源程序：

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

from Tkinter import \* # 导入 Tkinter 库

import os

import tkFileDialog

import sys

name=r""

root = Tk() # 创建窗口对象的背景色

root.title("数据挖掘算法实现")

root.geometry('500x600') #是x 不是\*

root.resizable(width=True, height=True) #宽不可变, 高可变,默认为True

view = Frame(root)

var = StringVar()

e = Entry(view,textvariable = var)

var.set("hello")

e.pack()

def seeview(a):

#此函数为输出窗口

t.insert('0.0',a)

t = Text()

t.pack()

def chooseFile():

#此函数为选择文件

global name

default\_dir = r"C:\Users" # 设置默认打开目录

filename = tkFileDialog.askopenfilename(title=u"选择文件")

if filename != '':

lb.config(text = u"您选择的文件是："+filename);

else:

lb.config(text = u"您没有选择任何文件");

name = filename

filea = open(filename)

lines = filea.readlines()

for line in lines:

seeview(line)

seeview( "\n")

def dealwith():

"""

此函数为aprioria算法的主程序

"""

data\_set = loadDataSet()#载入数据

# seeview(data\_set)

if data\_set==[]:

seeview( u"请选择数据集")

return

L, support\_data = generate\_L(data\_set, k=2, min\_support=0.2)#生成所有的频繁项集。

# seeview(L)

big\_rules\_list = generate\_big\_rules(L, support\_data, min\_conf=0.7)#从大的频繁项集产生的规则集

for Lk in L:#输出生成的频繁项集以及各自的支持度

seeview( "~"\*50),

seeview( "\n")

seeview(str(len(list(Lk)[0])) + "-item\tsupport" + "\t\t频繁项集支持度% ")

seeview( "\n")

for freq\_set in Lk:

seeview(str(support\_data[freq\_set])),

seeview(freq\_set),

seeview( "\n")

seeview( "\n")

seeview(u"计算结果")

seeview( "\n")

for item in big\_rules\_list:#输出找到的强关联规则

seeview(str(item[2])+"\t"),

seeview(str(item[1])+ u"支持度: "+"\t"),

seeview(str(item[0])+ "-------->"+"\t"),

seeview( "\n")

def loadDataSet(): #读取数据（这里只有两个特征）

dataMat = []

labelMat = []

if name=='':

seeview( u"请选择数据集")

return

fr = open(name)

for line in fr.readlines():

lineArr = line.strip().split()

try:

dataMat.append([str(lineArr[0]), str(lineArr[1]),str(lineArr[2])])

except:

dataMat.append([str(lineArr[0]), str(lineArr[1])])

return dataMat

def create\_C1(data\_set):

"""

通过扫描数据集，创建频繁候选对象1-itemset C1

输入:

data\_set: 事务列表。每个事务包含若干项。

输出:

C1:包含所有频繁候选项集的集合

"""

C1 = set()

for t in data\_set:

for item in t:

item\_set = frozenset([item])

C1.add(item\_set)

return C1

"""

判断是否有候选频繁k项集满足Apriori性质。

"""

def is\_apriori(Ck\_item, Lksub1):

"""

输入:

Ck\_item: a frequent candidate k-itemset in Ck which contains all frequent

candidate k-itemsets.

Lksub1: Lk-1, a set which contains all frequent candidate (k-1)-itemsets.

"""

for item in Ck\_item:

sub\_Ck = Ck\_item - frozenset([item])

if sub\_Ck not in Lksub1:

return False

return True

"""

输出:

True: satisfying Apriori property.

False: Not satisfying Apriori property.

