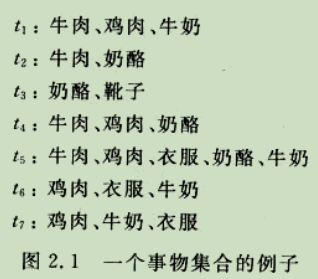
# Apriori算法python实现

## 数据结构：

算法整体结构：

|  |  |  |
| --- | --- | --- |
| 变量名 | 含义 | 数据结构 |
| T / data\_set | 数据集 | 二维列表 |
| item | 项 | String |
| fk | 频繁k项集frozenset([item]) | frozenset |
| Ck | 频繁候选k项集集合（fk的集合）。  合并：Fk-1两两合并得到C，  剪枝：C的任意(k-1)子集都在Fk-1中才加入Ck | 集合 |
| Fk | 频繁k项集集合（fk的集合）  Ck中达到最小支持度的k项集组成Fk | 集合 |
| support\_data | 所有频繁项集的支持度，字典结构，  键：频繁项集fk，值：fk对应的支持度 | 字典 |
| F | 频繁k项集集合Fk的列表  频繁k项集集合Fk在F中位置：F[k-1] | 列表 |
| Hm | 后件有m项的后件集合  Hm生成Hm+1也要合并，剪枝 | 集合 |

书中图2.1的存储：



data\_set = [['牛肉', '鸡肉', '牛奶'],

            ['牛肉', '奶酪'],

            ['靴子', '奶酪'],

            ['牛肉', '鸡肉', '奶酪'],

            ['牛肉', '鸡肉', '牛奶', '奶酪', '衣服'],

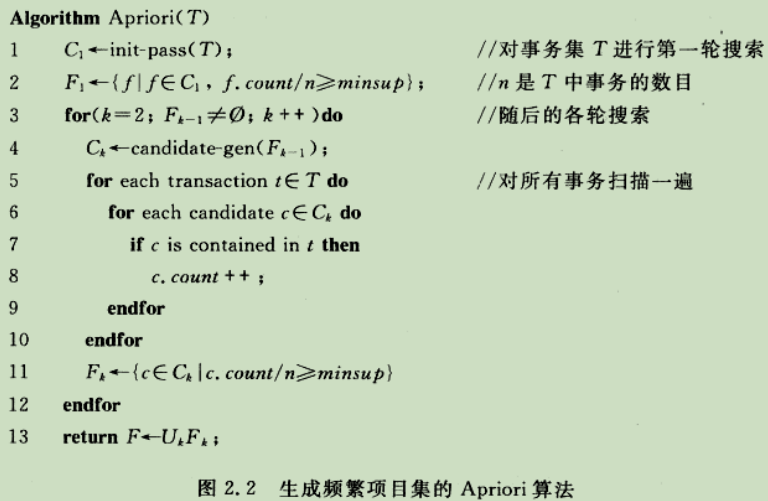
            ['鸡肉', '牛奶', '衣服'],

            ['鸡肉', '衣服', '牛奶']]

## 算法各功能实现：

### 频繁项集生成：

Apriori：利用向下封闭属性逐级搜索得到频繁项集。



'''

参数： 数据集T，最小支持度min\_support

返回：频繁项集集合列表F，所有频繁项集的支持度support\_data

'''

def Apriori(T, min\_support):

    F = []            # 频繁k项集集合Fk的列表

    support\_data = {} # 所有频繁项集支持度，字典结构，键：频繁项集fk，值：fk的支持度

C1 = create\_C1(T) # 对事务集T进行第一轮搜索得到频繁候选1项集C1

F1 = generate\_Fk\_by\_Ck(T, C1, min\_support, support\_data) # C1得到F1

    k = 2

    Fksub1 = F1.copy()

    F.append(Fksub1)

