**PythonApriori算法提取客户购买商品的关联规则**

**相关知识**

关联规则：

A->B的支持度support=P(AB)，指的是事件A和事件B同时发生的概率。

置信度confidence=P(B|A)=P(AB)/P(A)，指的是发生事件A的基础上发生事件B的概率。

同时满足最小支持度阈值和最小置信度阈值的规则称为强规则。

如果事件A中包含k个元素，那么称这个事件A为k项集，并且事件A满足最小支持度阈值的事件称为频繁k项集。

2）挖掘过程：

第一，找出所有的频繁项集；

第二，由频繁项集产生强规则。

Apriori算法使用频繁项集的先验知识，使用一种称作逐层搜索的迭代方法，k项集用于探索(k+1)项集。首先，通过扫描事务（交易）记录，找出所有的频繁1项集，该集合记做L1，然后利用L1找频繁2项集的集合L2，L2找L3，如此下去，直到不能再找到任何频繁k项集。最后再在所有的频繁集中找出强规则，即产生用户感兴趣的关联规则。

其中，Apriori算法具有这样一条性质：任一频繁项集的所有非空子集也必须是频繁的。因为假如P(I)< 最小支持度阈值，当有元素A添加到I中时，结果项集（A∩I）不可能比I出现次数更多。因此A∩I也不是频繁的。

连接步和剪枝步

在上述的关联规则挖掘过程的两个步骤中，第一步往往是总体性能的瓶颈。Apriori算法采用连接步和剪枝步两种方式来找出所有的频繁项集。

1） 连接步

为找出Lk（所有的频繁k项集的集合），通过将Lk-1（所有的频繁k-1项集的集合）与自身连接产生候选k项集的集合。候选集合记作Ck。设l1和l2是Lk-1中的成员。记li[j]表示li中的第j项。假设Apriori算法对事务或项集中的项按字典次序排序，即对于（k-1）项集li，li[1]<li[2]<……….<li[k-1]。将Lk-1与自身连接，如果(l1[1]=l2[1])&&( l1[2]=l2[2])&&……..&& (l1[k-2]=l2[k-2])&&(l1[k-1]<="" p="" style="margin: 0px; padding: 0px; box-sizing: border-box;">

2） 剪枝步

CK是LK的超集，也就是说，CK的成员可能是也可能不是频繁的。通过扫描所有的事务（交易），确定CK中每个候选的计数，判断是否小于最小支持度计数，如果不是，则认为该候选是频繁的。为了压缩Ck,可以利用Apriori性质：任一频繁项集的所有非空子集也必须是频繁的，反之，如果某个候选的非空子集不是频繁的，那么该候选肯定不是频繁的，从而可以将其从CK中删除。

（Tip：为什么要压缩CK呢？因为实际情况下事务记录往往是保存在外存储上，比如数据库或者其他格式的文件上，在每次计算候选计数时都需要将候选与所有事务进行比对，众所周知，访问外存的效率往往都比较低，因此Apriori加入了所谓的剪枝步，事先对候选集进行过滤，以减少访问外存的次数。）

**系统环境**

Linux Ubuntu 16.04

Python3.6

**任务内容**

以超市销售数据为例，提取客户购买商品的关联规则。

**任务步骤**

1.首先在Linux上新建/data/python14目录，并切换到该目录下。

[view plain](https://www.ipieuvre.com/e/183/311/9539) [copy](https://www.ipieuvre.com/e/183/311/9539)

1. sudo mkdir -p /data/python14/

[view plain](https://www.ipieuvre.com/e/183/311/9539) [copy](https://www.ipieuvre.com/e/183/311/9539)

1. cd /data/python14/

修改/data/python14目录下的所有文件及子目录的所有者和所属的组为zhangyu

[view plain](https://www.ipieuvre.com/e/183/311/9539) [copy](https://www.ipieuvre.com/e/183/311/9539)

