1．实验目的

（1）掌握Apriori算法基本原理

（2）能够用Apriori算法挖掘关联规则与强关联规则

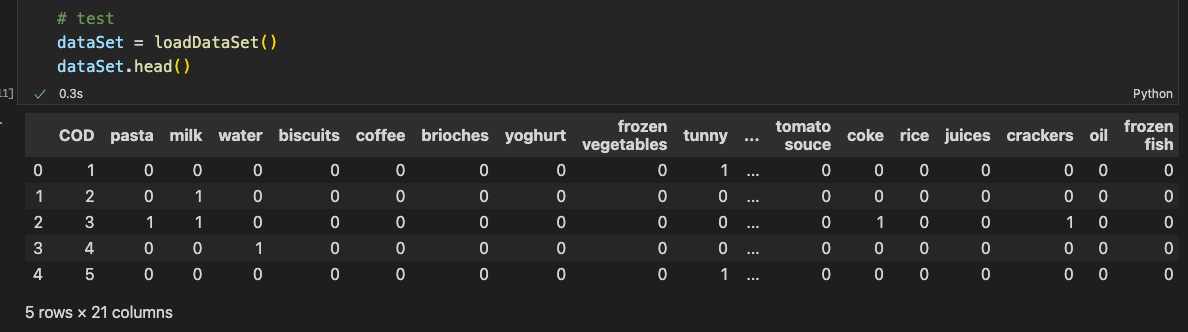
（3）理解Apriori算法的优缺点，理解频繁项与置信度

（4）能够对Apriori算法提出优化改进方案

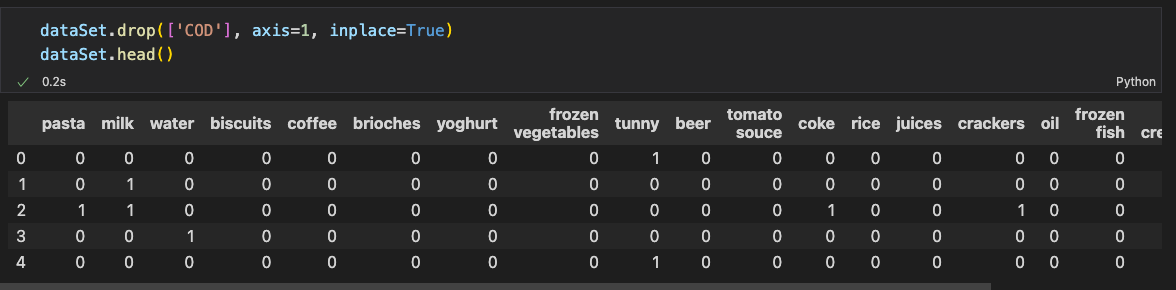
2．实验要求和步骤

（1）读取Transactions.txt数据，完成数据预处理。建议对原始项目建立编码映射表。

