Apriori关联分析案例——购物车的关联分析

链接：<https://blog.csdn.net/weixin_48077303/article/details/115537001>

## 一、Apriori关联分析的介绍

在机器学习中，除了聚类算法外，Aprior算法也是在数据集中寻找数据之间的某种关联关系，通过该算法，我们可以在大规模的数据中发现有价值的价值，比如著名的啤酒与尿布的案例就是一种关联分析。本文将详细介绍Apriori算法挖掘数据的原理以及案例。

### 1.Apriori算法的原理

#### 1.1 项集

项集是项的集合，包含k个项的集合称为k项集，如{啤酒，尿布}就是个2项集。

项集在所有事务中出现的次数总和称为绝对支持度或支持度计数。

频繁项集：某项集的支持度计算满足预定的要求即称该项集为频繁项集。

#### 1.2 关联规则

支持度：项集A和B的支持度被定义为数据集中同时包含这两项集的记录所占的比例。（通俗理解，就是事件A和B同时发生的概率），记做：

在这里插入图片描述

可信度（置信度）：项集A发生，则项集B发生的概率为关联规则的置信度（通俗理解,在A发生的情况下B发生的概率为多少P（B/A））记做：

在这里插入图片描述

#### 1.3 最小支持度和最小置信度

最小支持度：即衡量支持度的一个阈值，表示项集之间的支持度满足该阈值才能证明该支持度有效。

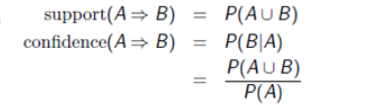
最小置信度：即衡量置信度的一个阈值，表示项集之间的置信度满足该阈值才能证明该置信度有效。

强规则：同时满足最小支持度和最小置信度规则的规则。

#### 1.4 支持度计数

项集A的支持度计数是项集A在所有项集中出现的次数（也称频数和计数）。

根据项集A,B的支持度计数可以计算出项集A,B的支持度和置信度。



Ps：support可以理解为A,B同时发生的概率。confidence可以理解为条件概率。

### 2.实例说明

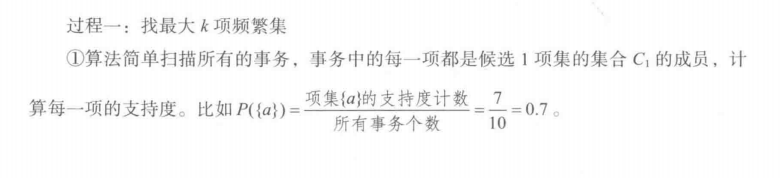
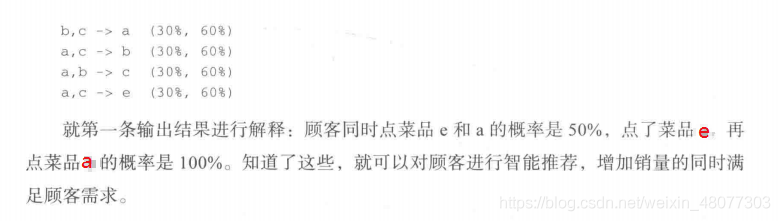
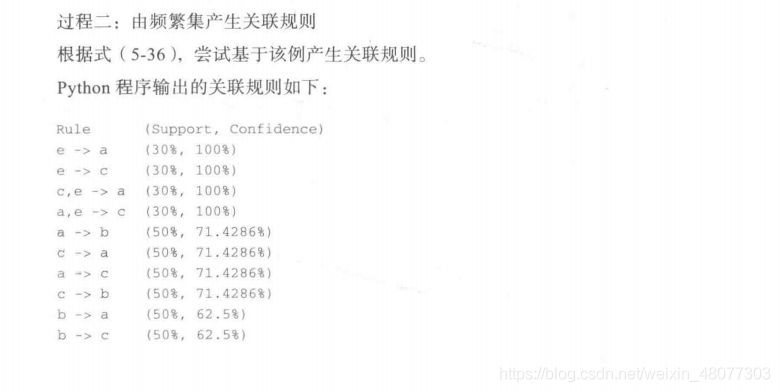
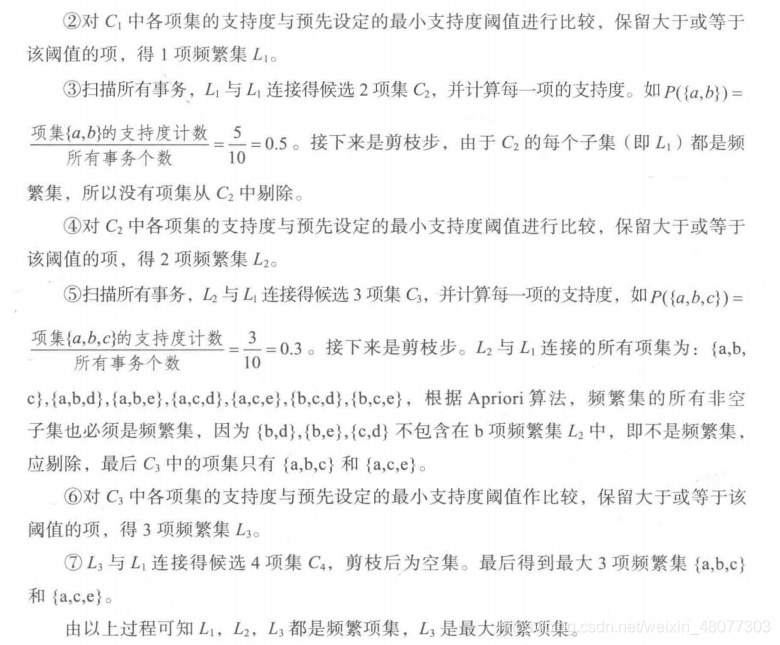
这里我们首先需要初始化一些参数，如：设置最小支持为0.2



