

**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 实验目的**

1、理解Apriori算法思想与流程，了解其生成频繁项集的具体步骤。

2、应用Apriori算法解决关联规则提取问题；

3、掌握PCY算法，理解PCY算法相对于Apriori的改进思路和流程变化，验证PCY和Apriori算法的性能。

## **3.2 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

## **3.3 实验过程**

### 3.3.1 系统整体框架

Apriori算法是一种常用的关联规则挖掘算法，用于发现数据集中的频繁项集及其关联规则。通过逐层筛选频繁项集，我们可以挖掘出最终的规则。Apriori算法的框架如图3-1所示（仅展示两个pass）[2]。

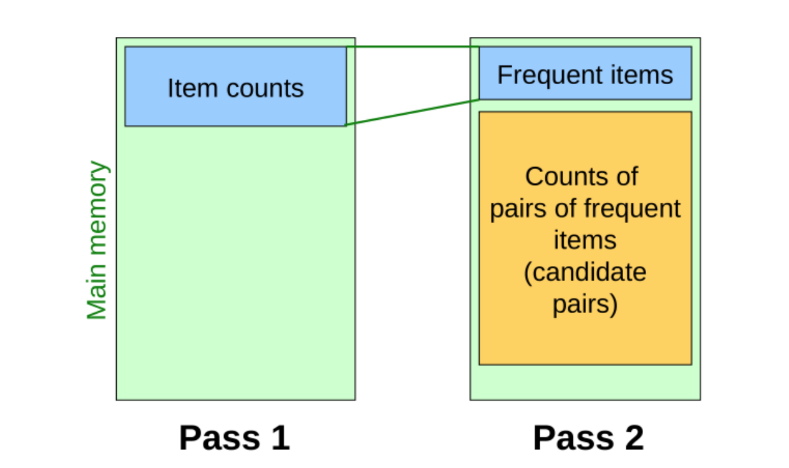


图3-1 Apriori算法框架示意图

在Apriori的基础上，PCY算法引入了哈希函数来降低内存消耗和提高性能。通过使用哈希函数将项集映射到内存中的一个存储桶（bucket）中，并使用bitmap标记频繁项桶的信息，能够减少生成频繁项集的时间消耗，快速得到频繁项候选集。PCY算法的框架如图3-2所示。