调用Pandas库，读入其中的文件中的数据

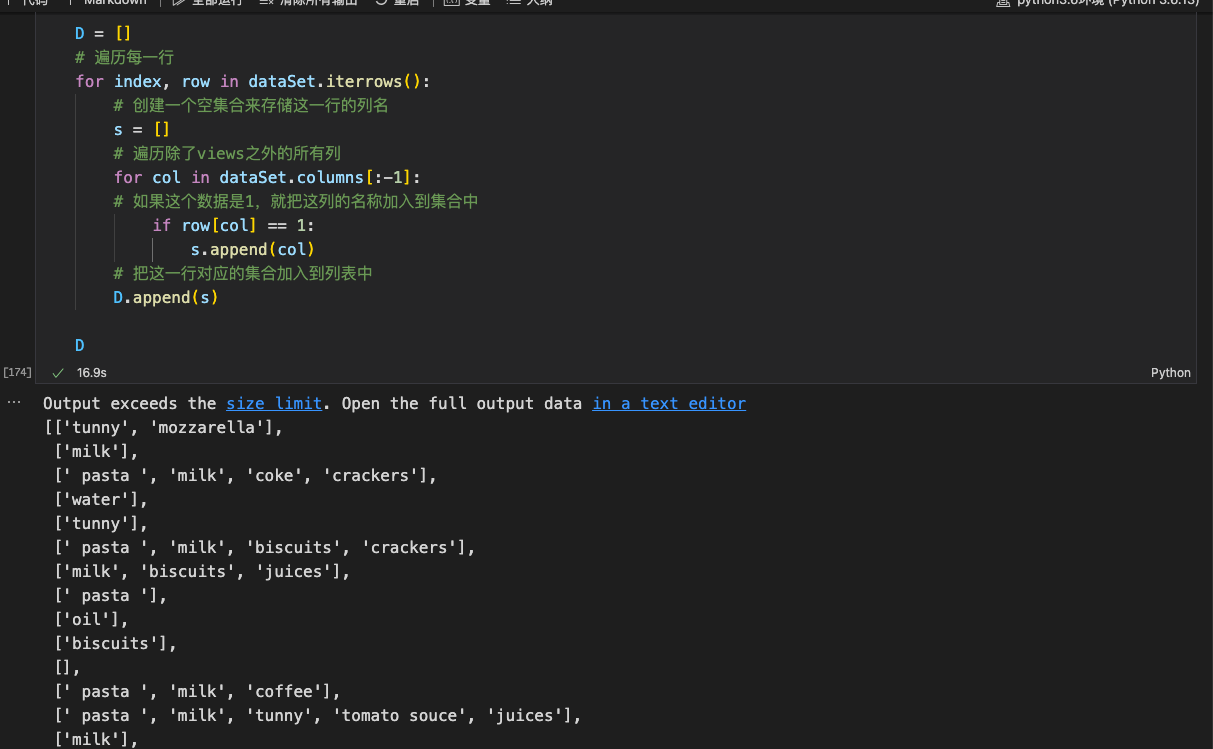


随后将用于表示序号的第一列去掉



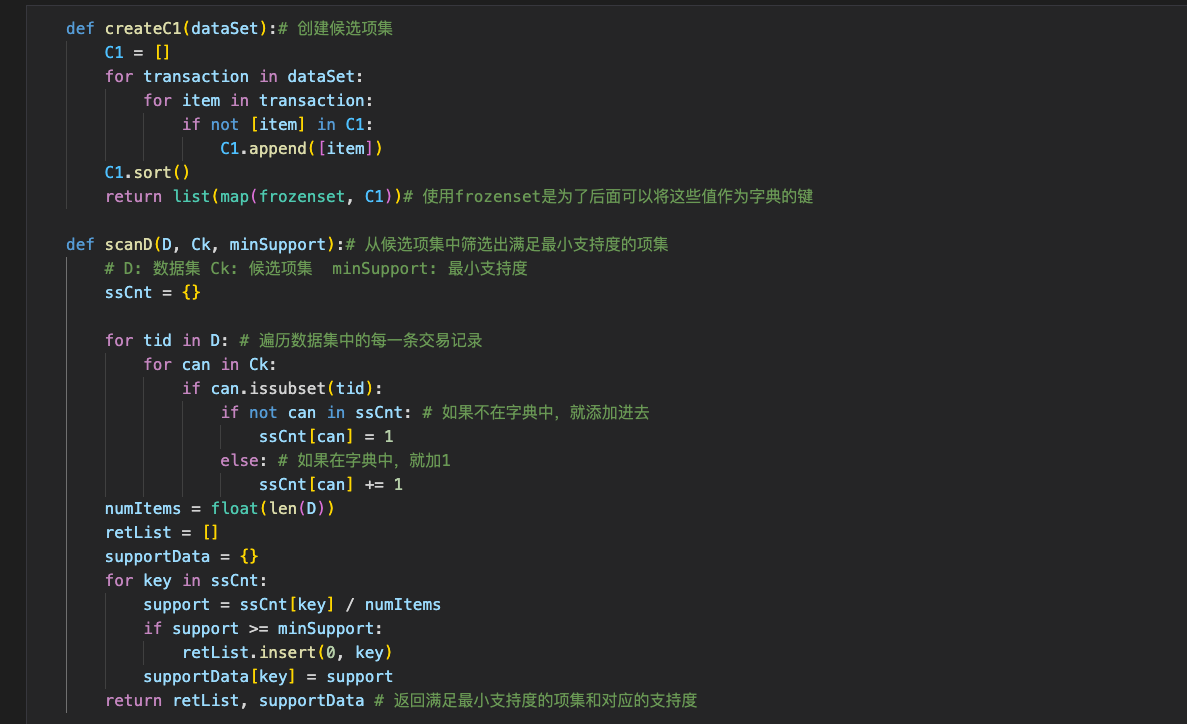
1. 采用合理的数据结构存储交易数据的各项目集。

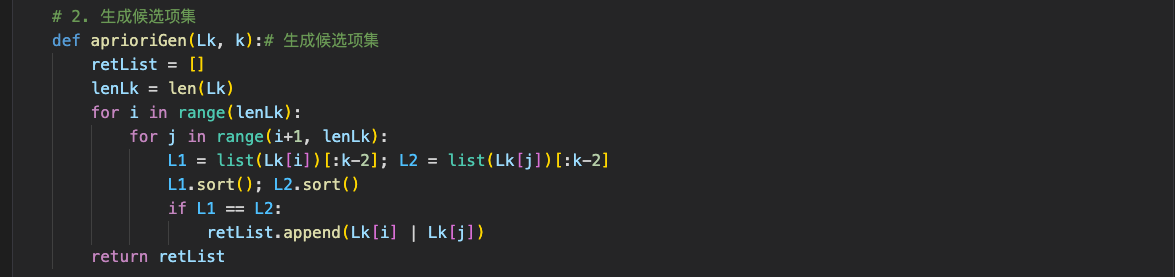
为了便于后面的处理，将读入的表格转化为list类型，每一行视为一个list，如果这一行上对应的位置是1就将列名加入到list当中，在处理之后得到如下的结果