#### 2.1计算 1项集，2项集，3项集



#### 2.2简述过程：

  
3.代码实现

#### 3.1定义Apriori算法过程：

**# -\*- coding: utf-8 -\*-**

**import numpy as np**

**import pandas as pd**

**#改文件是定义Apriori的算法过程**

**def connect\_string(x, ms):**

**x = list(map(lambda i: sorted(i.split(ms)), x))**

**l = len(x[0])**

**r = []**

**for i in range(len(x)):**

**for j in range(i, len(x)):**

**if x[i][:l - 1] == x[j][:l - 1] and x[i][l - 1] != x[j][l - 1]:**

**r.append(x[i][:l - 1] + sorted([x[j][l - 1], x[i][l - 1]]))**

**return r**

**# 寻找关联规则的函数**

**def find\_rule(d, support, confidence, ms=u'--'):**

**result = pd.DataFrame(index=['support', 'confidence']) # 定义输出结果**

**support\_series = 1.0 \* d.sum() / len(d) # 支持度序列**

**column = list(support\_series[support\_series > support].index) # 初步根据支持度筛选**

**k = 0**

**while len(column) > 1:**

**k = k + 1**

**print(u'\n正在进行第%s次搜索...' % k)**

**column = connect\_string(column, ms)**

**print(u'数目：%s...' % len(column))**

**sf = lambda i: d[i].prod(axis=1, numeric\_only=True) # 新一批支持度的计算函数**

**# 创建连接数据，这一步耗时、耗内存最严重。当数据集较大时，可以考虑并行运算优化。**

**d\_2 = pd.DataFrame(list(map(sf, column)), index=[ms.join(i) for i in column]).T**

**support\_series\_2 = 1.0 \* d\_2[[ms.join(i) for i in column]].sum() / len(d) # 计算连接后的支持度**

**column = list(support\_series\_2[support\_series\_2 > support].index) # 新一轮支持度筛选**

**support\_series = support\_series.append(support\_series\_2)**

**column2 = []**

**for i in column: # 遍历可能的推理，如{A,B,C}究竟是A+B-->C还是B+C-->A还是C+A-->B？