ASSOCIATIVE ANALYSIS REPORT

Tariq Siddiqui – 50208470

Kaushik Ramasubramanian – 50207352

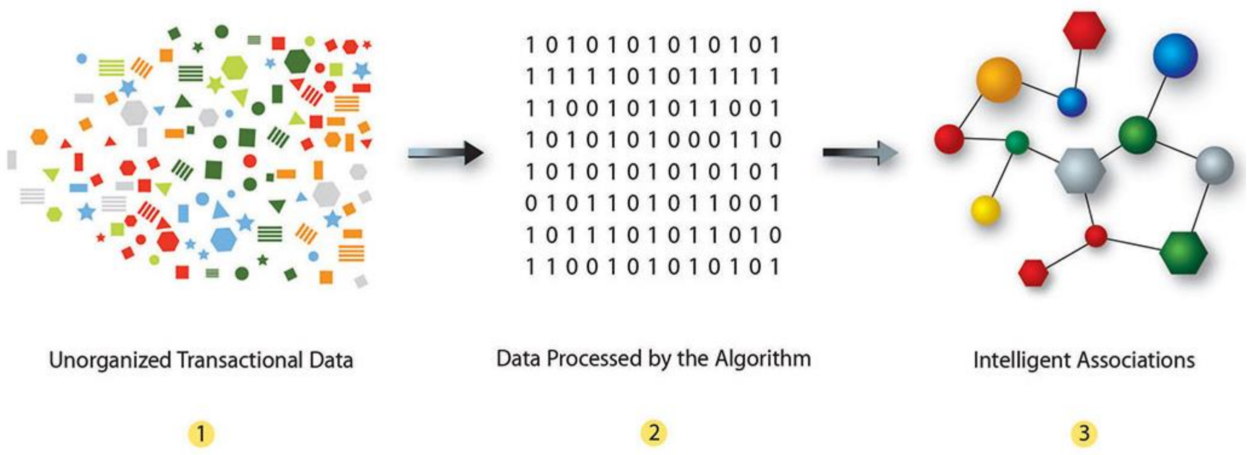
Junaid Shaikh - 50208476

1 Association Analysis

Association analysis is a process which is meant to find correlations/associations in data sets found in various kinds of databases such as relational databases, transactional databases, etc.

Given a set of transactions, association rule mining aims to find the rules which enable us to predict the occurrence of a specific item based on the occurrences of the other items in the transaction.

Association rules are if/then statements that help uncover relationships between seemingly unrelated data in a relational database or other information repository. An example of an association rule would be "If a customer buys a dozen eggs, he is 80% likely to also purchase milk."



Association rules are created by analyzing data for frequent if/then patterns and using the criteria support and confidence to identify the most important relationships (Support is an indication of how frequently the items appear in the database. Confidence indicates the number of times the if/then statements have been found to be true.)

In data mining, association rules are useful for analyzing and predicting customer behavior. They play an important part in shopping basket data analysis, product clustering, catalog design and store layout:

A person might buy milk, diaper and beer at same time. Understanding these buying patterns can help to increase sales in several ways. If there is a pair of items, Item1 and Item2, that are frequently bought together:

- ❖ Both Item1 and Item2 can be placed on the same shelf, so that buyers of one item would be prompted to buy the other.
- ❖ Advertisements on Item1 could be targeted at buyers who purchase Item2.
- ❖ Promotional discounts could be applied to just one out of the two items.
- ❖ Item1 and Item2 could be combined into a new product, such as having Item2 in flavors of Item1.

As below Amazon use association mining to recommend you the items based on the current item you are browsing/buying:

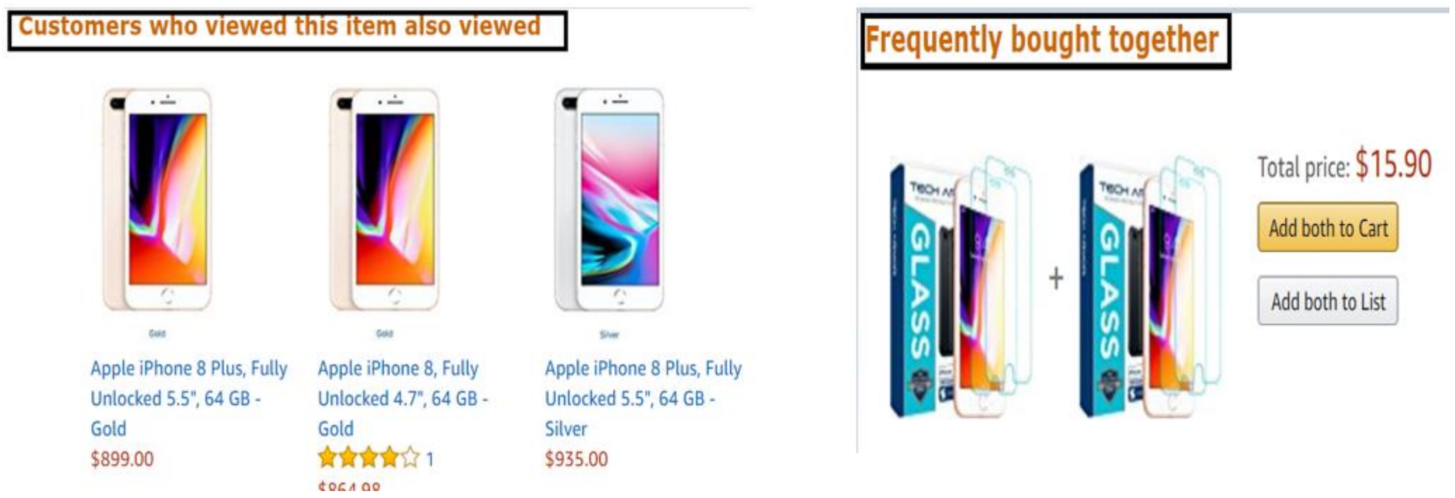

Customers who viewed this item also viewed
Apple iPhone 8 Plus, Fully Unlocked 5.5", 64 GB - Gold
$899.00
Apple iPhone 8, Fully Unlocked 4.7", 64 GB - Gold
Apple iPhone 8 Plus, Fully Unlocked 5.5", 64 GB - Silver
$935.00
Frequently bought together
GLASS
Total price: $15.90
Add both to Cart
Add both to List