    while(Fksub1):      # k初始为2，Fk-1为F1，直到Fk为空循环结束

        Ck = candidate\_gen(Fksub1, k) # Fk-1合并剪枝得到候选Ck

        Fk = generate\_Fk\_by\_Ck(T, Ck, min\_support, support\_data)# Ck得到Fk

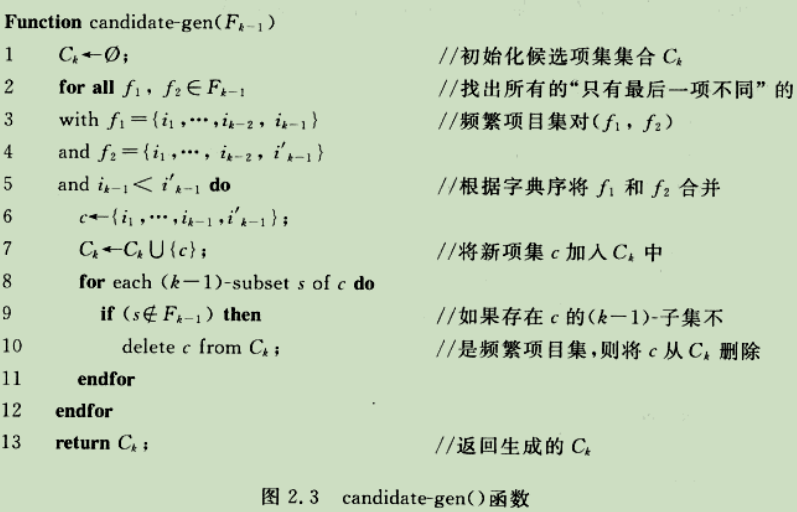
k += 1                        # 跟新k和Fk-1

        Fksub1 = Fk.copy()

        F.append(Fksub1)

    return F, support\_data

Candidate\_gen：频繁k-1项集Fk-1生成频繁候选k项集Ck



'''Fk-1作为种子集合产生Ck，满足Ck的任意(k-1)子集都在Fk-1中'''

def candidate\_gen(Fksub1, k):

    Ck = set()

    len\_Fksub1 = len(Fksub1)

    list\_Fksub1 = list(Fksub1)           # Fk-1转换为列表以便有序查找

    for i in range(len\_Fksub1):

        for j in range(i, len\_Fksub1):   # 不重复地选出待合并的项集f1，f2

            f1 = list(list\_Fksub1[i])

            f2 = list(list\_Fksub1[j])

            f1.sort()

            f2.sort() # 项集转换为列表后排序，以便比较是否只有最后一项不同

# 找出只有最后一项不同的频繁项目集f1、f2

if f1[0:k-2] == f2[0:k-2] and f1[-1] != f2[-1]:

                c = list\_Fksub1[i] | list\_Fksub1[j]    # 合并： f1、f2生成新项集c

                is\_add = 1

                for item in c: # 剪枝：存在c的(k-1)子集不在Fk-1中，就不添加c

                    if((c - frozenset([item])) not in Fksub1):

                        is\_add = 0

                        break

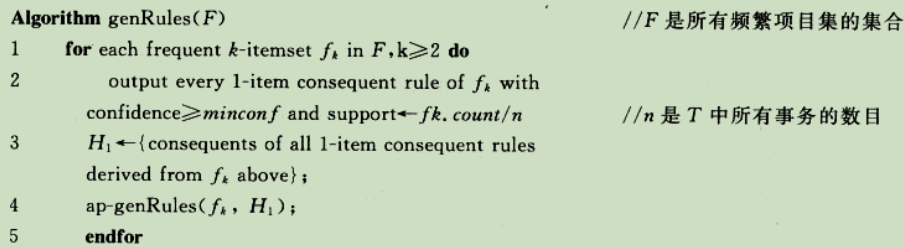
                if(is\_add == 1):

                    Ck.add(c)

    return Ck

### 关联规则生成：

genRules：遍历所有频繁项集，先生成后件1项的关联规则



def genRules(F, support\_data, min\_conf):

    # F是频繁项集集合的列表，最后必有一个空集合，所以频繁k项集k的取值是[1, len(F))

    for k in range(1, len(F)): # 频繁k项集集合：F[k-1]

        for fk in F[k-1]:      # 频繁k项集：   fk

            H1 = set()         # 存关联规则1-后件

            for item in fk:  # 每个频繁k项集中的一项，都可能是关联规则1-后件

# 满足fk / (fk - item) >= min\_conf就输出后件1项的关联规则

                if(fk - frozenset([item]) and support\_data[fk] / support\_data[fk - frozenset([item])] >= min\_conf):

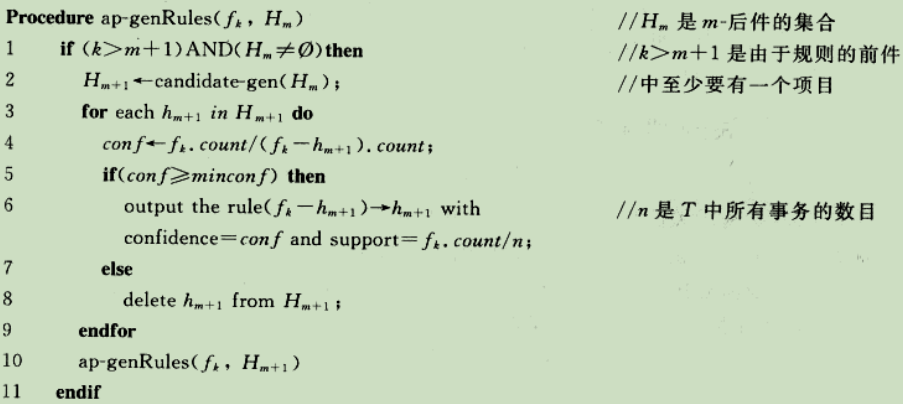
                    print(list(fk - frozenset([item])), ' => ', list(frozenset([item])), '\tsupport: ', support\_data[fk], '\tconf: ', support\_data[fk] / support\_data[fk - frozenset([item])])