**

**i = i.split(ms)**

**for j in range(len(i)):**

**column2.append(i[:j] + i[j + 1:] + i[j:j + 1])**

**cofidence\_series = pd.Series(index=[ms.join(i) for i in column2]) # 定义置信度序列**

**for i in column2: # 计算置信度序列**

**cofidence\_series[ms.join(i)] = support\_series[ms.join(sorted(i))] / support\_series[ms.join(i[:len(i) - 1])]**

**for i in cofidence\_series[cofidence\_series > confidence].index: # 置信度筛选**

**result[i] = 0.0**

**result[i]['confidence'] = cofidence\_series[i]**

**result[i]['support'] = support\_series[ms.join(sorted(i.split(ms)))]**

**result = result.T.sort\_values(['confidence', 'support'], ascending=False) # 结果整理，输出**

**print(u'\n结果为：')**

**print(result)**

**return result**

#### 3.2导入数据进行关联分析

**# -\*- coding: utf-8 -\*-**

**from \_\_future\_\_ import print\_function**

**import pandas as pd**

**from apriori import \* # 导入自行编写的apriori函数**

**inputfile = '../data/menu\_orders.xls'**

**outputfile = '../tmp/apriori\_rules.xls' # 结果文件**

**data = pd.read\_excel(inputfile, header = None)**

**print(u'\n转换原始数据至0-1矩阵...')**

**ct = lambda x : pd.Series(1, index = x[pd.notnull(x)]) # 转换0-1矩阵的过渡函数**

**b = map(ct, data.as\_matrix()) # 用map方式执行**

**data = pd.DataFrame(list(b)).fillna(0) # 实现矩阵转换，空值用0填充**

**print(u'\n转换完毕。')**

**del b # 删除中间变量b，节省内存**

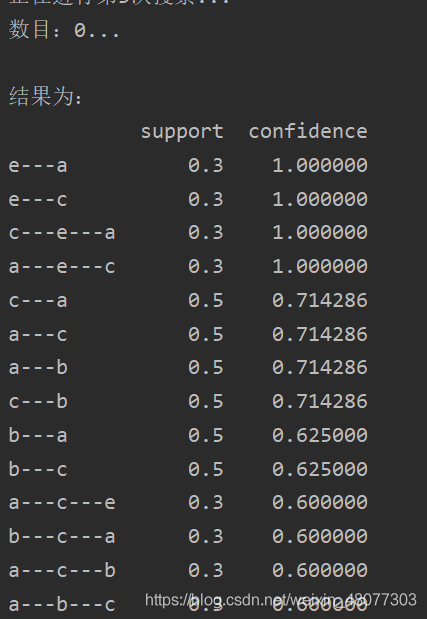
**support = 0.2 # 最小支持度**

**confidence = 0.5 # 最小置信度**

**ms = '---' # 连接符，默认'--'，用来区分不同元素，如A--B。需要保证原始表格中不含有该字符**

**find\_rule(data, support, confidence, ms).to\_excel(outputfile) # 保存结果**

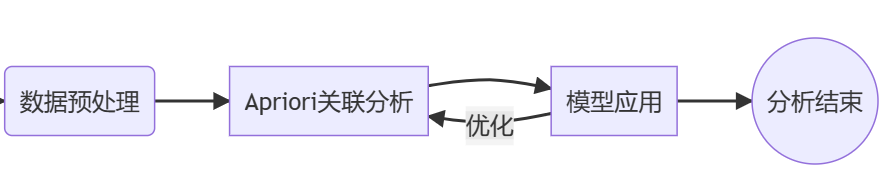
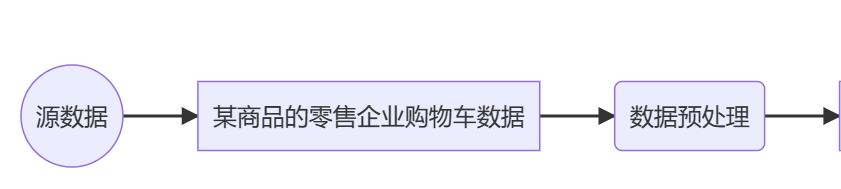
#### 3.3结果如下



