关联分析之Apriori算法

算法简介

Apriori 算法是常用的用于挖掘关联规则的算法，它用于找出数据集中频繁出现的数据集合，找出这些数据结合有助于我们做出相应的决策。经典的用于购物车分析，找出强相关的商品集合，如啤酒和尿布。

准备知识

频繁项集的基础知识：

什么样的数据集合可以称之为频繁项集呢？ 绝不是肉眼观察的，而是靠标准来评估出来的。

频繁项集的评估标准：

支持度

支持度就是几个关联的数据在数据集中出现的次数占总数据集的权重，或者说是几个关联数据的出现的概率。

$$Support(X,Y) = P(XY) = \frac{number(XY)}{number(AllSamples)}$$

以上是两个事件的支持度，以此类推，我们可以给出三个事件的支持度：

$$Support(X,Y,Z) = P(XYZ) = \frac{number(XYZ)}{number(AllSamples)}$$

支持度与频繁项集的关系： 必要不充分，支持度高的不一定是频繁项集，支持度低的一定不是频繁项集，但是频繁项集一定是支持度高的

置信度

置信度是一个数据出现之后，另外一个数据出现的概率，它是一个条件概率

例如两个项目X，Y，Y对X的置信度如下：

$$Confidence(Y <= X) = P(Y|X) = P(XY)/P(X) = \frac{number(XY)}{number(X)}$$

类似也可以给出三个项目，X，Y，Z， Z对X和Y的置信度：

$$Confidence(Z <= (X,Y)) = P(Z|XY) = P(XYZ)/P(XY) = \frac{number(XYZ)}{number(XY)}$$

提升度

提升度表示含Y的条件下，同时含有X的概率，与X总体发生的概率之比：

$$Lift(X <= Y) = P(X|Y)/P(X) = Confidence(X <= Y)/P(X)$$

提升度的意义：

提升度体现了X，Y之间的关联关系，提升度大于1，则 X <= Y 是一条有效的关联规则，提升度小于等于1，那么就是一条弱的无用的关联规则，特别是当X和Y独立分布时，提升度就是1

通常人工定义支持度和置信度作为标准定义频繁项集：

比如， 购买杜蕾斯对绿箭口香糖的支持度是3%，置信度是50%的含义是：既买了杜蕾斯，又买了绿箭口香糖的用户占比为3%，买了杜蕾斯的用户中有50%的人买了绿箭口香糖。

什么样的数据集合可以称之为频繁项集呢?

如果项集I的相对支持度满足我们设置的最小支持度的阀值（即出现频度大于设定的最小频度或者最小支持度计数），那么他就是频繁项集。

什么样的规则是强关联规则?

满足最小支持度和最小置信度的关联规则。

实现原理

实现步骤：

- 找出所有的频繁项集
- 由频繁项集产生强关联规则

代码

```
import sys
from itertools import chain , combinations
from collections import defaultdict
from optparse import OptionParser
```

```
5
6   def subsets(arr):
7       return chain(*[combinations(arr,i+1) for i ,a in enumerate(arr)])
8
9   def runItemsWithMinSupport(itemSet , transactionList , minSupport ,
    freqSet):
10      """计算每个项集的频次，并按照最小支持度进行过滤"""
11      _itemSet = set()
12      localSet = defaultdict(int)
13      totalLength = len(transactionList)
14      # 计算每个商品集合的频数
15      for item in itemSet:
16          for transaction in transactionList :
17              if item.issubset(transaction)  :
18                  freqSet[item] += 1
19                  localSet[item] += 1
20      # 过滤掉那些支持度不达标的集合
21      for item,counts in localSet.items():
22          support = counts*1.0/totalLength
23          if support >= minSupport:
24              _itemSet.add(item)
25      return _itemSet
26
27
28  def joinSet(itemSet,length):
29      """根据已有的频繁项集获取新的频繁项集"""
30      return set([ i.union(j) for i in itemSet for j in itemSet if
    len(i.union(j)) == length ])
31
32  def getItemSetAndTransactionList(data):
33      """获取原始数据，然后获取所有的一元项集，
34      这里的项集并非频繁项集，而是所有可能的项集"""
35      transactionList = list()
36      itemSet = set()
37      for transaction in data :
38          transactionList.append(transaction)
39          transactionSet = frozenset(transaction)
40          for item in transactionSet:
41              itemSet.add(frozenset([item]))
42      return transactionList , itemSet
43
44  def runApriori(dataIterator,minSupport,minConfidence):
45      """计算频繁项集并获取关联规则"""
46      # 获取数据信息及一元项集
47      transactionList , itemSet =
    getItemSetAndTransactionList(dataIterator)
48      transactionLen = len(transactionList)
49      #定义一个存储全局项集的频次的字典
50      freqSet = defaultdict(int)
```

```
    #定义一个存储所有频繁项集的集合
    globalFreqSet = {}
    #定义一个存储全局关联规则的列表
    globalRules = list()
    #获取一元频繁项集
    oneItemSet = runItemsWithMinSupport(itemSet, transactionList,
minSupport,freqSet)
    initialSet = oneItemSet
    print(" 1 ----> " , initialSet)
    k = 2
    #计算所有的频繁项集
    while True:
        print( len(initialSet))
        if len(initialSet) > 0 :
            #添加频繁项集
            globalFreqSet[k] = initialSet
        else:
            break
        # 生成可能的频繁更多一个元素的项集
        probaFreqSet = joinSet(initialSet,k)
        print(probaFreqSet)
        confirmedFreqSet = runItemsWithMinSupport(probaFreqSet ,
transactionList , minSupport , freqSet)
        initialSet = confirmedFreqSet
        k += 1
    #获取关联规则，计算每条规则的置信度，并按找minConfidence进行过滤
    #定义一个计算事务概率的函数
    print(freqSet)
    def get_probability(seta):
        return freqSet[seta]*1.0/transactionLen

    globalFreqSetWithSupport = []
    for key, value in globalFreqSet.items():
        globalFreqSetWithSupport.extend([(tuple(item),
get_probability(item))
                            for item in value])

    #循环每个频繁项集
    for freqItems in globalFreqSet.values() :
        print("freqItems = > " , freqItems)
        for oneSet in freqItems:
            _subsets = map(frozenset, [x for x in subsets(oneSet)])
            #获取该频繁项集的每个子集
            for leftSet in _subsets:
                # 获取规则的右边
                rightSet = oneSet.difference(leftSet)
                if len(rightSet) == 0 or len(leftSet) == 0 :
                    continue
                else:
```

```
                    # 计算置信度
                    print("{s1} {s2} {f1} {f2}
".format(s1=oneSet,s2=leftSet,f1=freqSet[oneSet],f2=freqSet[leftSet]))
                    confidence =
get_probability(oneSet)/get_probability(leftSet)
                if confidence >= minConfidence:
                    globalRules.append(
(tuple(leftSet),tuple(rightSet),confidence) )
    return  globalFreqSetWithSupport,globalRules
```