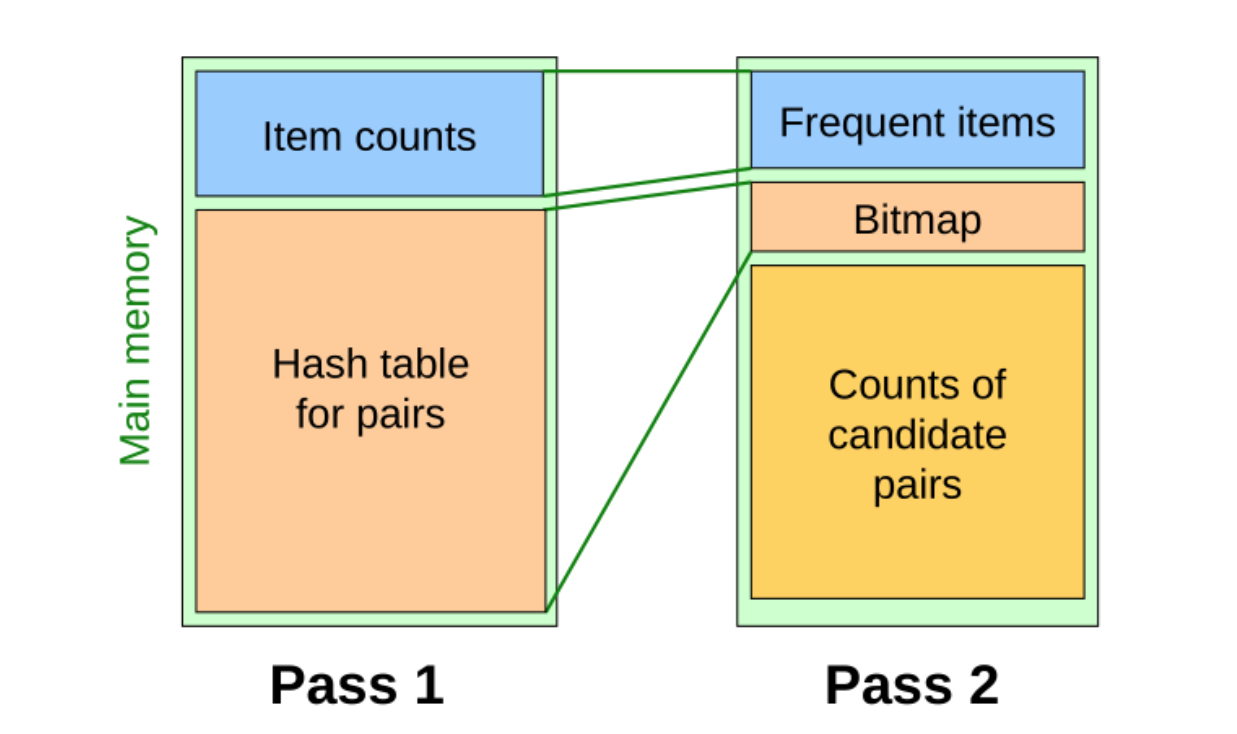


图3-2 PCY算法框架示意图

在PCY的实现上，可以采用multi-stage（多阶段），multi-hash（多哈希）等变式，进一步优化算法性能。multi-stage在每个pass中均采用了哈希映射的方式，multi-hash在单个pass中将pairs分割成多个hash table，分别进行映射。两种变式的框架如图3-3，3-4所示，由于其仅为PCY的变式，本次实验不做实现。

