

*[Assignment report]*

# Association Rule by Apriori Algorithm

2012023871

경영학과

강민철

## 1) Summary of Algorithm

Apriori algorithm finds frequent item sets by generating confined candidate set. First, the algorithm extract frequent itemsets(L1) with one elements from the given minimum support by scanning the input.txt file. It proceeds to generate candidate set(C2) from L1 and filter L2 set by rejecting itemsets whose support is lower than the minimum support. The finding of each  $L_k$  requires one full scan of the database which can be massively expensive as the occasion demands.

## 2) Detailed Description of Codes

*[Library Import]*

```
from optparse import OptionParser
from itertools import chain, combinations
from collections import defaultdict
```

- 'Optionparser' enables a .py file to take arguments on the bash level.
- 'chain and combinations' are functions needed to process set operator such as generating subset.
- 'defaultdict' is used to count the frequency of each combination of itemset

### [Main function]

```
if __name__ == "__main__":
    #optionParser setting
    optparser = OptionParser()
    optparser.add_option('-s', '--minSupport', dest='minSupport',
                        help='Minimum Support value by %!!!',
                        default=12, type='float')
    optparser.add_option('-i', '--input', dest='input',
                        help='Input file name',
                        default='input.txt')
    optparser.add_option('-o', '--output', dest='output',
                        help='output file name', default='output.txt')

    (options, args) = optparser.parse_args()

    Data = Generator(options.input)
    minSupport = options.minSupport * 0.01

    Asso_rules = ProcessApriori(Data, minSupport)
    #Make output.txt file
    out_file = open(options.output, 'w')

    #Write obtained association rules into the output file
    Result = WriteFile(Asso_rules, out_file)
    out_file.close() #Close file
```

First few lines are for setting the OptionParser. Specifically, 1<sup>st</sup> option takes the float type input data from users and store it in the minSupport instance (default is 12%), 2<sup>nd</sup> option takes input data file name while 3<sup>rd</sup> option takes the file name of output file.

Then call generation function to read input file and change the unit of minSupport. ProcessApriori function is the principal function which runs apriori algorithm. I will explain it in detail later. By using the options.output, we make an empty output.txt file. WriteFile function write the obtained association rules into the generated output file. Finally, we have an output file which contains association rules and related data and close the file.

### [Generator function]

```
def Generator(file_path):
    file_iter = open(file_path, 'rU')
    for trID in file_iter:
        trID = trID.strip().rstrip('\t') # Remove disturbing tab
        tr = frozenset(trID.split('\t'))
        yield tr
```

This function opens the input file and read the file by each rows (transaction ID). As the input.txt file used '\t' to separate value, it split the value by that symbol. This yields an object which can only be accessed by each rows. (Different from return)

### [ProcessApriori function -1]

```
def ProcessApriori(data_iter, minSupport):
    # Extract non-overlapping 1-itemset(itemSet) and transaction list(trList)
    itemSet, trList = ClassifyInput(data_iter)

    # Make default dictionary to count the frequency of each transaction
    FreqSet = defaultdict(int)

    # To store whole L set
    wholeSet = dict()

    # Find frequent 1-itemsets
    Freq_One = MinSupportCheck(itemSet, trList, minSupport, FreqSet)

    # Set Freq_One as L1
    LSet_K = Freq_One

    # Starting with L2 and so on until there is no element in L(k-1) set
    K = 2
    while(LSet_K != set([])):
        # record L(k-1) in whole set with index
        wholeSet[K-1] = LSet_K

        #Generate Candidate Set by using union function
        CandSet_K = set([i.union(j) for i in LSet_K for j in LSet_K if len(i.union(j)) == K])

        #Filter Candidate Set which has lower support than minSupport
        LSet_K = MinSupportCheck(CandSet_K, trList, minSupport, FreqSet)

        K += 1
```

By using the ClassifyInput function below, processapriori function gets itemSet (non-overlapping 1-itemset) and trList(transaction set). Then it makes default dictionary to count the frequency of each subsets of transaction. The wholeSet is for storing the whole L set.

By applying MinSupportCheck function to itemSet, we get frequent 1-itemsets which meets the condition of minSupport. We proceed to set this Freq\_One as L1

and starts with finding L2 and iterate the process until there is no element in L<sub>k-1</sub> set. The process is the code block under while. As I briefly explained in the algorithm summary, it generates candidate set by using union function of Python. Then it filters the candidate set which has lower support than minSupport.

### ***[ClassifyInput function]***

```
def ClassifyInput(data):  
    trList = list()  
    itemSet = set()  
    for tr in data:  
        transaction = frozenset(tr)  
        trList.append(transaction)  
  
        # Make non-overlapping set  
        for item in transaction:  
            itemSet.add(frozenset([item])) #characteristics of 'add' function  
    return itemSet, trList
```

This function classifies the information of input file(the object yielded from Generator function). frozenset transforms the indexed list to a set type instance. trList.append(transaction) is the code that store each transaction items by the transaction ID. itemSet.add(frozenset([item])) has a unique characteristic that it removes the duplicated element and only stores it as a single element. It returns itemSet and trList which is used in the processapriori function.

### ***[MinSupportCheck function]***

```
def MinSupportCheck(itemSet, trList, minSupport, FreqSet):  
    SurvivedSet = set()  
    temp = defaultdict(int) #As we use FreqSet out of function, we need same temporary data  
  
    for item in itemSet:  
        for transaction in trList:  
            if item.issubset(transaction):  
                FreqSet[item] += 1  
                temp[item] += 1  
  
    for item, cnt in temp.items(): #Use temp instead of FreqSet  
        support = float(cnt)/len(trList)  
  
        if support >= minSupport:  
            SurvivedSet.add(item) #Add item whose support over minSupport  
  
    return SurvivedSet
```

MinSupportCheck function checks the support of given itemSet, compare with MinSupport, and only contains the survived itemSet in SurvivedSet.