## 二、案例背景与分析过程

现代商品种类繁多，顾客往往会纠结选择买啥，尤其对于选择困难症的顾客来说，选择商品更是难上加难。繁杂的选购往往会给顾客带来疲惫的购物体验。对于部分商品，比如面包和牛奶，薯片和可乐等，是顾客经常同时购买的东西，当这些东西距离十分遥远时，就会减小顾客的购买欲望。因此，为了获取最大销售利润，我们需要了解销售什么样的商品、采用什么样的促销手段、商品在货架上如何摆放以及了解顾客购买习惯和偏好等对销售商尤其重要。

分析目的:构建商品的Apriori模型，分析商品之间的关联关系。根据模型结果给出销售策略。



具体步骤如下：

查看原始数据的形式。

对原始数据进行数据的预处理，转换数据形式，使其满足Apriori关联分析的要求。

建立Apriori模型，调整超参数。

对模型结果进行分析。提供销售建议。

### 1.数据的初步处理

对本案例的数据先要有个大致的了解，本案例一共有9835个购物篮数据，一共有两个表，它主要包括3个属性：

****

用python读取数据并查看数据特征。

**#此文件是Apriori模型的主要文件**

**import pandas as pd**

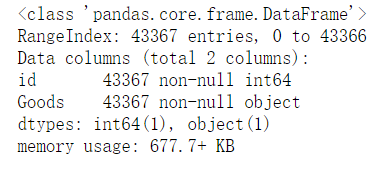
**file\_path=open("....这里自己填文件路径//GoodsOrder.csv")**

**data=pd.read\_csv(file\_path)**

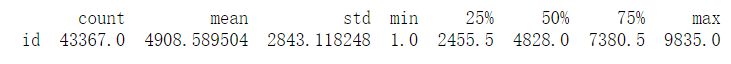
**data.info()**

**des=pd.DataFrame(data.describe()).T**

**print(des)#查看信息描述**



一共有43367个观测值，不存在缺失值id最大值为9835，说明一共有9835个购物篮信息。



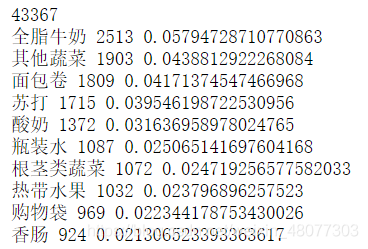
查看销量占比

**#销量占比**

**data\_nums = data.shape[0]**

**for idnex, row in sorted[:10].iterrows():#iterrows就是pd的相当于enumerate依次数pd的行索引和数据**

**print(row['Goods'],row['id'],row['id']/data\_nums)**



### 2.对商品进一步分析

在步骤1中，我们分析了一些具体的商品，这样的分类对后续决策的执行存在一定的困难，我们应该将这些具体的商品和自己所属的大类联系起来。我们通过Good Types表中的数据找出对应商品所属的大类，就相当于数据库中的相关联。

**#各类别商品的销量及其占比**

**import pandas as pd**

**inputfile1 =open("C://Users//Administrator//Desktop//python-code//《Python数据分析与挖掘实战（第2版）》源数据和代码//《Python数据分析与挖掘实战（第2版）》源数据和代码-各章节//chapter8//demo//data//GoodsOrder.csv")**

**inputfile2 =open("C://Users//Administrator//Desktop//python-code//《Python数据分析与挖掘实战（第2版）》源数据和代码//《Python数据分析与挖掘实战（第2版）》源数据和代码-各章节//chapter8//demo//data//GoodsTypes.csv")**

