2014000082 구희강

Data Science

PROGRAMMING ASSIGNMENT #1 - APRIORI ALGORITHM

Code Description and Algorithm

<module>

```
import itertools
import sys
```

Used <itertools> module to generate combination set of items and <sys> module to get arguments from prompt (minimum support, input file name and output file name)

<main>

```
if __name__ == "__main__":
    minSup = int(sys.argv[1])
    name_in = sys.argv[2]
    name_out = sys.argv[3]

    # read data from input file
    data_in = readData(name_in)
    # create data
    data_out = scanData(data_in,minSup)
    # write output file
    writeData(data_out,name_out)
```

First, call readData() function which returns list type data(data_in). With input file data and given minimum support, call scanData() function which returns list type of data(data_out). As last step, write result file by calling writeData() function.

<scanData>

```
def scanData(data,minSup):
    level = 1
    # value to return
    ret = []
    lenData = len(data)
    candidates = []

    # create level 1 candidates
    for transaction in data:
        for item in transaction:
            if [item] not in candidates:
                candidates.append([item])
    candidates.sort()
    # calculate support value
    count = []
    for i in candidates:
        count.append(counts(i,data))
    for i in range(len(count)):
        count[i] = count[i]/lenData*100
        # pruning marking
        if count[i]<minSup:
            candidates[i] = []
    # pruning
    candidates = list(filter(lambda a: a != [], candidates))
```

First, find single item candidates from whole transaction data and sort candidate items. Then prune candidates with given minimum support value. The counts() function returns the number of transactions containing certain candidates

```
while True:
        # create new candidates of upper level
        tmp = candidates
        candidates = []
        for i in range(len(tmp)):
            for j in range(i+1,len(tmp)):
                l1 = tmp[i]
                l2 = tmp[j]
                if matchK(level-1,l1,l2):
                    rule = list(set(l1 + l2))
                    rule.sort()
                    candidates.append(rule)
```

In while statement, it creates new candidates of next level by merge two different previous candidates. The function matchK() returns False if the two candidates' (level-1) number of element from the head does not match. By this function, new candidates could be generated without overlapped.

```
# calculate support value
        count = []
        for i in candidates:
            count.append(counts(i,data))
        for i in range(len(count)):
            count[i] = count[i]/lenData*100
            # pruning marking
            if count[i]<minSup:
                candidates[i] = []
        # pruning
        candidates = list(filter(lambda a: a != [], candidates))
```

Then, while iterating new candidates, it calculates support value and prune by given minimum support value.

```
        if len(candidates) == 0 :
            break
        level = level + 1
        ret.append(write(candidates,data))

    # return output data
    return ret
```

At each level, write() function get candidates value and return new form of string(item sets, support, confidence). Each string is appended to 'ret' list. The while statement is stopped when there are no more candidates. By return the 'ret' value scanData() function ends.

<write>

```
def write(candidates,data):
    line = []
    for c in candidates:
        sublist = sublists(c)
        for sub in sublist:
            item = set(sub)
            asso = set(c) - set(sub)
            sup = getSup(c,data)
            conf = getConf(c,item,data)

            line.append(str(item)+'\t'+str(asso)+'\t'+format(sup,".2f")+'\t'+fo
            rmat(conf,".2f"))
    return line
```
With candidate items, and original transaction data, creates all possible association rules with support and confidence.

<sublists>

```
def sublists(data):
    ret = []
    for i in range(1,len(data)):
        ret = ret + list(itertools.combinations(data,i))
    for i in range(len(ret)):
        ret[i] = set(ret[i])
    return ret
```
sublists() function makes all possible proper subsets without empty set.

<getSup & getConf>

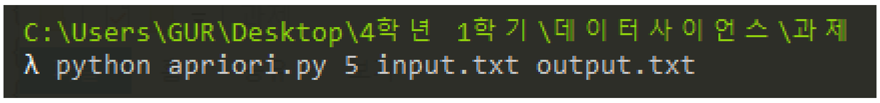
```
def getSup(items,data):
    return counts(items,data)/len(data)*100


def getConf(items,x,data):
    return counts(items,data)/counts(x,data)*100
```
getSup() returns support value(%) and getConf() returns confidence value(%).

Test Result

Program usage :

>> python priori.py [min support] [input file name] [output file name]

<example>



```
C:\Users\GUR\Desktop\4학년 1학기\데이터사이언스\과제
λ python apriori.py 5 input.txt output.txt
```

Test environment

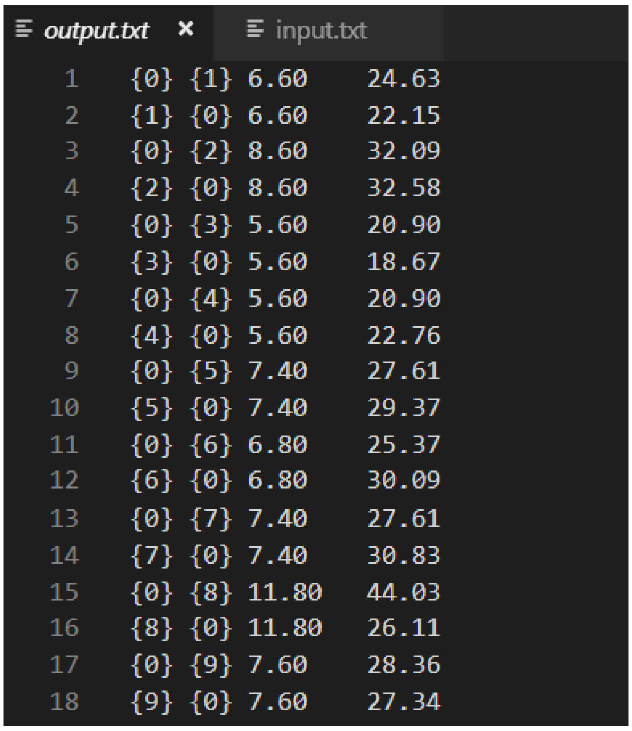
- Windows 10

Test requirement

- Python 3

Python program created all possible association rules and it's support value and confidence value.

<output text file>



```
{0} {1} 6.60    24.63
{1} {0} 6.60    22.15
{0} {2} 8.60    32.09
{2} {0} 8.60    32.58
{0} {3} 5.60    20.90
{3} {0} 5.60    18.67
{0} {4} 5.60    20.90
{4} {0} 5.60    22.76
{0} {5} 7.40    27.61
{5} {0} 7.40    29.37
{0} {6} 6.80    25.37
{6} {0} 6.80    30.09
{0} {7} 7.40    27.61
{7} {0} 7.40    30.83
{0} {8} 11.80   44.03
{8} {0} 11.80   26.11
{0} {9} 7.60    28.36
{9} {0} 7.60    27.34
```