As we use FreqSet out of this function, we need temporary data which bounded and can be disposed when this function is over. That is why it uses temp instead of FreqSet in third for loop above. Still, it updates the value of FreqSet.

### **[ProcessApriori function -2]**

```
#local function that makes subset of given set
def makeSubset(Set):
    return chain(*[combinations(Set, i + 1) for i, a in enumerate(Set)])

RuleResult = []
for key, SurvSet in wholeSet.items()[1:]: #Total set of survived transactions index: number of element
    for item in SurvSet:
        #Generate subsets
        Subsets = map(frozenset, [x for x in makeSubset(item)])

        for element in Subsets: #pick arbitrary element from Subsets
            left = item.difference(element) #complementary set of element with respect to subset
            if len(left) > 0:
                #Calculate rule support
                ruleSupport = float(FreqSet[item])/len(trList)

                #Calculate rule confidence
                itemSupport = float(FreqSet[item])/len(trList)
                elementSupport = float(FreqSet[element])/len(trList)
                ruleConfidence = itemSupport/elementSupport

                #As there is restriction about the minimum confidence we don't check that
                #Save as list according in accordance with output requirement
                RuleResult.append(((tuple(element), tuple(left)), ruleSupport, ruleConfidence))

return RuleResult
```

This is the second part of processapriori function. (Don't misunderstand; part 1 and 2 are actually in the same function) There is a local function named makeSubset which makes subsets of given set. As we extracted wholeSet which contains every combinations of transaction items which meets the minimum support condition. For each item in these combinations, we generate subsets of the items and pick arbitrary element from that subset. We use the arbitrary element and if\_items of association rule and draw then\_items by extracting complementary set of that if\_items with respect to subset. Then we calculate the support and confidence of every association rule respectively. As there is no restriction about the minimum confidence in the project guideline, we don't check it. Finally, we save the result as list in accordance with the output format.

### [WriteFile function]

```
def WriteFile(Asso_rules, f):  
    #Pick each row by the value of rule confidence (Descending)  
    for rule, support, confidence in sorted(Asso_rules, key=lambda (rule, support, confidence): confidence, reverse=True):  
        base_item, asso_item = rule  
  
        #As the data format is 'set' in python, we should extract the value only  
        base_item = [int(list(base_item)[i]) for i in xrange(len(list(base_item)))]  
        asso_item = [int(list(asso_item)[i]) for i in xrange(len(list(asso_item)))]  
        base_item_str = str(base_item[0])  
        asso_item_str = str(asso_item[0])  
  
        #To meet the output file format, unlist the item set and make as string instance  
        for i in xrange(len(base_item)-1):  
            base_item_str += ','  
            base_item_str += str(base_item[i+1])  
  
        for j in xrange(len(asso_item)-1):  
            asso_item_str += ','  
            asso_item_str += str(asso_item[j+1])  
  
        #Change support and confidence unit to percent and round to two decimal point  
        data= "{%s}\t{%s}\t%.2f\t%.2f\n" % (base_item_str, asso_item_str, support*100, confidence*100)  
        f.write(data)  
  
    return f
```

This function picks each row of Asso\_rules by the value of rule confidence in a descending manner. However, the result's type is python 'set', we should extract the value only. To meet the output file format, it unlists the item set and make as string instance. when we write each line to the output file f, it changes the value of support and confidence from percent to float, and rounds to two decimal point. It returns the edited output file f.

### 3) Instructions for compiling the source codes

[Step 1]: copy the repository and change directory to ~/project\_apriori/

```
git clone http://hconnect.hanyang.ac.kr/2017_ITE4005_10065/2017_ITE4005_2012023871.git
```

- Copy my gitlab repository by using *git clone* or download the *zip file* from the website. [http://hconnect.hanyang.ac.kr/2017\_ITE4005\_10065/2017\_ITE4005\_2012023871]

```
cd DS_git/2017_ITE4005_2012023871/Programming_Assignment_1/project_apriori/
```

- go the the project\_apriori directory by 'cd' command.

[Step 2] : Print help message about arguments

```
python apriori.py -h
```

- type *\$python apriori.py -h* on command line then it returns the help message of Apriori.py. (see below)





```
Options:
-h, --help            show this help message and exit
-s MINSUPPORT, --minSupport=MINSUPPORT
                        Minimum Support value by %!!!
-i INPUT, --input=INPUT
                        Input file name
-o OUTPUT, --output=OUTPUT
                        output file name
```

- **-S** is for minimum support and you should give the value by percentage (%)
- **-i** is for input file name. The input file must be in **the same directory**.
- **-O** is for the output file name.

[Step 3] : Actually run the code by python (**2.7 ONLY**)

```
python apriori.py -s 12 -i input.txt -o output.txt
```

- run *\$python apriori.py -s 12 -i input.txt -o output.txt* on terminal or command line (**python should be 2.7 python3 ~ will not cause error**)

이름	수정일
 apriori.py	오늘 오후 8:20
 input.txt	2017년 3월 9일 오후 3:12
 output.txt	오늘 오후 8:42
 README.md	오늘 오후 8:25

- Then you will see the "output.txt" file has been made in the same directory.

It looks like this

{3,16}	{8}	24.00	95.24
{8,3}	{16}	24.00	93.02
{3}	{8}	25.80	86.00
{3}	{16}	25.20	84.00
{3}	{8,16}	24.00	80.00
{8,16}	{3}	24.00	79.47
{16}	{8}	30.20	71.23
{8}	{16}	30.20	66.81
{16}	{3}	25.20	59.43
{8}	{3}	25.80	57.08