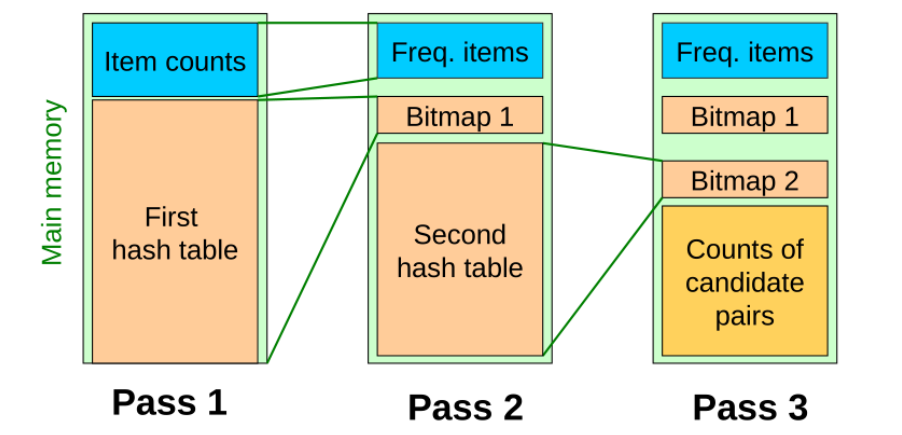


图3-3 multi-stage框架示意图

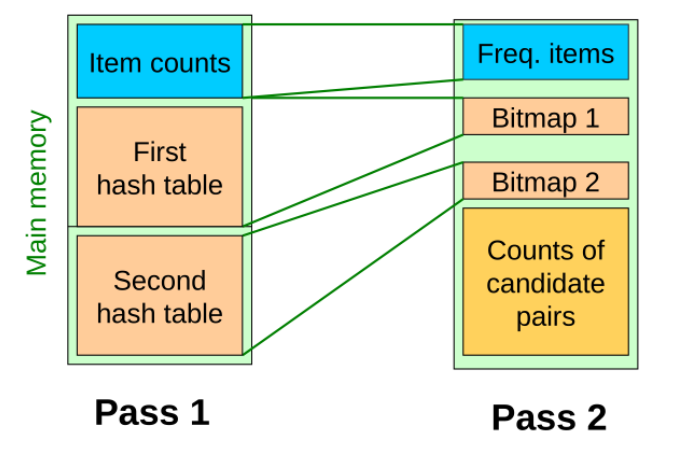


图3-4 multi-hash框架示意图

### 3.3.2 编程思路与实现

**1）数据处理**

原始数据集Groceries.csv中的数据格式如图3-5所示，每一行代表一个“购物车”（basket），第二列代表购物车中的物品集（items）。

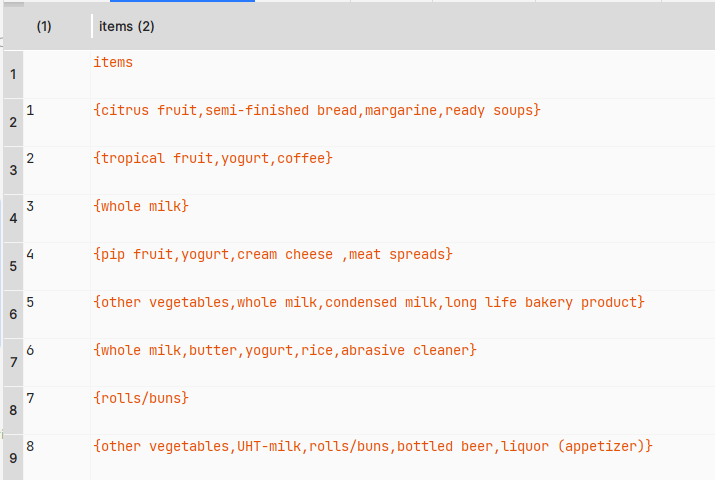


图3-5 原始数据集截图

然而，这样的数据无法直接读取，我们首先采用list[list]的格式将数据存储进dataset中方便进一步的操作。为了方便展示数据格式，我将其写入了txt问题件（代码如图3-6所示，结果如图3-7所示）。由于本实验中涉及到多个不同的数据结构，接下来会对其进行总结。



图3-6 数据处理代码截图

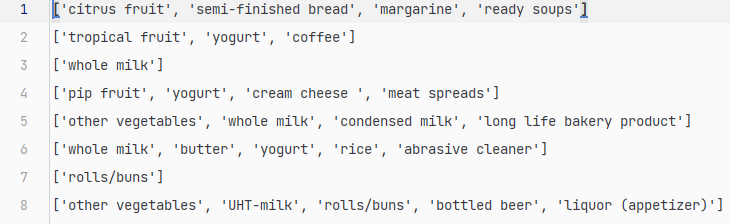


图3-7 dataset数据展示

**2）数据结构选择**

A）dataset

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

2．功能：转换原有数据集格式，简化后续读取操作。

B）频繁项候选集Ck

1．类型：set(frozenset)类型。

2．功能：存储第k次频繁项候选集。其中k代表每个frozenset中有k个item。事实上，frozenset（不可变集合）的选用是在查阅网上资料后得出的结论，大部分实现均采用了此种手段，其具体原因我将在3.4.1节进行阐述。

C）频繁项集Lk

1．类型：set集合类型，集合内的元素格式为frozenset。

2．功能：存储第k次频繁项集。其中k代表每个frozenset中有k个item。

D）频繁项集及其置信度frequent\_setk（conf）

1．类型：dict{frozenset,float}字典类型。

2．功能：存储第k次频繁项集及其置信度。

E）关联规则rule\_list

1．类型：list[set,set,float]列表类型。

2．功能：存储由频繁项集生成的关联规则。

F）hash\_table

1．类型：list列表类型。

2．功能：存储每个哈希桶中集合的数量。

G）bucket

1．类型：list[list]列表类型。

2．功能：存储每个哈希桶中集合的具体items。

F）bitmap

1．类型：list列表类型。

2．功能：判断每个哈希桶是否是频繁桶，如果其值为1，则为频繁桶。

**3）Apriori算法实现**

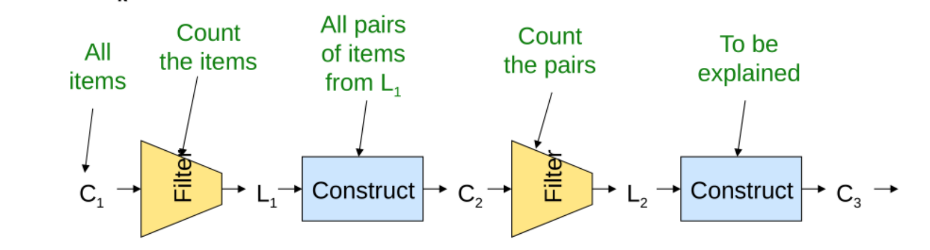


图3-8 Apriori算法流程图

Apriori算法实现的部分流程如图3-8所示，具体地可分为四个原子操作：生成初始频繁项候选集C1，生成下一个频繁项集Lk，生成下一个频繁项候选集Ck，生成关联规则rule\_list。下面我将具体阐述些操作。

