Apriori发现频繁项集和关联规则

作者: 刘伟杰 日期:2015-12-13
参考: [1] 《机器学习实战》 Peter

1. 理论

1. 概述:

   Apriori算法可以用来发现频繁项集，进而在频繁项集的基础上发现关联规则。

2. 一些概念:

   频繁项集（frequent item sets): 物品的集合称为项集，经常出现的项集称为频繁项集，例如{啤酒，尿布，豆奶};

   支持度(support)：是针对一个项集来定义的，数据集中包含该项集的记录所占的比例，用来衡量一个项集的频繁程度;

   关联规则(association rules)：暗指两个项集之间有可能存在很强的关系，例如{尿布} - > {葡萄酒}这条关联规则

   可信度(confidence)：针对于一条关联规则来定义的，例如“{尿布} - > {葡萄酒}”这条关联规则，其可信度为”支持度（{啤酒，尿布}）／支持度（{尿布}）”，用来衡量关联规则的关联程度。

3. Apriori原理:

   用枚举遍历所有可能的项集来发现频繁项集或者关联规则，这样的方法复杂度太高了。所以聪明的人类发现了Apriori原理。

   Apriori原理: 如果一个项集是频繁的，那么它的所有子集也是频繁的。

   Apriori原理这么看好像没有什么用，但是我们反过来看：如果一个项集是非频繁的，那么它的所有超集也是非频繁的。

   这样，就可以通过Apriori原理大幅度地减少遍历的数量了。

2. 实现:

1. 发现频繁项集:

输入: 所有集合，最小支持度

输出: 大于最小支持度的所有频繁项集的集合, 各项集对应的支持度

实现: 附录中的 def apriori(dataSet, minSupport = 0.5):

```
data_set = [[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]
L, support = apriori(data_set, 0.5)
print L, support
```

2. 发现关联规则:

输入: 频繁项集的集合，各项集对应的支持度，最小置可信度

输出：所有大于最小可信度的关联规则的集合

实现：附录中的 def generateRules(L, supportData, minConf=0.7):

```
data_set = [[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]
L, support = apriori(data_set, 0.5)
rules = generateRules(L, support, 0.7)
print rules
```

3. 扩展：

发现频繁项集还有更快的算法，叫做FP－growth算法。

附录

peter提供的代码:

```
'''
Created on Mar 24, 2011
Ch 11 code
@author: Peter
'''
from numpy import *


def loadDataSet():
    return [[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]


def createC1(dataSet):
    C1 = []
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:
                C1.append([item])

    C1.sort()
    return map(frozenset, C1) # use frozen set so we
                              # can use it as a key in a dict


def scanD(D, Ck, minSupport):
    ssCnt = {}
    for tid in D:
        for can in Ck:
            if can.issubset(tid):
                if not ssCnt.has_key(can): ssCnt[can]=1
                else: 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


def aprioriGen(Lk, k): #creates Ck
    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: #if first k-2 elements are equal
                retList.append(Lk[i] | Lk[j]) #set union
```

```
    return retList


def apriori(dataSet, minSupport = 0.5):
    C1 = createC1(dataSet)
    D = 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)#scan DB to get Lk
        supportData.update(supK)
        L.append(Lk)
        k += 1
    return L, supportData


def generateRules(L, supportData, minConf=0.7):  #supportData is a dict coming from sc
    bigRuleList = []
    for i in range(1, len(L)):#only get the sets with two or more items
        for freqSet in L[i]:
            H1 = [frozenset([item]) for item in freqSet]
            if (i > 1):
                rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf)
            else:
                calcConf(freqSet, H1, supportData, bigRuleList, minConf)
    return bigRuleList

def calcConf(freqSet, H, supportData, brl, minConf=0.7):
    prunedH = [] #create new list to return
    for conseq in H:
        conf = supportData[freqSet]/supportData[freqSet-conseq] #calc confidence
        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):
    m = len(H[0])
    if (len(freqSet) > (m + 1)): # try further merging
        Hmp1 = aprioriGen(H, m+1) # create Hm+1 new candidates
        Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)
        if (len(Hmp1) > 1):    # need at least two sets to merge
            rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)


def using_example():
    data_set = loadDataSet()
    L, support = apriori(data_set, 0.5) # find the frequent item set
```

```
    rules = generateRules(L, support, 0.7) # find the relative rules
    print rules



if __name__ == '__main__':
    using_example()
```