测试一下代码

```
data = """
apple,beer,rice,chicken
apple,beer,rice
apple,beer
apple,mango
milk,beer,rice,chicken
milk,beer,rice
milk,beer
milk,mango
"""
# 定义一个读取事务集的函数
def dataFromString(s):
    s = s.strip()
    for line in s.split('\n'):
        record = frozenset(line.strip().split(','))
        yield record
minSupport = 0.2
minConfidence = 0.5
dataIterator = dataFromString(data)
dataIterator = dataFromString(data)
globalSet , globalRule =
 runApriori(dataIterator,minSupport,minConfidence)
```

输出结果：

```
In [81]: print(globalSet)
```

```
[(('mango',), 0.25), (('beer',), 0.75), (('rice',), 0.5), (('milk',), 0.5), (('apple',), 0.5), (('chicken',), 0.25),
(('beer', 'chicken'), 0.25), (('beer', 'milk'), 0.375), (('beer', 'apple'), 0.375), (('rice', 'milk'), 0.25), (('bee
r', 'rice'), 0.5), (('rice', 'apple'), 0.25), (('rice', 'chicken'), 0.25), (('beer', 'rice', 'apple'), 0.25), (('bee
r', 'rice', 'chicken'), 0.25), (('beer', 'rice', 'milk'), 0.25)]
```

```
In [82]: print(globalRule)
```

```
[(('chicken',), ('beer',), 1.0), (('beer',), ('milk',), 0.5), (('milk',), ('beer',), 0.75), (('beer',), ('apple',),
0.5), (('apple',), ('beer',), 0.75), (('rice',), ('milk',), 0.5), (('milk',), ('rice',), 0.5), (('beer',), ('rice',),
0.6666666666666666), (('rice',), ('beer',), 1.0), (('rice',), ('apple',), 0.5), (('apple',), ('rice',), 0.5), (('ric
e',), ('chicken',), 0.5), (('chicken',), ('rice',), 1.0), (('rice',), ('beer', 'apple'), 0.5), (('apple',), ('beer',
'rice'), 0.5), (('beer', 'rice'), ('apple',), 0.5), (('beer', 'apple'), ('rice',), 0.6666666666666666), (('rice', 'ap
ple'), ('beer',), 1.0), (('rice',), ('beer', 'chicken'), 0.5), (('chicken',), ('beer', 'rice'), 1.0), (('beer', 'ric
e'), ('chicken',), 0.5), (('beer', 'chicken'), ('rice',), 1.0), (('rice', 'chicken'), ('beer',), 1.0), (('rice',),
('beer', 'milk'), 0.5), (('milk',), ('beer', 'rice'), 0.5), (('beer', 'rice'), ('milk',), 0.5), (('beer', 'milk'),
('rice',), 0.6666666666666666), (('rice', 'milk'), ('beer',), 1.0)]
```

优点和缺点

优点

简单粗暴，能够找到频繁项集，并且得到需要的关联规则

缺点

不够高效，需要多次扫描数据集，所以后面有人专门写了fp-growth算法来优化效率的问题

用途拓展

挖掘频繁项集并非局限于商品，任何同时发生的事物，都可以拿来分析统计。

最经典的应用就是分析购物车了。

京东的分析师通过同一时间对同一商品同时下单的用户作为数据集，用关联规则来获取黑名单并取得了不错的效果，这个日后可以尝试。