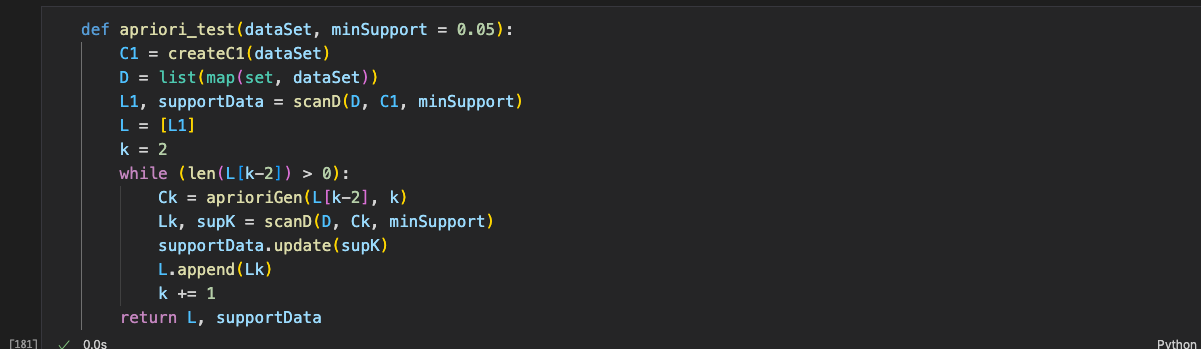


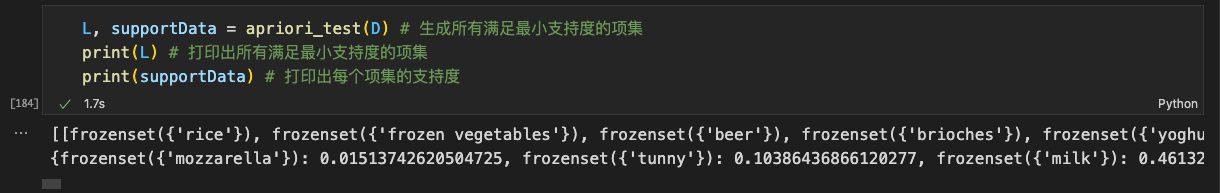
1. 利用Aprioir算法挖掘关联规则。

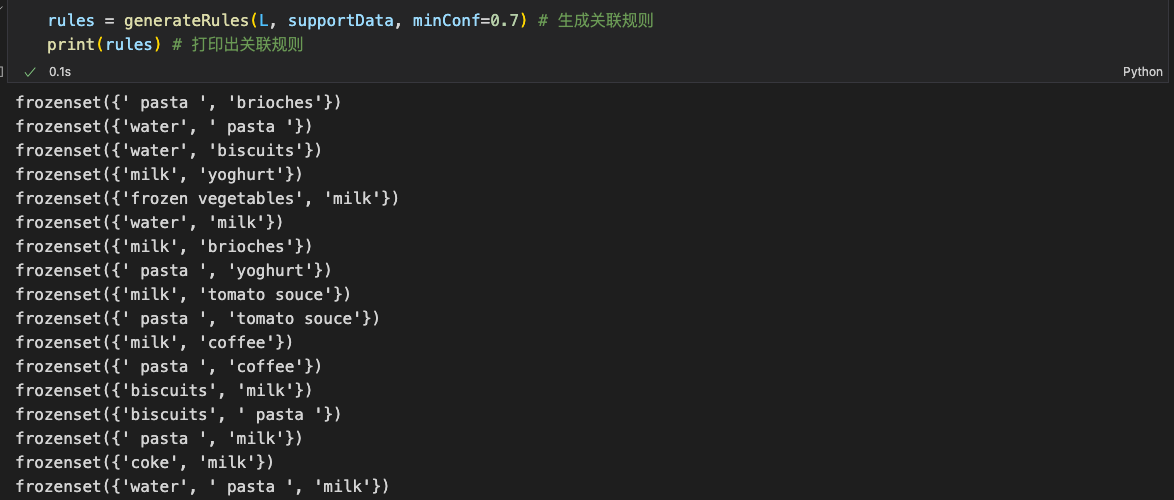
编写Aprior 算法:









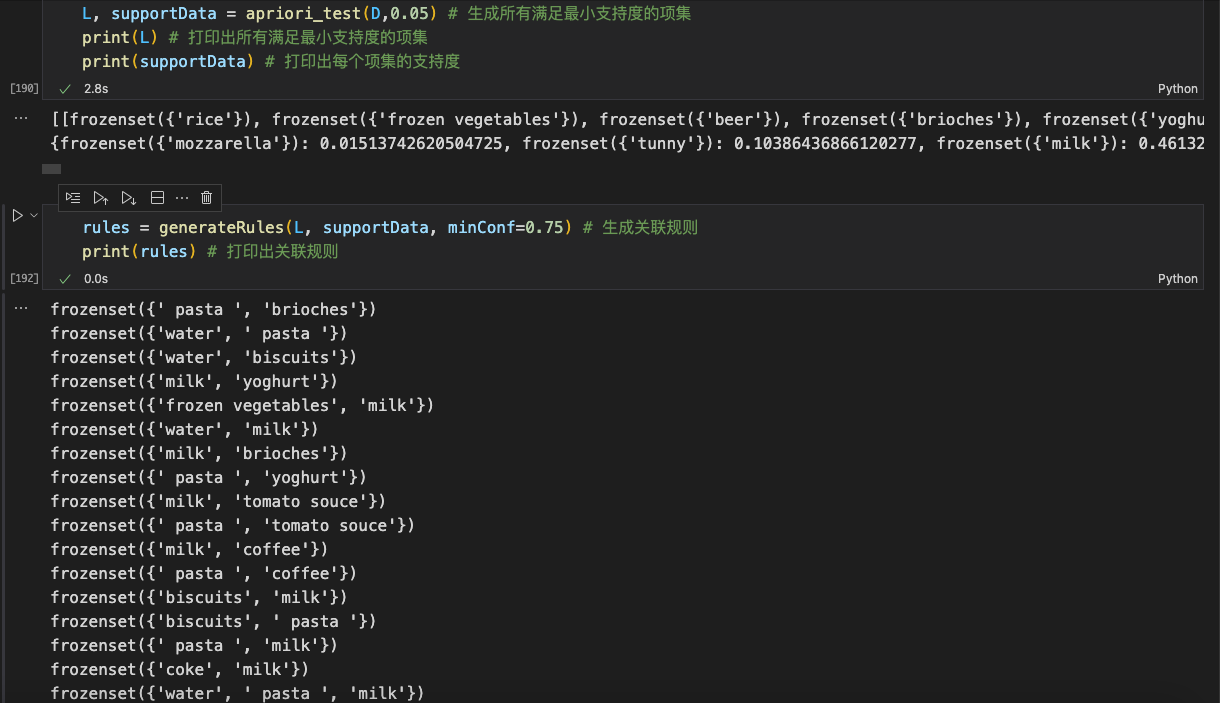


1. 调整支持度和置信度，直至获得满意的关联规则集合。

在最开始进行测试的时候我将支持度和置信度分别设为0.05 和0.7，为了得到置信度更高一点的规则，我将支持度改为0.1和0.8，这个时候没有任何满足的规则被输出



随后我接着对于支持度和置信度进行调整，将支持度维持在0.05，将置信度调整为0.75，可以得到下面的情形：

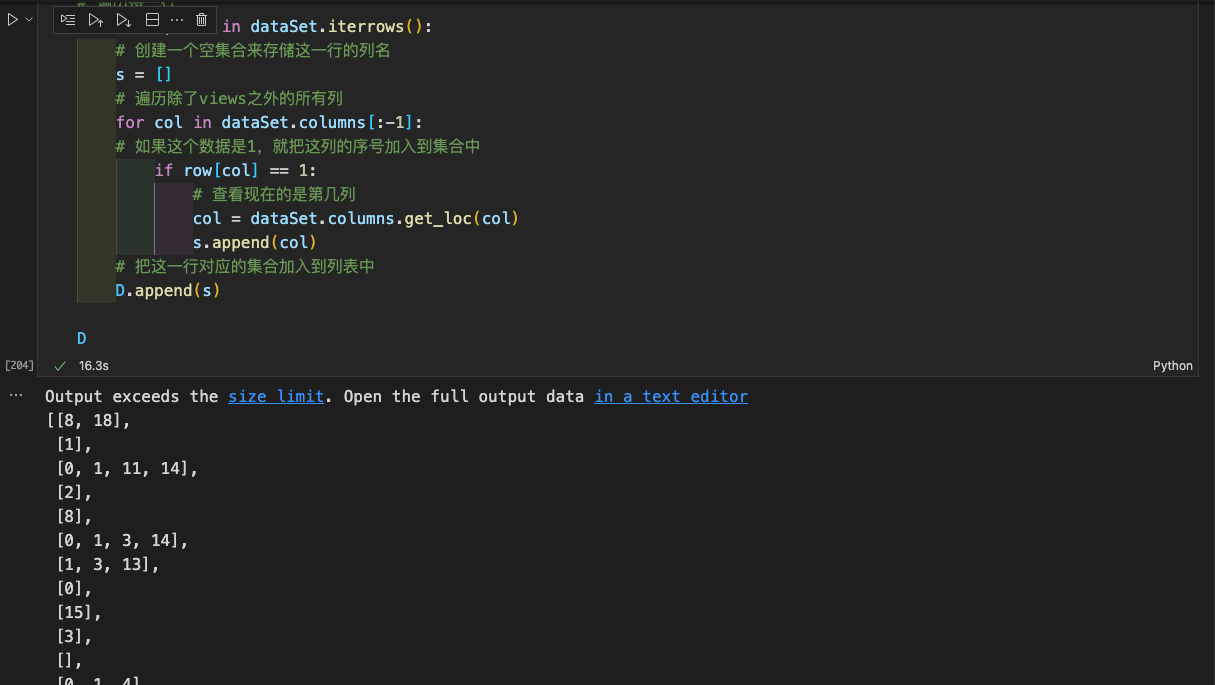


进一步将支持度提升为0.07，保持置信度为0.7得到下面的结果



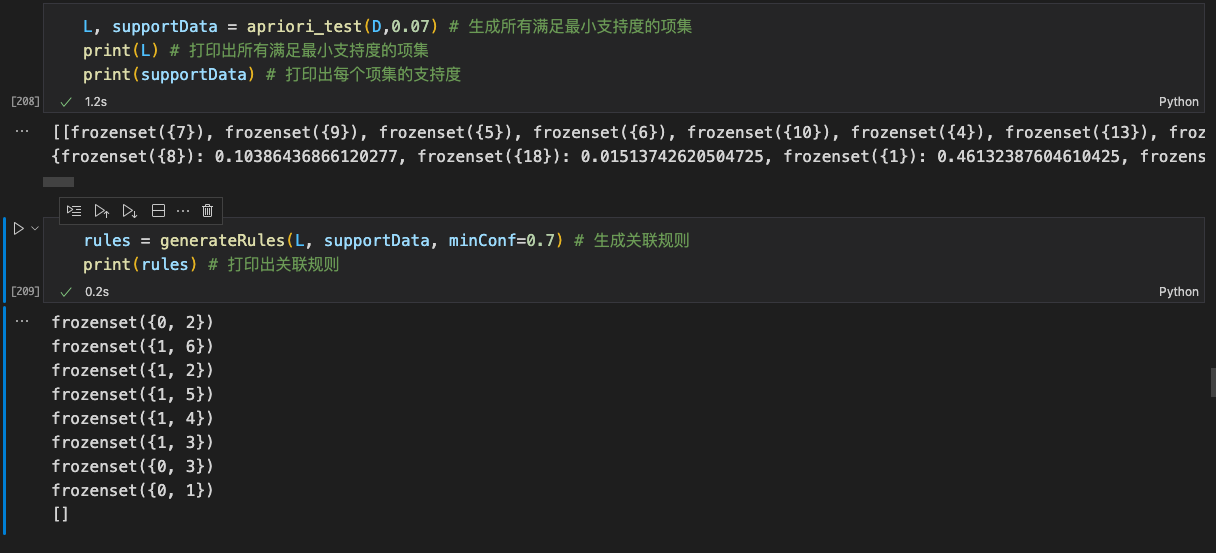
1. 分析算法瓶颈，设计一种改进策略，并尝试实现。

在运算的时候发现由于数据集比较大，因此扫描数据集比较浪费时间。为了减少磁盘IO次数，需要对于数据集进行进一步压缩，在之前处理的过程中，我选择的直接将用字符串来存储，耗费了不必要耗费的存储空间，对于这个做进一步的处理，将其中的字符串全部采用数字来代替



（6）对比优化前、后的算法性能，包括时间开销与空间开销。

在同样的支持度和置信度下在之前生成规则需要2.7s，而在进行改进之后需要1.2s，时间开销上提升明显，而采用数字代替字符串进行存储可以得到显然空间开销会更小



3．分析与讨论

（1）讨论进一步优化Apriori的技术方案。

1.应该设置合理的支持度和置信度的值，在开始我选择了过高的置信度，导致了算法无法正常工作，因为在数据集当中无法得到有效的结果

