**APRIORI IMPLEMENTATION**

**NAME:**

**REG :**

**PROBLEM DISCRIPTION:**

Apriori is an algorithm for frequent item set mining and [association rule learning](https://en.wikipedia.org/wiki/Association_rule_learning" \o "Association rule learning) over transactional [databases](https://en.wikipedia.org/wiki/Databases" \o "Databases). It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine [association rules](https://en.wikipedia.org/wiki/Association_rules" \o "Association rules) which highlight general trends in the [database](https://en.wikipedia.org/wiki/Database" \o "Database). The Algorithm to be implemented is the Frequent Itemset Mining Algorithm which allows marketing platforms (or other related commercial businesses) to be able to find probable correlations between their products from the purchase records of their customers. This has applications in domains such as [market basket analysis](https://en.wikipedia.org/wiki/Market_basket_analysis" \o "Market basket analysis).

//above given description is about apriori algorithm (it’s copied from wikipedia)

//describe here about your project

**ALGORITHM DESIGN:**

// I can’t get what type of design ( flow chart or program design or algorithm design).

// make sure and edit it in your own.

**SOURCE CODE: //**  it’s python code

**CODE:**

# coding: utf-8

# Read csv in the form of dataframe

'''import pandas as pd

data=pd.read\_csv('Apriori.csv')

#data.list <- split(data.df, seq(nrow(data.df)))

data=data.values.tolist()

data'''

# In[100]:

import csv

import sys

with open('Apriori.csv','r') as f:

reader=csv.reader(f)

data=list(reader)

data

# In[101]:

# to remove nan

'''for i in range(len(data)):

data[i]=[x for x in data[i] if str(x) != 'nan']

data'''

# In[102]:

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))#use frozen set so we

#can use it as a key in a dict

# In[103]:

#D is a dataset in the setform.

D = list(map(set,data))

D

# In[104]:

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 = {}

ssCnt

for key in ssCnt:

support = ssCnt[key]#/numItems

if support >= minSupport:

retList.insert(0,key)

supportData[key] = support

return retList, supportData

# In[105]:

C1 = createC1(data)

C1

# In[106]:

L1,suppDat0 = scanD(D,C1,2)

L1

# In[107]:

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

# In[108]:

def apriori(dataSet, minSupport = 2):

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

# In[109]:

L,suppData = apriori(data)

L

# In[110]:

def generateRules(L, supportData, minConf=0.6): #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

# In[111]:

def calcConf(freqSet, H, supportData, brl, minConf=0.6):

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

# In[112]:

def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.6):

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)

# In[113]:

L,suppData= apriori(data,minSupport=2)

L

# In[114]:

rules= generateRules(L,suppData, minConf=0.6)

**TEST CASES:**

**CASE I:**

[['i1', 'i2', 'i5'],

['i2', 'i4'], ['i2', 'i3'],

['i1', 'i2', 'i4'], ['i1', 'i3'],

['i2', 'i3'], ['i1', 'i3'],

['i1', 'i2', 'i3', 'i5'],

['i1', 'i2', 'i3']]

frozenset({'i3'}) --> frozenset({'i1'}) conf: 0.6666666666666666

frozenset({'i1'}) --> frozenset({'i3'}) conf: 0.6666666666666666

frozenset({'i3'}) --> frozenset({'i2'}) conf: 0.6666666666666666

frozenset({'i4'}) --> frozenset({'i2'}) conf: 1.0

frozenset({'i1'}) --> frozenset({'i2'}) conf: 0.6666666666666666

frozenset({'i5'}) --> frozenset({'i1'}) conf: 1.0

frozenset({'i5'}) --> frozenset({'i2'}) conf: 1.0

frozenset({'i5'}) --> frozenset({'i2', 'i1'}) conf: 1.0

**CASE II:**

[['apple', 'beer', 'rice', 'chicken'],

['apple', 'beer', 'rice'],

['apple', 'beer'],

['apple', 'mango'],

['milk', 'beer', 'rice', 'chicken'],

['milk', 'beer', 'rice'],

['milk', 'beer'], ['milk', 'mango']]

frozenset({'milk'}) --> frozenset({'beer'}) conf: 0.75

frozenset({'apple'}) --> frozenset({'beer'}) conf: 0.75

frozenset({'chicken'}) --> frozenset({'beer'}) conf: 1.0

frozenset({'beer'}) --> frozenset({'rice'}) conf: 0.6666666666666666

frozenset({'rice'}) --> frozenset({'beer'}) conf: 1.0

frozenset({'chicken'}) --> frozenset({'rice'}) conf: 1.0

frozenset({'chicken'}) --> frozenset({'rice', 'beer'}) conf: 1.0

**CASE III:**

[['a', 'b', 'c', 'd'],

['c', 'd'],

['a', 'b', 'd'],

['d', 'e', 'f', 'g', 'h'],

['a', 'c', 'h'],

['d'],

['h', 'g']]

frozenset({'h'}) --> frozenset({'g'}) conf: 0.6666666666666666

frozenset({'g'}) --> frozenset({'h'}) conf: 1.0

frozenset({'a'}) --> frozenset({'b'}) conf: 0.6666666666666666

frozenset({'b'}) --> frozenset({'a'}) conf: 1.0

frozenset({'a'}) --> frozenset({'c'}) conf: 0.6666666666666666

frozenset({'c'}) --> frozenset({'a'}) conf: 0.6666666666666666

frozenset({'a'}) --> frozenset({'d'}) conf: 0.6666666666666666

frozenset({'b'}) --> frozenset({'d'}) conf: 1.0

frozenset({'c'}) --> frozenset({'d'}) conf: 0.6666666666666666

frozenset({'a'}) --> frozenset({'d', 'b'}) conf: 0.6666666666666666

frozenset({'b'}) --> frozenset({'d', 'a'}) conf: 1.0

// I think it’s enough for test cases, if it’s not enough or you want to add something extra then add yourself by run the project and add the output, (change different input items in the .csv file and make the test cases.

**SUMMARY:**

The goal of this project is to implement the Apriori algorithm to find all frequent itemsets. Also, generate all Strong Association Rules from the frequent itemsets, And, find minimum support and minimum confidence values from the transaction database.

// it’s also by own if you want add some more points.