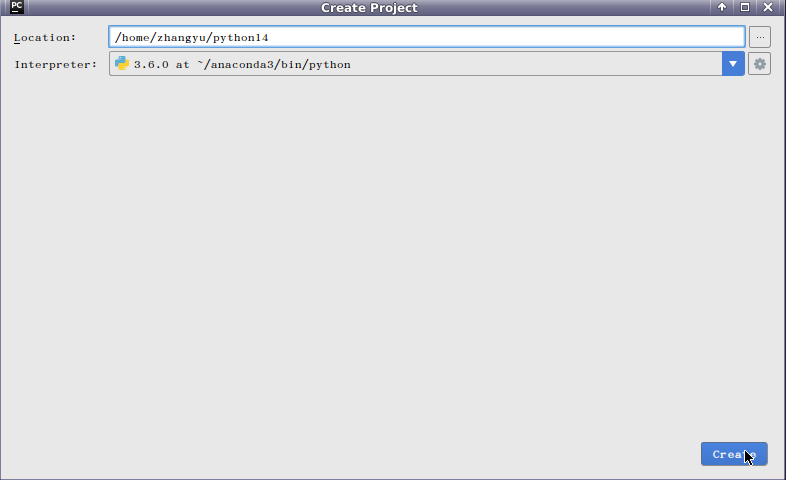
1. sudo chown -R zhangyu.zhangyu /data/python14

2.使用wge命令，从网址 http://192.168.1.100:60000/allfiles/python14/目录下，将实验所需数据下载到linux本地/data/python14目录下。

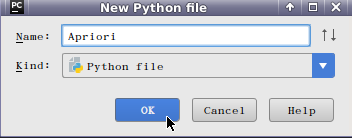
[view plain](https://www.ipieuvre.com/e/183/311/9539) [copy](https://www.ipieuvre.com/e/183/311/9539)

1. sudo wget http://192.168.1.100:60000/allfiles/python14/menu\_orders.xls

3.新建Python项目，名为python14

[](https://www.ipieuvre.com/doc/exper/288d8b2c-91ad-11e9-beeb-00215ec892f4/img/04.png)

在python14项目下，新建Python file，名为：Apriori

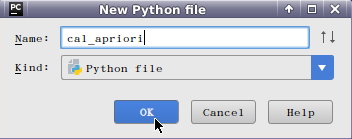
[](https://www.ipieuvre.com/doc/exper/288d8b2c-91ad-11e9-beeb-00215ec892f4/img/01.png)

4.自行编写Apriori算法

[view plain](https://www.ipieuvre.com/e/183/311/9539) [copy](https://www.ipieuvre.com/e/183/311/9539)

1. # -\*- coding: utf-8 -\*-
2. from \_\_future\_\_ **import** print\_function
3. **import** pandas as pd
4. # 自定义连接函数，用于实现L\_{k-1}到C\_k的连接
5. def connect\_string(x, ms):
6. x = list(map(lambda i: sorted(i.split(ms)), x))
7. l = len(x[0])
8. r = []
9. **for** i **in** range(len(x)):
10. **for** j **in** range(i, len(x)):
11. **if** x[i][:l - 1] == x[j][:l - 1] and x[i][l - 1] != x[j][l - 1]:
12. r.append(x[i][:l - 1] + sorted([x[j][l - 1], x[i][l - 1]]))
13. **return** r
14. # 寻找关联规则的函数
15. def find\_rule(d, support, confidence, ms=u'--'):
16. result = pd.DataFrame(index=['support', 'confidence'])  # 定义输出结果
17. support\_series = 1.0 \* d.sum() / len(d)  # 支持度序列
18. column = list(support\_series[support\_series > support].index)  # 初步根据支持度筛选
19. k = 0
20. **while** len(column) > 1:
21. k = k + 1
22. print(u'\n正在进行第%s次搜索...' % k)
23. column = connect\_string(column, ms)
24. print(u'数目：%s...' % len(column))
25. sf = lambda i: d[i].prod(axis=1, numeric\_only=True)  # 新一批支持度的计算函数
26. # 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。
27. d\_2 = pd.DataFrame(list(map(sf, column)), index=[ms.join(i) **for** i **in** column]).T
28. 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
29. column = list(support\_series\_2[support\_series\_2 > support].index)  # 新一轮支持度筛选
30. support\_series = support\_series.append(support\_series\_2)
31. column2 = []
32. **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？
33. i = i.split(ms)
34. **for** j **in** range(len(i)):
35. column2.append(i[:j] + i[j + 1:] + i[j:j + 1])
36. cofidence\_series = pd.Series(index=[ms.join(i) **for** i **in** column2])  # 定义置信度序列
37. **for** i **in** column2:  # 计算置信度序列
38. cofidence\_series[ms.join(i)] = support\_series[ms.join(sorted(i))] / support\_series[ms.join(i[:len(i) - 1])]
39. **for** i **in** cofidence\_series[cofidence\_series > confidence].index:  # 置信度筛选
40. result[i] = 0.0
41. result[i]['confidence'] = cofidence\_series[i]
42. result[i]['support'] = support\_series[ms.join(sorted(i.split(ms)))]
43. result = result.T.sort\_values(['confidence', 'support'], ascending=False)  # 结果整理，输出
44. print(u'\n结果为：')
45. print(result)
46. **return** result