**data = pd.read\_csv(inputfile1)**

**types = pd.read\_csv(inputfile2) # 读入数据**

**group = data.groupby(['Goods']).count().reset\_index()**

**sort = group.sort\_values('id',ascending = False).reset\_index()**

**data\_nums = data.shape[0] # 总量**

**#print(sort)**

**del sort['index']**

**sort\_links = pd.merge(sort,types) # 合并两个datafreame 根据两表中相同的type**

**# 根据类别求和，每个商品类别的总量，并排序**

**sort\_link = sort\_links.groupby(['Types']).sum().reset\_index()**

**sort\_link = sort\_link.sort\_values('id',ascending = False).reset\_index()**

**del sort\_link['index'] # 删除“index”列**

**# 求百分比，然后更换列名，最后输出到文件**

**sort\_link['count'] = sort\_link.apply(lambda line: line['id']/data\_nums,axis=1)**

**sort\_link.rename(columns = {'count':'percent'},inplace = True)**

**print('各类别商品的销量及其占比:\n',sort\_link)**

**# 画饼图展示每类商品销量占比**

**import matplotlib.pyplot as plt**

**data = sort\_link['percent']**

**labels = sort\_link['Types']**

**plt.figure(figsize=(8, 6)) # 设置画布大小**

**plt.pie(data,labels=labels,autopct='%1.2f%%')**

**plt.rcParams['font.sans-serif'] = 'SimHei'**

**plt.title('每类商品销量占比') # 设置标题**

**#plt.savefig('../tmp/persent.png') # 把图片以.png格式保存**

**plt.show()**

将每个商品具体分类后，得到如下的大分类表。  


根据我们的分析，非酒精类饮料占比最高，为17.51%，为销售的大军，所以我们对非酒精类饮料的商品结构进行分析。

**selected = sort\_links.loc[sort\_links['Types'] == '非酒精饮料'] # 挑选商品类别为“非酒精饮料”并排序**

**child\_nums = selected['id'].sum() # 对所有的“非酒精饮料”求和**

**selected['child\_percent'] = selected.apply(lambda line: line['id']/child\_nums,axis = 1) # 求百分比**

**selected.rename(columns = {'id':'count'},inplace = True)**

**print('非酒精饮料内部商品的销量及其占比:\n',selected)**

**# outfile2 = '../tmp/child\_percent.csv'**

**# sort\_link.to\_csv(outfile2,index = False,header = True,encoding='gbk') # 输出结果**

**# 画饼图展示非酒精饮品内部各商品的销量占比**

**import matplotlib.pyplot as plt**

**data = selected['child\_percent']**

**labels = selected['Goods']**

**plt.figure(figsize = (8,6)) # 设置画布大小**

**explode = (0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.08,0.3,0.1,0.3) # 设置每一块分割出的间隙大小**

**plt.pie(data,explode = explode,labels = labels,autopct = '%1.2f%%',**

**pctdistance = 1.1,labeldistance = 1.2)**

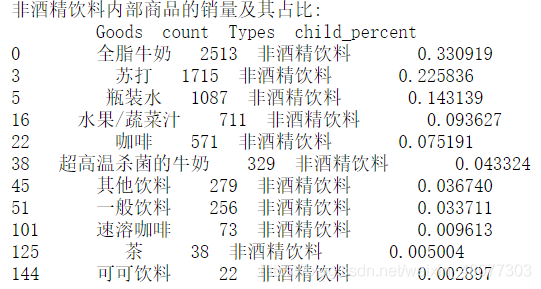
**plt.rcParams['font.sans-serif'] = 'SimHei'**

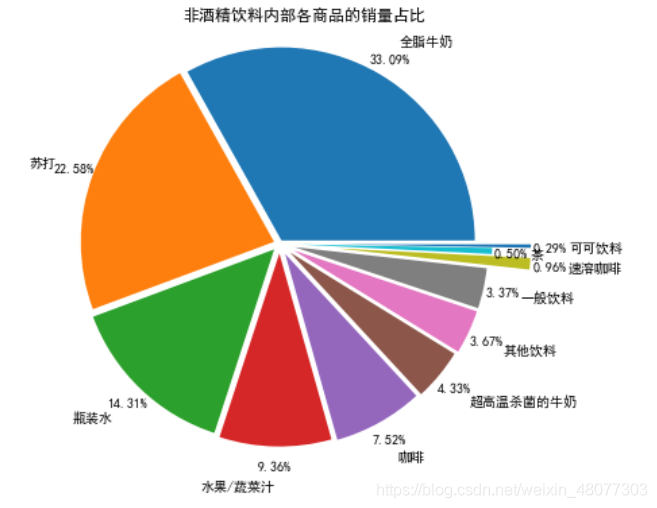
**plt.title("非酒精饮料内部各商品的销量占比") # 设置标题**

**plt.axis('equal')**

**# plt.savefig('../tmp/child\_persent.png') # 保存图形**

**plt.show() # 展示图形**





这里我们在做一下数据转化，以前的数据每一行代表每一件商品,以及对应的id

**import pandas as pd**

**inputfile =open("C://Users//Administrator//Desktop//python-code//《Python数据分析与挖掘实战（第2版）》源数据和代码//《Python数据分析与挖掘实战（第2版）》源数据和代码-各章节//chapter8//demo//data//GoodsOrder.csv")**