                    H1.add(frozenset([item]))

# 利用后件的向下封闭属性继续输出后件多项的关联规则

            ap\_genRules(fk, k, H1, 1, support\_data, min\_conf)

ap\_genRules：利用后件的向下封闭属性继续输出后件多项的关联规则



'''

频繁项集fk生成的关联规则中：

利用关联规则后件的向下封闭属性，后件有m项的后件集合Hm生成后件有m+1项的后件集合Hm+1

并判断是否满足fk的关联规则

如此递归

'''

def ap\_genRules(fk, k, Hm, m, support\_data, min\_conf):

    if(k > m + 1 and Hm):             # 递归终止条件

        HmAdd1 = candidate\_gen(Hm, m) # 满足Hm+1的任意m项子集都在Hm中

        for h in HmAdd1:

            if(support\_data[fk] / support\_data[fk - h] >= min\_conf):

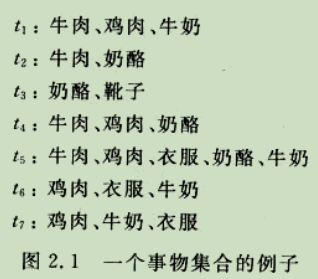
                print(list(fk - h), ' => ', list(h), '\tsupport: ', support\_data[fk], '\tconf: ', support\_data[fk] / support\_data[fk - h])

            else:

                HmAdd1 -= h

        ap\_genRules(fk, k, HmAdd1, m+1, support\_data, min\_conf)

## 书中案例：



数据集：

data\_set = [['牛肉', '鸡肉', '牛奶'],

            ['牛肉', '奶酪'],

            ['靴子', '奶酪'],

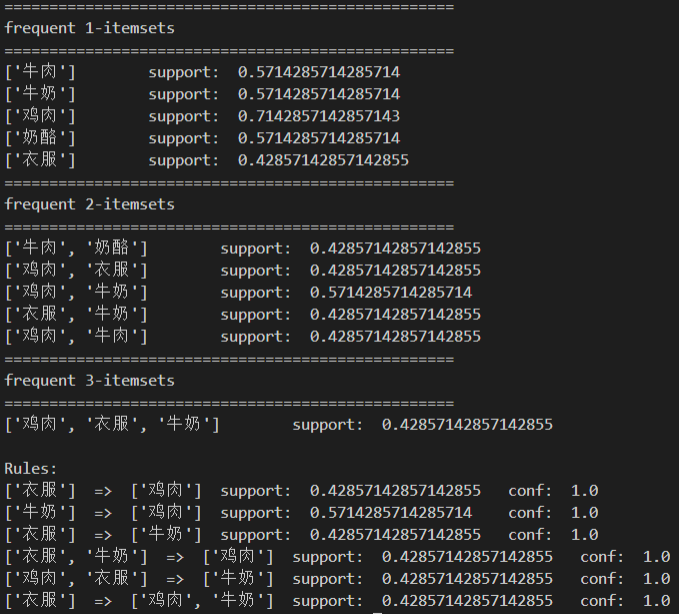
            ['牛肉', '鸡肉', '奶酪'],

            ['牛肉', '鸡肉', '牛奶', '奶酪', '衣服'],

            ['鸡肉', '牛奶', '衣服'],

            ['鸡肉', '衣服', '牛奶']]

当min\_support = 0.3，min\_conf = 0.8时，输出频繁项集和关联规则：



## 最后：

参考：

https://www.cnblogs.com/llhthinker/p/6719779.html

原文代码关联规则的生成采用效率底下的穷举策略。

按照书上的方法：利用后件的向下封闭属性，重新写了关联规则生成代码。

我的实现数据集是直接写死在代码的二维列表里，实际上数据集应该要从外部导入。

我的代码：

https://github.com/SmallBirdsLab/DataMining/blob/main/%E5%85%B3%E8%81%94%E8%A7%84%E5%88%99/Apriori%E7%AE%97%E6%B3%95/apriori.py