**实验七 Apriori算法**

Apriori算法常用于关联分析。关联分析又称关联规则学习，是指从大规模数据中寻找物品间的隐含关系，最有名的例子是“尿布与啤酒”。关联分析中最常见的四个概念如下：

**频繁项集(frequent item sets)**：经常一块出现的物品的集合。

**关联规则(association rules)**：暗示两种物品之间可能存在的很强的关系。

**支持度(support)**：某个项集的支持度指数据集中包含该项集的记录所占的比例。

**置信度(confidence)**：指某条关联规则的可信度，比如对于“尿布->啤酒”这条关联规则，其置信度为支持度({尿布，啤酒})/支持度({尿布})。

Apriori这个词来源于拉丁语，指的是“来自以前”，因此Apriori算法指的是利用先验知识对未知的知识进行判断。Apriori原理是说如果某个项集是频繁的，那么它的所有子集也是频繁的。反过来，如果一个项集是非频繁项集，那么它的所有超集也是非频繁的。使用Apriori原理可以避免项集数目的指数增长，从而在合理时间内计算出频繁项集。

关联分析的目标包括两项：发现频繁项集和发现关联规则。

Apriori算法是发现频繁项集的一种方法。该算法首先生成所有单个物品的项集列表，接着扫描交易记录来查看哪些项集满足最小支持度要求，这一步将去除那些不满足最小支持度的集合；然后，对剩下的集合进行两两组合，再重新扫描交易记录，去掉不满足最小支持度的项集。该过程重复进行直到所有项集都被去掉。

我们使用k-候选集表示包含k个物品的项集，用k-频繁项集表示其对应的频繁项集。

**实验要求：**

1、实现Apriori算法，用以发现频繁项集；

2、从频繁项集中挖掘关联规则。

**实验步骤：**

新建apriori.py文件。

1、实现Apriori算法

①创建一些辅助函数，包括读数据、生成单个物品项集。

# 读数据，这里使用一个简单数据集

def loadDataSet():

return [[1,3,4],[2,3,5],[1,2,3,5],[2,5]]

# 生成单个物品项集

def createC1(dataSet):

C1 = []

# 循环查看每一条交易记录

# 对每一条交易记录，判断记录中的物品是否已经在C1中

# 如果不在则添加

# 使用C1.append([XXX])，注意要使用中括号

#【代码待补全】

# 返回创建好的项集

# 这里使用了frozenset构建不可变集合

return list(map(frozenset,C1))

②编写数据集扫描函数。其伪代码如下：

对数据集中的每条交易记录tran：

对每个候选项集can：

检查一下can是否是tran的子集：

如果是则增加can的计数值

对每个候选项集：

如果其支持度不低于最小支持度，则保留该项集

返回频繁项集列表

根据伪代码和提示，补全代码。

def scanD(D, Ck, minSupport):

# 创建一个字典保存各个候选集的计数

ssCnt = {}

for tid in D:

for can in Ck:

# 判断can是否为tid子集，可使用issubset()函数

# 如果是则增加can的计数，保存在ssCnt中

# 更新计数时要先判断can是否已经在ssCnt的keys中

#【代码待补全】

# 获取总的交易数

numItems = float(len(D))

# 创建返回列表保存频繁项集

retList = []

# 创建支持度字典保存各个项集的支持度，以便后续计算置信度

supportData = {}

for key in ssCnt:

# 计算项集的支持度，使用该项集的计数除以总的交易数即可

# 如果支持度大于最小支持度，则将项集即key保存到retList中

# 同时将支持度的值保存到supportData中

#【代码待补全】

# 返回频繁项集列表和支持度字典

return retList,supportData

③编写函数，构建一个由k个项组成的候选集列表。这里的Lk是已知的每一项包

含k-1个物品的频繁项集列表。

def createCk(Lk, k):

retList = []

lenLk = len(Lk)

for i in range(lenLk):

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

# 获取两个集合的前k-2个项

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

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

L1.sort()

L2.sort()

if L1 == L2:

# 如果前k-2个项相同，则将两个集合合并

# 将合并结果添加到返回列表中

# 添加使用append函数，合并集合使用|

#【代码待补全】

return retList

④编写apriori函数，发现所有频繁项集。整个Apriori算法的伪代码如下：

当集合中项的个数大于0时：

构建一个k个项组成的候选项集的列表

检查数据获取频繁项集

保留频繁项集及其支持度计数并构建k+1项组成的候选项集的列表

# apriori函数接收两个参数，第一个为数据集，第二个为最小支持度，为可选

# 参数，默认使用0.5

def apriori(dataSet, minSupport=0.5):