**data = pd.read\_csv(inputfile)**

**# 根据id对“Goods”列合并，并使用“，”将各商品隔开**

**data['Goods'] = data['Goods'].apply(lambda x:','+x)**

**print(data.head(10))**

**data = data.groupby('id').sum().reset\_index()**

**print(data.head(10))**

**# 对合并的商品列转换数据格式**

**data['Goods'] = data['Goods'].apply(lambda x :[x[1:]])#转化为数组**

**data\_list = list(data['Goods'])**

**# 分割商品名为每个元素**

**data\_translation = []**

**for i in data\_list:**

**p = i[0].split(',')**

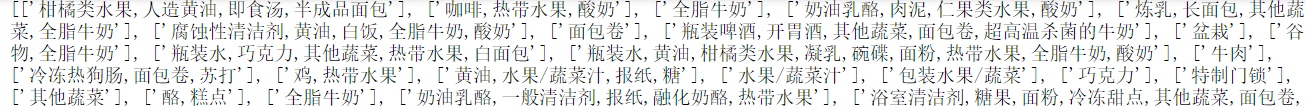
**data\_translation.append(p)**

**print('数据转换结果的前5个元素：\n', data\_translation[0:5])**

原数据：



转换后的数据：  
即将id为1的顾客所买的所有商品转换成一个列表。同理，id为2的顾客买的…



### 3.关联分析

#### 3.1建模实现的步骤：

首先设置Apriori的最小支持度、最小置信度、输入建模样本数据

采用Apriori关联分析算法对建模样本数据进行分析，以模型设置的参数作为条件和目标

目前如何设置最小支持度与最小置信度设置并没有统一的大小的标准，大部分都是通过设置初始值，然后通过不断的调整，获取与业务相符的关联结果。本文输入的模型参数为：最小支持度0.02，最小置信度为0.35。