A）生成初始频繁项候选集C1

此部分仅需遍历dataset中所有的basket，将每个basket的所有item以frozenset的类型add进C1的集合中。代码如图3-9所示



图3-9 生成初始频繁候选集C1代码展示

B）生成下一个频繁项集Lk

由频繁项候选集生成频繁项集的过程是筛选候选集合的过程。首先遍历数据集中的每一个basket，再遍历候选集中的每一个k阶集合是否为其子集，若是则其频次增加。之后计算每个k阶集合的支持度（support），计算方式为：支持度 = 出现频次/总项集数，即出现在数据集中的交易中的频率。如果支持度大于最小支持度（通常为0.005），则视其为频繁项，加入频繁项集Lk中。此部分的代码如图3-10所示。



图3-10 生成下一个频繁项集Lk代码展示

C）生成下一个频繁项候选集Ck

在生成频繁项候选集的过程中，Apriori算法采用了剪枝策略（Pruning Strategy）和连接策略（Join Strategy）。剪枝策略基于一个先验性质，即如果一个项集是频繁的，那么它的所有子集也一定是频繁的。连接策略的思想是若两个长度为k-1的频繁项集，如果他们前k-2项均相同，则可以将他们连接形成一个长度为k的频繁项候选集。

具体地，在创建频繁项候选集时，我们应用两种策略。首先将两个频繁项集（每个集合为k-1项）取并集，若其长度为k则满足连接策略。判断子集是否是频繁项集，若不是则执行剪枝操作。通过应用这两种策略，能够减少搜索的集合范围，提升运行效率。



图3-11 生成下一个频繁项候选集Ck代码展示

D）生成关联规则rule\_list

由频繁项集生成关联规则的过程中，我首先对每个集合进行了置信度（conf）判断，将超过最小置信度的规则判定为有效规则并输出。在生成规则后，写入结果文件中。



图3-12 生成关联规则代码展示

**4）PCY算法实现**

在PCY算法实现中，可分为三个步骤：生成哈希表统计每个桶的数量，生成存储集合的桶，判断是否为频繁桶（bitmap），生成下一个频繁项候选集。下面我将详细阐述这些过程：