2.应该尽可能压缩数据集的大小，减少磁盘的IO次数以使得程序消耗更小的时间开销

3. Aprior算法是一种经典的挖掘频繁项集和关联规则的数据挖掘算法12。它基于一个先验性质，即频繁项集的所有非空子集也一定是频繁的。Aprior算法使用一种称为逐层搜索的迭代方法，从频繁项集开始，通过连接和剪枝生成候选k项集，然后通过扫描数据库计算支持度并删除不满足最小支持度阈值的项集，得到频繁k项集。这个过程一直持续到不能再找到更大的频繁项集为止。然后，根据频繁项集和最小置信度阈值生成强关联规则。

4．附录

# %%

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# %%

# Apriori 算法

# 1. 支持度计算

def loadDataSet():

# 读取数据txt中的数据

data = pd.read\_csv('Transactions.csv')

return data

# %%

def createC1(dataSet):# 创建候选项集

C1 = []

for transaction in dataSet:

for item in transaction:

if not [item] in C1:

C1.append([item])

C1.sort()

return list(map(frozenset, C1))# 使用frozenset是为了后面可以将这些值作为字典的键

def scanD(D, Ck, minSupport):# 从候选项集中筛选出满足最小支持度的项集

# D: 数据集 Ck: 候选项集 minSupport: 最小支持度

ssCnt = {}

for tid in D: # 遍历数据集中的每一条交易记录

for can in Ck:

if can.issubset(tid):

if not can in ssCnt: # 如果不在字典中，就添加进去

ssCnt[can] = 1

else: # 如果在字典中，就加1

ssCnt[can] += 1

numItems = float(len(D))

retList = []

supportData = {}

for key in ssCnt:

support = ssCnt[key] / numItems

if support >= minSupport:

retList.insert(0, key)

supportData[key] = support

return retList, supportData # 返回满足最小支持度的项集和对应的支持度

# %%

# 2. 生成候选项集

def aprioriGen(Lk, k):# 生成候选项集

retList = []

lenLk = len(Lk)

for i in range(lenLk):

for j in range(i+1, lenLk):

L1 = list(Lk[i])[:k-2]; L2 = list(Lk[j])[:k-2]