**# -\*- coding: utf-8 -\*-**

**# 代码8-6 构建关联规则模型**

**from numpy import \***

**def loadDataSet():**

**return [['a', 'c', 'e'], ['b', 'd'], ['b', 'c'], ['a', 'b', 'c', 'd'], ['a', 'b'], ['b', 'c'], ['a', 'b'],**

**['a', 'b', 'c', 'e'], ['a', 'b', 'c'], ['a', 'c', 'e']]**

**def createC1(dataSet):**

**C1 = []**

**for transaction in dataSet:**

**for item in transaction:**

**if not [item] in C1:**

**C1.append([item])**

**C1.sort()**

**# 映射为frozenset唯一性的，可使用其构造字典**

**return list(map(frozenset, C1))**

**# 从候选K项集到频繁K项集（支持度计算）**

**def scanD(D, Ck, minSupport):**

**ssCnt = {}**

**for tid in D: # 遍历数据集**

**for can in Ck: # 遍历候选项**

**if can.issubset(tid): # 判断候选项中是否含数据集的各项**

**if not can in ssCnt:**

**ssCnt[can] = 1 # 不含设为1**

**else:**

**ssCnt[can] += 1 # 有则计数加1**

**numItems = float(len(D)) # 数据集大小**

**retList = [] # L1初始化**

**supportData = {} # 记录候选项中各个数据的支持度**

**for key in ssCnt:**

**support = ssCnt[key] / numItems # 计算支持度**

**if support >= minSupport:**

**retList.insert(0, key) # 满足条件加入L1中**

**supportData[key] = support**

**return retList, supportData**

**def calSupport(D, Ck, min\_support):**

**dict\_sup = {}**

**for i in D:**

**for j in Ck:**

**if j.issubset(i):**

**if not j in dict\_sup:**

**dict\_sup[j] = 1**

**else:**

**dict\_sup[j] += 1**

**sumCount = float(len(D))**

**supportData = {}**

**relist = []**

**for i in dict\_sup:**

**temp\_sup = dict\_sup[i] / sumCount**

**if temp\_sup >= min\_support:**

**relist.append(i)**

**# 此处可设置返回全部的支持度数据（或者频繁项集的支持度数据）**

**supportData[i] = temp\_sup**

**return relist, supportData**

**# 改进剪枝算法**

**def aprioriGen(Lk, k):**

**retList = []**

**lenLk = len(Lk)**

**for i in range(lenLk):**

**for j in range(i + 1, lenLk): # 两两组合遍历**

**L1 = list(Lk[i])[:k - 2]**

**L2 = list(Lk[j])[:k - 2]**

**L1.sort()**

**L2.sort()**

**if L1 == L2: # 前k-1项相等，则可相乘，这样可防止重复项出现**

**# 进行剪枝（a1为k项集中的一个元素，b为它的所有k-1项子集）**

**a = Lk[i] | Lk[j] # a为frozenset()集合**

**a1 = list(a)**

**b = []**

**# 遍历取出每一个元素，转换为set，依次从a1中剔除该元素，并加入到b中**

**for q in range(len(a1)):**

**t = [a1[q]]**

**tt = frozenset(set(a1) - set(t))**

**b.append(tt)**

**t = 0**

**for w in b:**

**# 当b（即所有k-1项子集）都是Lk（频繁的）的子集，则保留，否则删除。