Course name: Data Science(ITE4005)
Student id: 2016024793
Name: 김유진

Data Science
Programming Assignment #1

PROGRAMMING ASSIGNMENT #1 – APRIORI ALGORITHM

ENVIRONMENT

OS: Windows 10

Language: python 3.6.8

Used library: sys, combinations and chain from itertools

```
import sys
from itertools import combinations, chain
```

SUMMARY OF ALGORITHM

In order to get all the association rules, apriori algorithm was used to mine all frequent itemsets. All the support value data of candidate itemsets were saved during the process.

Next, association rules were generated and following confidence value were calculated using frequent itemsets and support value data minded from apriori algorithm.

DETAILED DESCRIPTIONS

1. loadDatabase()

A. function that read the inputfile and returns list containing transactions. Used in `apriori()` function.

B. Parameters

i. inputfile: str type, name of the input file containing data

C. Returns

i. data: list of sets, each set means a transaction

```
def loadDatabase(inputfile):
    f = open(inputfile, mode='rt')
    data = []
    while True:
        line = f.readline()
        if not line:
            break
        line = line[:-1]
```

```
        items = []
        for i in line.split('\t'):
            items.append(int(i))
        data.append(set(items))
    f.close()
    return data
```

2. scanDatabase()

A. function that calculate support value for each candidate and returns frequent itemsets. Used in the apriori() function.

B. Parameters

i. D: list of sets, datasets which contains transactions

ii. Ck: list of frozensets, candidate itemsets of size k

iii. minSupport: float, minimum support value

C. Returns

i. Lk: list of frozensets, frequent itemset of size k

ii. supportData: dictionary of which key is itemset and value is support value

```
def scanDatabase(D, Ck, minSupport):
    supportCnt = {}

    for cand in Ck:
        for trans in D:
            if cand.issubset(trans):
                if not cand in supportCnt:
                    supportCnt[cand] = 1
                else:
                    supportCnt[cand] += 1
    transNum = float(len(D))
    Lk = []                 # frequent itemsets of size k
    supportData = {}        # support data
    for key, value in supportCnt.items():
        s = value / transNum * 100
        if s >= float(minSupport):
            Lk.append(key)
        supportData[key] = s
    return Lk, supportData
```

3. createC1()

A. function that generate candidate itemset of size 1 by scanning through database. Used in the apriori() function.

B. Parameters

i. D: list of sets. Each set means a transaction

C. Returns

i. C1: list containing candidate itemset of length 1. Each element is frozenset type, in order to use it as key of dictionary.

```
def createC1(D):
    C1 = []             # candidate itemsets of size 1
    for trans in D:
        for item in trans:
            if not [item] in C1:
                C1.append([item])
    C1.sort()
    return map(frozenset, C1)
```

4. generateCandidate()

A. function that generate candidate itemsets of length k from frequent itemsets. Used in the apriori() function.

B. Parameters

i. Lk: frequent itemsets

ii. k: integer, length of the candidate itemsets

C. Returns

i. List of frozensets, each means a candidate itemset

```
def generateCandidate(Lk, k):
    retList =  []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i+1,lenLk):
            L1 = Lk[i]; L2 = Lk[j]
            if len(L1 & L2) == k-2:
                if not (L1 | L2) in retList:
                    retList.append(L1 | L2)
    return retList
```

5. apriori()

A. function that implements apriori algorithm. While Lk(frequent itemsets of length k) is not empty it generate candidates using generateCandidate() function and check its support data using scanDatabase() function. The function gives us a list that contains all the frequent sets and a dictionary containing support values.

B. Parameters

i. D: list of sets, dataset from input file

ii. minSupport: minimum support value, given by user

C. Returns

i. L: list containing lists of frequent itemsets. Each element is a list containing frequent itemsets that have same length.

ii. supportData: dictionary containing support value data. Its keys are itemsets and its values are support value.

```
def apriori(D, minSupport):
    C1 = createC1(D)
    L1, supportData = scanDatabase(D,C1,minSupport)
    L = [L1]
    k = 2
    while(len(L[k-2]) > 0):
        Ck = generateCandidate(L[k-2], k)
        Lk, sup = scanDatabase(D, Ck, minSupport)
        supportData.update(sup)
        L.append(Lk)
        k += 1
    return L[:-1],supportData
```

6. getSubsets()

A. function that finds the subsets of input array. Used in generateRules() function. Used combinations() and chain() from itertools library.

B. Parameters: a set or list

C. Returns: list of subsets

```
def getSubsets(arr):
    return list(chain(*[combinations(arr, i+1) for i in range(len(arr))]))
```

7. generateRules()

A. function that generate association rules from given frequent itemsets and support value data.

B. Parameters

i. L: frequent itemsets

ii. supportData: support datas

C. Returns

i. retRules: List containing tuples. Each tuple means ((itemset, associative itemset), confidence value).

```
def generateRules(L,supportData):
    retRules = []
    for Lk in L[1:]:
        for item in Lk:
            subsets = map(frozenset, [x for x in getSubsets(item)])
            for element in subsets:
                remain = item.difference(element)
                if len(remain) > 0:
                    confidence = supportData[item] / supportData[element] * 100
                    retRules.append(((element, remain), confidence))
    return retRules
```

8. Main

A. Take 3 arguments and print data properly. First, it takes minimum support value, input file name, and output file name from the user. Next, it loads data in the form of list using loadDatabase() function. Then it mines frequent itemsets from the data using apriori(). Finally it finds out association rules using generateRules() function.

```
if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("need 3 arguments")
    else:
        # Execute the program with three arguments.
        #    minimum support
        #    input file name
        #    output file name
        minSupport = float(sys.argv[1])
        inputfile = sys.argv[2]
        outputfile = sys.argv[3]

        fout = open(outputfile, 'w')

        database = loadDatabase(inputfile)
        L, supportData = apriori(database, minSupport)
```

```
        assocRules = generateRules(L, supportData)

        for (a, b), conf in assocRules:
            sup = supportData[a | b]
            fout.write("{}\t{}\t".format(set(a), set(b)))
            fout.write("{0:.2f}\t".format(sup))
            fout.write("{0:.2f}\n".format(conf))

        fout.close()
```

COMPILATION

Please use the command below.

python apriori.py [*minimum support(%)*] [*input file name*] [*output file name*]

For example, `\ds\assignment>python apriori.py 5 input.txt output.txt`.