apriori

January 5, 2020

[1]:
```
from numpy import *
```

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

[3]:
```
def createC1(dataSet):
    C1 = []
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:
                C1.append([item])

    C1.sort()
    return list(map(frozenset, C1))
```

[4]:
```
def scanD(D, Ck, minSupport):
    ssCnt = {}
    for tid in D:
        for can in Ck:
            if can.issubset(tid):
                if not can in ssCnt: 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
```

[5]:
```
dataSet = loadDataSet()
dataSet
```

[5]:
```
[[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]
```

[6]:
```
C1 = createC1(dataSet)
```

[7]:
```
C1
```

[7]:
```
[frozenset({1}),
 frozenset({2}),
 frozenset({3}),
 frozenset({4}),
 frozenset({5})]
```

[8]:
```
D = list(map(set,dataSet))
D
```

[8]:
```
[{1, 3, 4}, {2, 3, 5}, {1, 2, 3, 5}, {2, 5}]
```

[9]:
```
L1,suppDat0 = scanD(D,C1,0.5)
L1
```

[9]:
```
[frozenset({5}), frozenset({2}), frozenset({3}), frozenset({1})]
```

[10]:
```
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
```

[11]:
```
def apriori(dataSet, minSupport = 0.5):
    C1 = createC1(dataSet)
    D = list(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
```

[13]:
```
L,suppData = apriori(dataSet)
L
```

[13]:
```
[[frozenset({5}), frozenset({2}), frozenset({3}), frozenset({1})],
 [frozenset({2, 3}), frozenset({3, 5}), frozenset({2, 5}), frozenset({1, 3})],
 [frozenset({2, 3, 5})],
 []]
```

[14]:
```
L[0]
```

[14]:
```
[frozenset({5}), frozenset({2}), frozenset({3}), frozenset({1})]
```

[15]:
```
L[1]
```

[15]:
```
[frozenset({2, 3}), frozenset({3, 5}), frozenset({2, 5}), frozenset({1, 3})]
```

[16]:
```
L[2]
```

[16]:
```
[frozenset({2, 3, 5})]
```

[17]:
```
aprioriGen(L[0],2)
```

[17]:
```
[frozenset({2, 5}),
 frozenset({3, 5}),
 frozenset({1, 5}),
 frozenset({2, 3}),
 frozenset({1, 2}),
 frozenset({1, 3})]
```

[18]:
```
def generateRules(L, supportData, minConf=0.7):  #supportData is a dict coming␣
 ↪from scanD
    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
```

[19]:
```
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
```

[20]:
```
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)
```

[21]:
```
L,suppData= apriori(dataSet,minSupport=0.5)
```

[22]:
```
rules= generateRules(L,suppData, minConf=0.7)
```

```
frozenset({5}) --> frozenset({2}) conf: 1.0
frozenset({2}) --> frozenset({5}) conf: 1.0
frozenset({1}) --> frozenset({3}) conf: 1.0
```

[ ]:
```

```