L1.sort(); L2.sort()

if L1 == L2:

retList.append(Lk[i] | Lk[j])

return retList

def apriori(dataSet, minSupport = 0.0): # 生成所有满足最小支持度的项集

# dataSet: 数据集 minSupport: 最小支持度

C1 = createC1(dataSet)

# D = list(map(set, dataSet))

D = []

# 遍历每一行

for index, row in dataSet.iterrows():

# 创建一个空集合来存储这一行的列名

s = set()

# 遍历除了views之外的所有列

for col in dataSet.columns[:-1]:

# 如果这个数据是1，就把这列的名称加入到集合中

if row[col] == 1:

s.add(col)

# 将s转为list

s = list(s)

# 把这一行对应的集合加入到列表中

D.append(s)

# print (D)

L1, supportData = scanD(D, C1, minSupport)

print (L1)

L = [L1]

k = 2 # k表示项集中元素的个数

while (len(L[k-2]) > 0):

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

Lk, supK = scanD(D, Ck, minSupport)

supportData.update(supK)

L.append(Lk)

k += 1

return L, supportData # 返回所有满足最小支持度的项集和对应的支持度

# %%

# 2. 生成候选项集

def aprioriGen(Lk, k):# 生成候选项集

retList = []

lenLk = len(Lk)

for i in range(lenLk):

for j in range(i+1, lenLk):

L1 = list(Lk[i])[:k-2]; L2 = list(Lk[j])[:k-2]

L1.sort(); L2.sort()

if L1 == L2:

retList.append(Lk[i] | Lk[j])

return retList

def apriori(dataSet, minSupport = 0.05): # 生成所有满足最小支持度的项集

# dataSet: 数据集 minSupport: 最小支持度

C1 = createC1(dataSet)

# D = list(map(set, dataSet))

D = []

# 遍历每一行

for index, row in dataSet.iterrows():

# 创建一个空集合来存储这一行的列名

s = set()

# 遍历除了views之外的所有列

for col in dataSet.columns[:-1]:

# 如果这个数据是1，就把这列的名称加入到集合中

if row[col] == 1:

s.add(col)

# 将s转为list

s = list(s)

# 把这一行对应的集合加入到列表中

D.append(s)

print (D)

L1, supportData = scanD(D, C1, minSupport)

print (L1)

L = [L1]

k = 2 # k表示项集中元素的个数

while (len(L[k-2]) > 0):

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

Lk, supK = scanD(D, Ck, minSupport)

supportData.update(supK)

L.append(Lk)

k += 1

return L, supportData # 返回所有满足最小支持度的项集和对应的支持度

# %%

# 3. 生成关联规则

def generateRules(L, supportData, minConf=0.1):# 生成关联规则

bigRuleList = []

for i in range(1, len(L)):

for freqSet in L[i]: # 遍历频繁项集中的每一个项集

print(freqSet)

H1 = [frozenset([item]) for item in freqSet]

if (i > 1): # 如果频繁项集中的元素超过2个，就需要进一步合并

rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf)

else:

calcConf(freqSet, H1, supportData, bigRuleList, minConf)

return bigRuleList

def calcConf(freqSet, H, supportData, brl, minConf=0.7):# 计算置信度

# freqSet:频繁项集 H:频繁项集中所有的元素的集合 supportData:支持度数据 brl:关联规则列表 minConf:最小置信度

prunedH = []

for conseq in H:

conf = supportData[freqSet] / supportData[freqSet - conseq] # 计算置信度

if conf >= minConf:

print(freqSet-conseq, '-->', conseq, 'conf:', conf)

brl.append((freqSet-conseq, conseq, conf))

prunedH.append(conseq)

return prunedH

def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):# 生成候选规则集:

# freqSet:频繁项集 H:频繁项集中所有的元素的集合 supportData:支持度数据 brl:关联规则列表 minConf:最小置信度

m = len(H[0])

if (len(freqSet) > (m + 1)):

Hmp1 = aprioriGen(H, m+1)

Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)

if (len(Hmp1) > 1):

rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)

# %%

# test

dataSet = loadDataSet()

dataSet.head()

# %%

dataSet.drop(['COD'], axis=1, inplace=True)

dataSet.head()

# %%

# 将数据集转换为集合,如果对应项为0，则将这一列的名称不添加到集合中，如果为不为0，则添加到集合中

# 假设df是Pandas读入的数据

# 创建一个空列表来存储每一行对应的集合

D = []