"""

"""

创建Ck, 包含所有频繁候选k项集的集合

通过LK-1自己的连接操作。

"""

def create\_Ck(Lksub1, k):

"""

输入:

Lksub1:LK-1，包含所有频繁候选（K-1）项集的集合。

k:频繁项集的项的数量。

"""

Ck = set()

len\_Lksub1 = len(Lksub1)

list\_Lksub1 = list(Lksub1)

for i in range(len\_Lksub1):

for j in range(1, len\_Lksub1):

l1 = list(list\_Lksub1[i])

l2 = list(list\_Lksub1[j])

l1.sort()

l2.sort()

if l1[0:k-2] == l2[0:k-2]: # 修剪

Ck\_item = list\_Lksub1[i] | list\_Lksub1[j]

if is\_apriori(Ck\_item, Lksub1):

Ck.add(Ck\_item)

"""

输出:

Ck:包含所有频繁候选k项集的集合。

"""

return Ck

"""

通过从CK执行删除策略生成LK。

"""

def generate\_Lk\_by\_Ck(data\_set, Ck, min\_support, support\_data):

"""

输入:

data\_set:

Ck:事务列表。每个事务包含若干项。

min\_support:最小支持度。

support\_data:一个字典，关键是频繁项集，值是支持度。

"""

Lk = set()

item\_count = {}

for t in data\_set:

for item in Ck:

if item.issubset(t):

if item not in item\_count:

item\_count[item] = 1

else:

item\_count[item] += 1

t\_num = float(len(data\_set))

for item in item\_count:

if (item\_count[item] / t\_num) >= min\_support:

Lk.add(item)

support\_data[item] = item\_count[item] / t\_num

"""

输出:

Lk:它包含了所有的频繁项集的所有k-项集。

"""

return Lk

"""

生成所有的频繁项集。

"""

def generate\_L(data\_set, k, min\_support):

"""

输入:

data\_set:事务列表。每个事务包含若干项。

k:所有频繁项的最大相数目

min\_support:最小支持度。

"""

support\_data = {}

C1 = create\_C1(data\_set)

L1 = generate\_Lk\_by\_Ck(data\_set, C1, min\_support, support\_data)

Lksub1 = L1.copy()

L = []

L.append(Lksub1)

for i in range(2, k+1):

Ci = create\_Ck(Lksub1, i)

Li = generate\_Lk\_by\_Ck(data\_set, Ci, min\_support, support\_data)

Lksub1 = Li.copy()

L.append(Lksub1)

"""

输出:

L: LK的列表。

support\_data:一个字典，关键是频繁项集，值是支持度。

"""

return L, support\_data

"""

从大的频繁项集产生的规则集。

"""

def generate\_big\_rules(L, support\_data, min\_conf):

"""

输入:

L: LK的列表。

support\_data:一个字典，关键是频繁项集，值是支持度。

min\_conf: 最小置信度

"""

big\_rule\_list = []

sub\_set\_list = []

for i in range(0, len(L)):

for freq\_set in L[i]:

for sub\_set in sub\_set\_list:

if sub\_set.issubset(freq\_set):

conf = support\_data[freq\_set] / support\_data[freq\_set - sub\_set]

big\_rule = (freq\_set - sub\_set, sub\_set, conf)

if conf >= min\_conf and big\_rule not in big\_rule\_list:

# print freq\_set-sub\_set, " => ", sub\_set, "conf: ", conf

big\_rule\_list.append(big\_rule)

sub\_set\_list.append(freq\_set)

"""

输出:

big\_rule\_list: 一个包含所有大规则的列表。每个大的规则表示为2元组。

"""

return big\_rule\_list

lb = Label(root,text = '')

lb.pack()

btn = Button(root,text=u"选择文件",command=chooseFile)

btn.pack()

Button(root, text="处理", command = dealwith).pack()

root.mainloop() # 进入消息循环

程序说明：

程序的可执行文件在dist文件夹下的apshi.exe文件，双击直接运行。

data.txt文件为样例数据集。在选择数据时选择data.txt文件及可。

程序的源代码为dist文件夹下的apshi.py文件。

论文原英文版为此文件夹下：08090317.pdf文件。

论文翻译为此文件夹下：论文翻译.docx文件。

# ****1.********Apriori算法简介****

Apriori算法是经典的挖掘频繁项集和关联规则的数据挖掘算法。A priori在拉丁语中指"来自以前"。当定义问题时，通常会使用先验知识或者假设，这被称作"一个先验"（a priori）。Apriori算法的名字正是基于这样的事实：算法使用频繁项集性质的先验性质，即频繁项集的所有非空子集也一定是频繁的。Apriori算法使用一种称为逐层搜索的迭代方法，其中k项集用于探索(k+1)项集。首先，通过扫描数据库，累计每个项的计数，并收集满足最小支持度的项，找出频繁1项集的集合。该集合记为L1。然后，使用L1找出频繁2项集的集合L2，使用L2找出L3，如此下去，直到不能再找到频繁k项集。每找出一个Lk需要一次数据库的完整扫描。Apriori算法使用频繁项集的先验性质来压缩搜索空间。