5.新建python file文件，名为：cal\_apriori

[](https://www.ipieuvre.com/doc/exper/288d8b2c-91ad-11e9-beeb-00215ec892f4/img/02.png)

6.导入数据

[view plain](https://www.ipieuvre.com/e/183/311/9539) [copy](https://www.ipieuvre.com/e/183/311/9539)

1. inputfile = '/data/python14/menu\_orders.xls'
2. outputfile = '/data/python14/apriori\_rules.xls' #结果文件
3. data = pd.read\_excel(inputfile, header = None)

7.数据预处理

[view plain](https://www.ipieuvre.com/e/183/311/9539) [copy](https://www.ipieuvre.com/e/183/311/9539)

1. ct = lambda x : pd.Series(1, index = x[pd.notnull(x)]) #转换0-1矩阵的过渡函数
2. b = map(ct, data.as\_matrix()) #用map方式执行
3. #print(list(b))
4. data = pd.DataFrame(list(b)).fillna(0) #实现矩阵转换，空值用0填充
5. print(u'\n转换完毕。')
6. del b #删除中间变量b，节省内存

8.建立模型

[view plain](https://www.ipieuvre.com/e/183/311/9539) [copy](https://www.ipieuvre.com/e/183/311/9539)

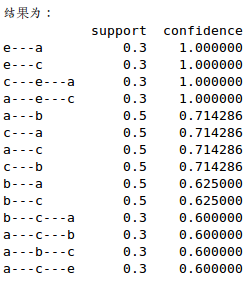
1. support = 0.2 #最小支持度
2. confidence = 0.5 #最小置信度
3. ms = '---' #连接符，默认'--'，用来区分不同元素，如A--B。需要保证原始表格中不含有该字符
4. find\_rule(data, support, confidence, ms).to\_excel(outputfile) #保存结果

9.完整代码

[view plain](https://www.ipieuvre.com/e/183/311/9539) [copy](https://www.ipieuvre.com/e/183/311/9539)

1. #-\*- coding: utf-8 -\*-
2. #使用Apriori算法挖掘菜品订单关联规则
3. from \_\_future\_\_ **import** print\_function
4. **import** pandas as pd
5. from Apriori **import** \* #导入自行编写的apriori函数
6. inputfile = '/data/python14/menu\_orders.xls'
7. outputfile = '/data/python14/apriori\_rules.xls' #结果文件
8. data = pd.read\_excel(inputfile, header = None)
9. print('\n转换原始数据至0-1矩阵...')
10. ct = lambda x : pd.Series(1, index = x[pd.notnull(x)]) #转换0-1矩阵的过渡函数
11. b = map(ct, data.as\_matrix()) #用map方式执行
12. #print(list(b))
13. data = pd.DataFrame(list(b)).fillna(0) #实现矩阵转换，空值用0填充
14. print(u'\n转换完毕。')
15. del b #删除中间变量b，节省内存
16. support = 0.2 #最小支持度
17. confidence = 0.5 #最小置信度
18. 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
19. find\_rule(data, support, confidence, ms).to\_excel(outputfile) #保存结果

10.运行结果



其中，e---a表示e发生能够推出a发生，置信度为100%，支持度为30%；b---c---a表示b、c同时发生时能够推出a发生，置信度为60%，支持度为30%等。搜索出来的关联规则不一定具有实际意义，需要根据问题背景筛选适当的有意义的规则，并赋予合理的解释。