# 生成单个物品项集

C1 = createC1(dataSet)

D = list(map(set,dataSet))

# 根据单个物品项集获取1-频繁项集及其支持度

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

# 使用L保存所有的频繁项集

L = [L1]

k = 2 # k为项集中每一项物品数

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

#【思考】为什么是k-2

# 生成k-候选集

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

# 调用scanD函数获取k-频繁项集及其支持度

# 保存到变量Lk和supK中

#【代码待补全】

# 更新支持度字典supportData、频繁项集列表L以及k

# 支持度字典使用update()函数，频繁项集使用append()函数

#【代码待补全】

return L, supportData

2、挖掘关联规则

一条规则A->B的置信度为support(A,B)/support(A)。

关联规则有一个性质：如果某条规则并不满足最小置信度要求，那么该规则的所有子集也不会满足最小置信度要求。如，假设规则0,1,2->3不满足最小置信度要求，那么任何左部为{0,1,2}子集的规则也不会满足最小置信度要求。

这一性质可以减少需要测试的规则数目。可以首从一个频繁项集开始，接着创建一个规则列表，其中规则右部只包含一个元素，然后计算其置信度，对其进行修剪，接着合并剩下的规则来创建一个新的规则列表，其中规则右部包含两个元素。以此类推。这种方法叫分级法。

可以将这一过程拆分成三个部分：一个主函数，负责调用其余部分的函数生成规则；一个函数负责计算规则的置信度并对规则列表进行修剪；一个函数负责合并规则。

①编写主函数。参数中，L表示前一步得到的所有频繁项集，supportData为其对应的支持度计数，minConf表示最小置信度,若用户没提供则为0.7。

def generateRules(L, supportData, minConf=0.7):

bigRuleList = []

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

#【思考】这里为什么是从1开始

# 注意这里的i是(i+1)-频繁项集的下标

for freqSet in L[i]:

# 构造只包含单个元素集合的列表

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

# 如果频繁项集中的元素数目超过2(通过i判断即可)

# 则调用规则合并函数

# 否则调用置信度计算函数

# 函数签名在接下来的第②和第③步给出

# 提示：这里均不需要使用到返回值

#【代码待补全】

return bigRuleList

②编写置信度计算及修剪函数。参数中的brl对应主函数的bigRuleList。

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

# 创建修剪后的规则列表H

prunedH = []

for conseq in H:

# 计算以conseq作为右件的规则的置信度conf

# 使用supportData中的数据即可

# 提示：conseq作为右件时，左件为freqSet-conseq

# 左右件同时出现的支持度刚好等于freqSet的支持度

#【代码待补全】

# 如果置信度满足最小置信度要求，则打印规则

# 并将规则添加到brl中，同时将右件添加到修剪后的规则列表中

if conf >= minConf:

# 打印规则

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

# 添加规则

# 使用append函数，添加内容为(左件,右件,置信度)

# 括号也要

#【代码待补全】

# 添加右件到修剪后的规则列表中，用append函数

#【代码待补全】

return prunedH

③合并规则函数。函数没有返回值。

def mergeRules(freqSet, H, supportData, brl, minConf):

# 获取目前规则列表中规则的右件的元素个数

m = len(H[0])

# 确保该频繁项集大到可以移除大小为m的子集

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

# 调用createCk函数来生成H中元素的无重复组合

Hmp1 = createCk(H, m + 1)

# 调用calcConf函数修剪规则列表

# 返回值保存到Hmp1

#【代码待补全】

# 如果修剪后的规则列表中包含多条规则，则继续尝试合并

if len(Hmp1) > 1:

mergeRules(freqSet, Hmp1, supportData, brl,\

minConf)

添加一个测试函数。

def test(minSupport=0.5, minConf=0.7):

data = loadDataSet()

L, supportData = apriori(data, minSupport)

rules = generateRules(L, supportData, minConf)

return rules

之后直接在python命令行使用以下命令即可查看结果：

import apriori

rules = apriori.test()

大家可以试着改一下最小支持度和最小置信度，看看生成的规则有什么不同。

直接在test函数中添加这两个参数即可。