# ****2. 基本概念****

****项与项集****：设items是所有项的集合，其中，item\_k(k=1,2,…,m)成为项。项的集合称为项集（items），包含k个项的项集称为k项集(k-items)。

* ****事务与事务集****：一个事务T是一个项集，它是items的一个子集，每个事务均与一个唯一标识符Tid相联系。不同的事务一起组成了事务集D，它构成了关联规则发

****关联规则****：关联规则是形如A=>B的蕴涵式，其中A、B均为itemset的子集且均不为空集，而A交B为空。

****支持度(support)****：关联规则的支持度定义如下：

IMG_256

其中IMG_257表示事务包含集合A和B的并（即包含A和B中的每个项）的概率。注意与P(A or B)区别，后者表示事务包含A或B的概率。

****置信度(confidence)****：关联规则的置信度定义如下：

IMG_258

****项集的出现频度(support count)****：包含项集的事务数，简称为项集的频度、支持度计数或计数。

****频繁项集(frequent itemset)****：如果项集I的相对支持度满足事先定义好的最小支持度阈值（即I的出现频度大于相应的最小出现频度（支持度计数）阈值），则I是频繁项集。

****强关联规则****：满足最小支持度和最小置信度的关联规则，即待挖掘的关联规则。

# ****3. 实现步骤****

关联规则的挖掘过程：

1找出所有的频繁项集

2由频繁项集产生强关联规则

## ****3.1挖掘频繁项集****

### ****3.1.1********相关定义****

****连接步骤：频繁(k-1)项集Lk-1的自身连接产生候选k项集Ck****

    Apriori算法假定项集中的项按照字典序排序。如果Lk-1中某两个的元素（项集）items1和items2的前(k-2)个项是相同的，则称itemset1和itemset2是可连接的。所以items1与items2连接产生的结果项集是{itemset1[1], items1[2], …, items1[k-1], items[k-1]}。

****剪枝策略****

由于存在先验性质：****任何非频繁的(k-1)项集都不是频繁k项集的子集****。因此，如果一个候选k项集Ck的(k-1)项子集不在Lk-1中，则该候选也不可能是频繁的，从而可以从Ck中删除，获得压缩后的Ck。

****删除策略****

基于压缩后的Ck，扫描所有事务，对Ck中的每个项进行计数，然后删除不满足最小支持度的项，从而获得频繁k项集。

### ****3.1.2 步骤****

每个项都是候选1项集的集合C1的成员。算法扫描所有的事务，获得每个项，生成C1（见下文代码中的create\_C1函数）。然后对每个项进行计数。然后根据最小支持度从C1中删除不满足的项，从而获得频繁1项集L1。

对L1的自身连接生成的集合执行剪枝策略产生候选2项集的集合C2，然后，扫描所有事务，对C2中每个项进行计数。同样的，根据最小支持度从C2中删除不满足的项，从而获得频繁2项集L2。

对L2的自身连接生成的集合执行剪枝策略产生候选3项集的集合C3，然后，扫描所有事务，对C3每个项进行计数。同样的，根据最小支持度从C3中删除不满足的项，从而获得频繁3项集L3。

以此类推，对Lk-1的自身连接生成的集合执行剪枝策略产生候选k项集Ck，然后，扫描所有事务，对Ck中的每个项进行计数。然后根据最小支持度从Ck中删除不满足的项，从而获得频繁k项集。

## ****3.2 由频繁项集产生关联规则****

一旦找出了频繁项集，就可以直接由它们产生强关联规则。产生步骤如下：

对于每个频繁项集itemset，产生itemset的所有非空子集（这些非空子集一定是频繁项集）；

对于itemset的每个非空子集s,如果IMG_259，则输出IMG_260，其中min\_conf是最小置信度阈值。