1. 生成哈希表（hash\_table）和桶（bucket）

由原始数据生成第一次频繁项候选集（C1）时，我们遍历每一个“购物车（basket）”中的所有物品对，将其哈希到某个桶里，相应地修改哈希表的值和桶的集合。

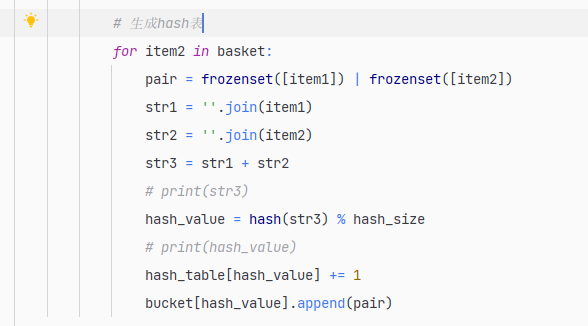


图3-13 生成哈希表和哈希桶代码展示

1. 判断是否是频繁桶

对于每一个桶，通过计算其支持度（即出现的频率），如果其超过最小支持度，就认为其为频繁桶，bitmap值置为1，反之则置0。



图3-14 判断频繁桶代码展示

1. 生成频繁项候选集

当我们完成了bitmap后，生成频繁项候选集会变得十分高效，只需将所有频繁桶中的集合加入候选集即可，其代码如图3-15所示。

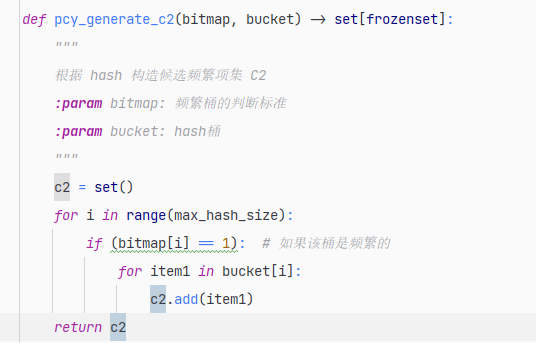


图3-15 生成频繁项候选集代码展示

### 3.3.3 实验测试

**1）实验环境**

本实验使用python进行开发，版本为3.11.2。使用的库为numpy，os，time，pandas等。

实验的目录设置和基本参数如图3-16所示：频繁项集最小支持度为0.005，关联规则最小置信度为0.5。



图3-16 实验基本参数配置

**2）Apriori算法测试**

首先测试该算法的正确性，生成各阶段频繁项集L1、L2、L3和关联规则。结果如图3-17所示。关联规则的部分截图如图3-18所示。



图3-17 Apriori算法运行结果

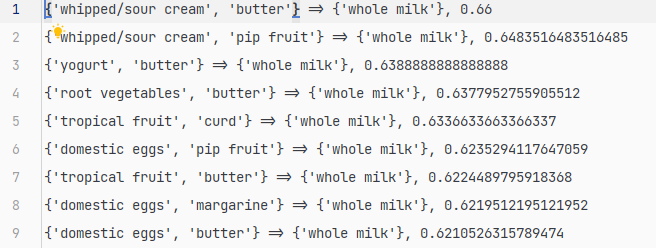


图3-18关联规则截图（降序排序）

3）PCY算法测试

经过测试，PCY算法得到的结果于Apriori算法相同，其部分bitmap值如图3-19所示。

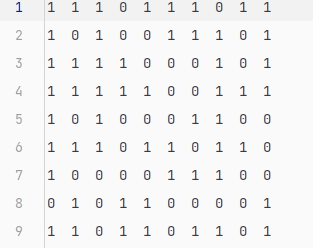


图3-19 部分bitmap值截图

### 3.3.4 对比实验与结果分析

本实验的对比主要分为两个方向，一个是Apriori算法和PCY算法在运行中的性能比较，第二个是PCY算法中不同参数的运行结果。

1. **哈希算法带来的性能改进**

在使用PCY算法之后，整个算法运行的时间有了明显的增加。为了更进一步的分析原因，我输出了每一个阶段所使用的时间。具体的结果如图3-20所示。从实验结果来看，PCY带来最大的优化就是在生成C2时减少了集合的数量，降低了搜索的范围，所以生成L2频繁集时可以大幅减少耗时。事实上，由C2生成L2这个过程是耗时最长的过程。不过，由于在生成C1和L1时加入了哈希操作，其耗时会对应增加。



图3-20 Apriori算法和PCY算法对比

1. **选择合适的哈希桶数量**

从理论上分析，如果选择更多的哈希桶，那么每个哈希桶中集合的数量会更少，更加均匀，由此生成的C2候选集数量会进一步减少，性能会相应地增加。但是我们还应注意到，更多的哈希桶会导致在判断频繁桶，遍历频繁桶的操作时，会有更多的耗时。但是就本实验来讲，增加哈希桶的数量的确促成了程序性能的提升，具体结果如图3-21所示。



图3-21 PCY算法中选用不同哈希桶数量运行结果

## **3.4 实验总结**

### 3.4.1 实验中遇到的问题及解决

**1）使用frozenset**

由于高阶候选集中的元素都是集合，一开始我使用简单的set当作其元素，但是发生了报错，问题是set类型为可变量，但是set中的元素是不可变的。在网上查阅资料后，我了解到了frozenset类型（不可变集合），既能满足集合的特性，又能够存储在集合或者字典的键之中。

1. 生成哈希表

生成哈希表需要遍历每一个basket的每一对集合，在一开始我没有完全理解这种操作，没有去掉相同item组成的pair，导致在生成L2时频繁项的数量不正确，也导致了程序耗时增加。在检索到问题后，我重新书写了遍历方式，得到了正确的结果。解决方法如图3-22所示（其实很简单，但是很有效）

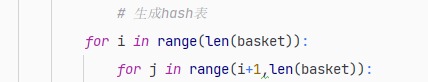


图3-22 解决问题的代码

### 3.4.2 实验总结

本次实验引入了关联规则，从生活中的具体现象抽象出一般的关系，从“购物车”行为中提取出商品相关的关系。这让我有两点感悟。一是，我们习以为常的平常背后都存在着充分地考虑。所以我们在生活中，不仅要对于异常的现象发问，更要对平常发问。而是，在计算机领域，学会抽象本质是我们进行科学研究非常需要的能力。以后我也会多锻炼这些能力，多实践，多思考。