算法。PCY算法相对于A-priori算法而言有了以下几个改进：一是通过使用哈希减少扫描数据的次数，在PCY算法中我们只需要进行两次扫描分别实现生成哈希桶和计算候选集的支持度；二是增加了内存的使用效率，在PCY算法中只需要在第一次扫描中存储哈希桶的计数值我们就可以减少存储空间的使用；三是在计算备选集时使用了哈希桶来计算，减少备选集的生成，方便处理更大的数据集。如下图所示，为A-priori算法两次扫描图



图十八 A-priori算法两次扫描图

根据这个图可以知道第一遍有大部分空间没有使用，在第二遍中，进行了最小支持度筛选得到后续阶的备选。

再来看PCY算法扫描两边后的内存占用图：



图十九 PCY算法两次扫描图

PCY算法在第一遍扫描时对购物篮进行检查,且不仅对篮中的每个项的计数值加1，而且通过一个双重循环生成所有的项对。对每个项对我们将哈希结果对应的桶元素加1，而且项对本身并不会放到哈希桶中，因此它只会影响桶中的单个整数。也就是第一遍扫描时图中下面的部分。第一遍扫描结束时，每个桶中都有一个计数值，记录的是所有哈希到该桶中的项对的数目之和，如果该桶的计数值高于支持度阈值，那么该桶称为频繁桶。

所以很直接就能看出PCY算法相对于A-priori算法减少了算法时间加快效率，减少了空间复杂度，

在现实生活中关系挖掘算法对人们的生活影响很大，它可以通过挖掘数据之间的关系来帮助人们更好的分析数据，进而发现数据中的规律，提高决策的准确率。比如最经典的购物，帮助商场更好的摆放商品顺序。在学习和实践过程中，我了解到体会到关系挖掘的挑战性。一方面，关系挖掘需要我们具备一定的数据分析和建模能力；另一方面，关系挖掘也需要我们具备一定的业务知识和领域理解能力。在现在的信息社会中，随着数据量的不断增加和数据应用场景的不断拓展，关系挖掘技术还存在一些待解决的问题和挑战。例如，如何改进算法来让其更好的处理大量的数据，又或者如何添加衡量标准来确定关联规则。