**

**if w in Lk:**

**t += 1**

**if t == len(b):**

**retList.append(b[0] | b[1])**

**return retList**

**def apriori(dataSet, minSupport=0.2):**

**# 前3条语句是对计算查找单个元素中的频繁项集**

**C1 = createC1(dataSet)**

**D = list(map(set, dataSet)) # 使用list()转换为列表**

**L1, supportData = calSupport(D, C1, minSupport)**

**L = [L1] # 加列表框，使得1项集为一个单独元素**

**k = 2**

**while (len(L[k - 2]) > 0): # 是否还有候选集**

**Ck = aprioriGen(L[k - 2], k)**

**Lk, supK = scanD(D, Ck, minSupport) # scan DB to get Lk**

**supportData.update(supK) # 把supk的键值对添加到supportData里**

**L.append(Lk) # L最后一个值为空集**

**k += 1**

**del L[-1] # 删除最后一个空集**

**return L, supportData # L为频繁项集，为一个列表，1，2，3项集分别为一个元素**

**# 生成集合的所有子集**

**def getSubset(fromList, toList):**

**for i in range(len(fromList)):**

**t = [fromList[i]]**

**tt = frozenset(set(fromList) - set(t))**

**if not tt in toList:**

**toList.append(tt)**

**tt = list(tt)**

**if len(tt) > 1:**

**getSubset(tt, toList)**

**def calcConf(freqSet, H, supportData, ruleList, minConf=0.7):**

**for conseq in H: #遍历H中的所有项集并计算它们的可信度值**

**conf = supportData[freqSet] / supportData[freqSet - conseq] # 可信度计算，结合支持度数据**

**# 提升度lift计算lift = p(a & b) / p(a)\*p(b)**

**lift = supportData[freqSet] / (supportData[conseq] \* supportData[freqSet - conseq])**

**if conf >= minConf and lift > 1:**

**print(freqSet - conseq, '-->', conseq, '支持度', round(supportData[freqSet], 6), '置信度：', round(conf, 6),**

**'lift值为：', round(lift, 6))**

**ruleList.append((freqSet - conseq, conseq, conf))**

**# 生成规则**

**def gen\_rule(L, supportData, minConf = 0.7):**

**bigRuleList = []**

**for i in range(1, len(L)): # 从二项集开始计算**

**for freqSet in L[i]: # freqSet为所有的k项集**

**# 求该三项集的所有非空子集，1项集，2项集，直到k-1项集，用H1表示，为list类型,里面为frozenset类型，**

**H1 = list(freqSet)**

**all\_subset = []**

**getSubset(H1, all\_subset) # 生成所有的子集**

**calcConf(freqSet, all\_subset, supportData, bigRuleList, minConf)**

**return bigRuleList**

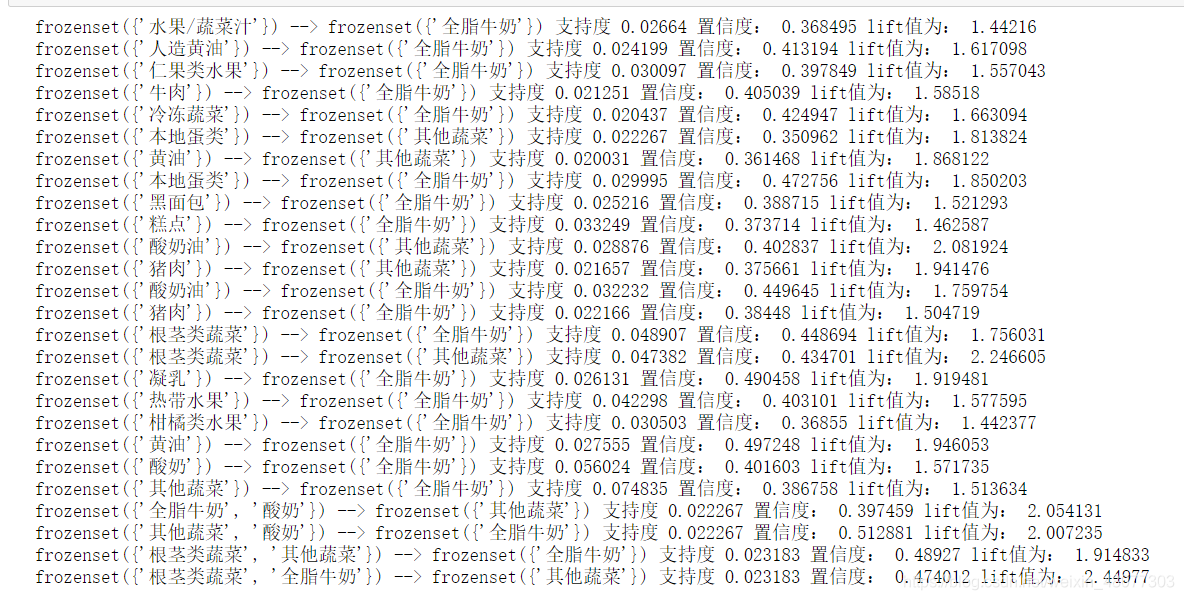
**if \_\_name\_\_ == '\_\_main\_\_':**

**dataSet = data\_translation**

**L, supportData = apriori(dataSet, minSupport = 0.02)**

**rule = gen\_rule(L, supportData, minConf = 0.35)**

#### 3.2结果如下：



我们对输出的结果进行解释：就拿第一条吧：

在这里插入图片描述

这说明同时购买（果汁/蔬菜汁）和（全脂牛奶）这两件商品的概率为36.85%，这种情况发生的可能性大小为2.66%

后续的结果可以进行相同的解释

根据模型的结果，我们发现大多购物者 买的东西主要为食品，随着生活水平的提高和健康意识的增加，其他蔬菜、根茎类蔬菜和全脂牛奶为现代家庭每日饮食所需品，因此，其他类、根茎类蔬菜和全脂牛奶同时购买的概率较高，符合现在人们的生活健康意识

### 4.模型应用

以上模型结果表明:顾客购买其他商品的时候会同时购买全脂牛奶，因此商场应该根据实际情况将全脂牛奶放在顾客的必经之路上，或是显眼的地方，以便顾客拿取。同时，顾客购买其他蔬菜、根茎类蔬菜、酸牛奶、猪肉、黄油、本地蛋类和多种水果概率较高，因此商场可以考虑捆绑销售，或者调整商品布局，将这些商品距离尽量拉近，从而提升顾客的购物体验。