# 遍历每一行

for index, row in dataSet.iterrows():

# 创建一个空集合来存储这一行的列名

s = set()

# 遍历除了views之外的所有列

for col in dataSet.columns[:-1]:

# 如果这个数据是1，就把这列的名称加入到集合中

if row[col] == 1:

s.add(col)

# 将s转为list

s = list(s)

# 把这一行对应的集合加入到列表中

D.append(s)

# %%

D

# %%

C1 = createC1(D) # 创建候选项集

C1

# %%

L1, supportData0 = scanD(D, C1, 0.0) # 从候选项集中筛选出满足最小支持度的项集

print(L1) # 打印出满足最小支持度的项集

print(supportData0) # 打印出每个项集的支持度

# %%

L, supportData = apriori(D) # 生成所有满足最小支持度的项集

print(L) # 打印出所有满足最小支持度的项集

print(supportData) # 打印出每个项集的支持度

# %%

rules = generateRules(L, supportData, minConf=0.7) # 生成关联规则

print(rules) # 打印出关联规则

# %%

# test

dataSet = loadDataSet()

dataSet.head()

# %%

dataSet.drop(['COD'], axis=1, inplace=True)

dataSet.head()

# %% [markdown]

# ### 在下面是成功的调试

# %%

# 将数据集转换为集合,如果对应项为0，则将这一列的名称不添加到集合中，如果为不为0，则添加到集合中

# 假设df是Pandas读入的数据

# 创建一个空列表来存储每一行对应的集合

D = []

# 遍历每一行

for index, row in dataSet.iterrows():

# 创建一个空集合来存储这一行的列名

s = []

# 遍历除了views之外的所有列

for col in dataSet.columns[:-1]:

# 如果这个数据是1，就把这列的序号加入到集合中

if row[col] == 1:

# 查看现在的是第几列

col = dataSet.columns.get\_loc(col)

s.append(col)

# 把这一行对应的集合加入到列表中

D.append(s)

D

# %%

def apriori\_test(dataSet, minSupport = 0.2):

C1 = createC1(dataSet)

D = list(map(set, dataSet))

L1, supportData = scanD(D, C1, minSupport)

L = [L1]

k = 2

while (len(L[k-2]) > 0):

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

Lk, supK = scanD(D, Ck, minSupport)

supportData.update(supK)

L.append(Lk)

k += 1

return L, supportData

# %%

print(type(D))

print(type(D[0]))

# %%

dataSet.head()

# %%

L, supportData = apriori\_test(D,0.07) # 生成所有满足最小支持度的项集

print(L) # 打印出所有满足最小支持度的项集

print(supportData) # 打印出每个项集的支持度

# %%

rules = generateRules(L, supportData, minConf=0.7) # 生成关联规则

print(rules) # 打印出关联规则

# %% [markdown]

# ### 上面是成功的

# %%

C1 = createC1(D) # 创建候选项集

C1

# %%

L1, supportData0 = scanD(D, C1, 0.2) # 从候选项集中筛选出满足最小支持度的项集

print(L1) # 打印出满足最小支持度的项集

# %%

print(supportData0) # 打印出每个项集的支持度

# %%

L = [L1]

print(L)

print(len(L))

k = 2

while (len(L[k-2]) > 0): # 在L[k-2]中的项集的元素个数大于0时，继续循环

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

Lk, supK = scanD(D, Ck, 0.05)

supportData.update(supK)

L.append(Lk)

k += 1 # k表示项集中元素的个数

# 去掉L中长度为1的

L = [L[i] for i in range(len(L)) if len(L[i]) > 1]

print(L) # 打印出所有满足最小支持度的项集

print (len(L)) # 打印出每个项集的支持度

# %%

print(supportData) # 打印出每个项集的支持度

print(len(supportData)) # 打印出项集的个数

# %%

# 获得规则关联集合

rules = generateRules(L, supportData, minConf=0.1) # 生成关联规则

print(rules) # 打印出关联规则

# %%

# test3

dataSet = loadDataSet()

dataSet.head()

dataSet.drop(['COD'], axis=1, inplace=True)

dataSet.head()

# %%

L, supportData = apriori(dataSet) # 生成所有满足最小支持度的项集

print(L) # 打印出所有满足最小支持度的项集

print(supportData) # 打印出每个项集的支持度