Figure 1.1-1 Snapshot from Amazon depicting real world implementation of Associative Analysis

Besides increasing sales profits, association rules can also be used in other fields :

1. **In medical diagnosis for instance**, understanding which symptoms tend to co-morbid can help to improve patient care and medicine prescription.
2. Another application is the **Google auto-complete**, where after you type in a word it searches frequently associated words that user type after that particular word.

There are different common ways to measure association in data. Below we discuss 2 ways:

1.1 Measure 1: Support

This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears.

The support of an itemset X, support(X) is the proportion of transaction in the database in which the item X appears. It signifies the popularity of an itemset.

$$supp(X) = \frac{\text{Number of transaction in which} X \text{appears}}{\text{Total number of transactions}}.$$

If the sales of a particular product (item) above a certain proportion have a meaningful effect on profits, that proportion can be considered as the support threshold.

1.2 Measure 2: Confidence

This says how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. This is measured by the proportion of transactions with item X, in which item Y also appears.

Confidence of a rule is defined as follows:

$$conf(X \longrightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}$$

Confidence signifies the likelihood of item Y being purchased when item X is purchased. Confidence is an indication of how often the rule has been found to be true..

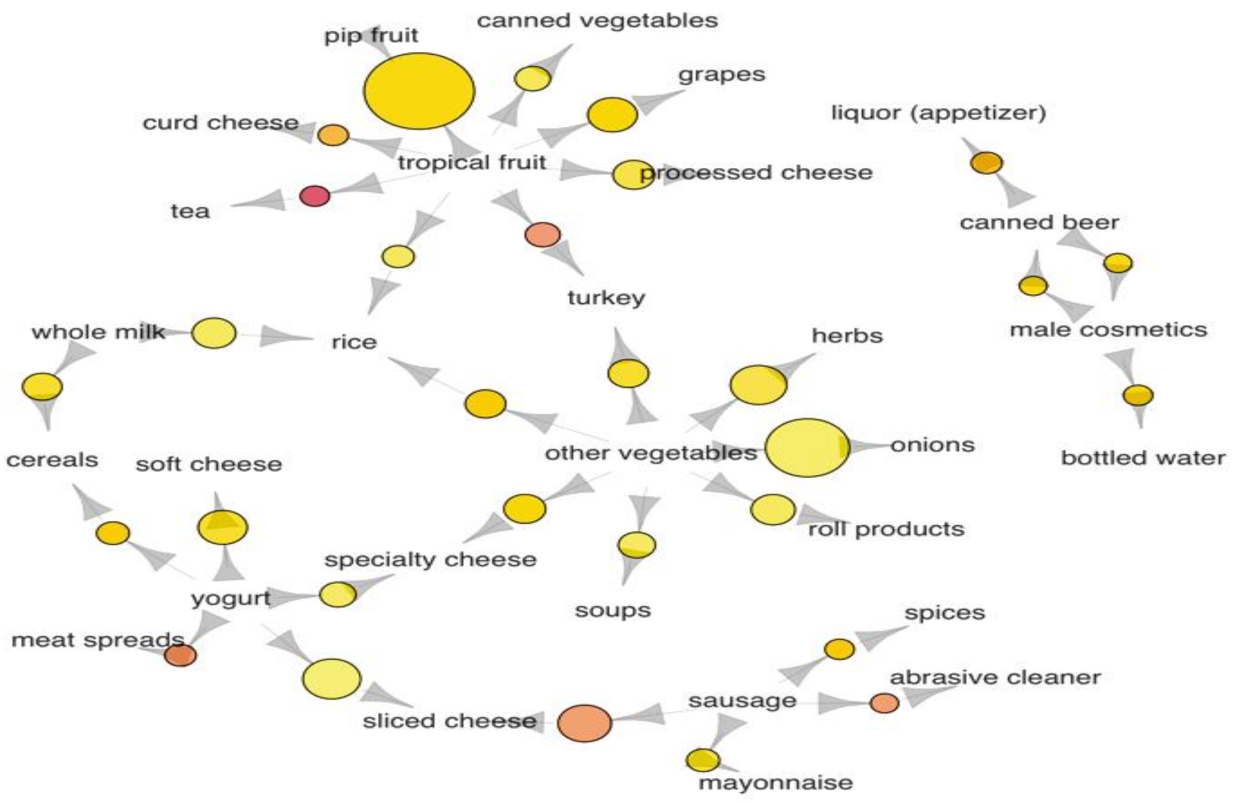


Figure 1-1.2-1 Network graph showing associations between selected items

2 Apriori Algorithm

Apriori algorithm is a classic influential algorithm used in data mining for learning boolean association rules.

Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate Frequency Itemset generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

Apriori algorithm generates candidate item sets of length K from item sets of length K-1. Then it prunes the candidates which have an infrequent sub pattern. Hence, Apriori is used to determine the frequent item sets of any database. This step is known as **Frequent Itemset generation**.

Then, next step of Apriori is to count up the number of occurrences, called the support, of each member item separately in the database

Then high confidence rules are generated from each frequent itemset, where each rule is a binary partitioning of a frequent itemset. This step is known as **rule generation**.

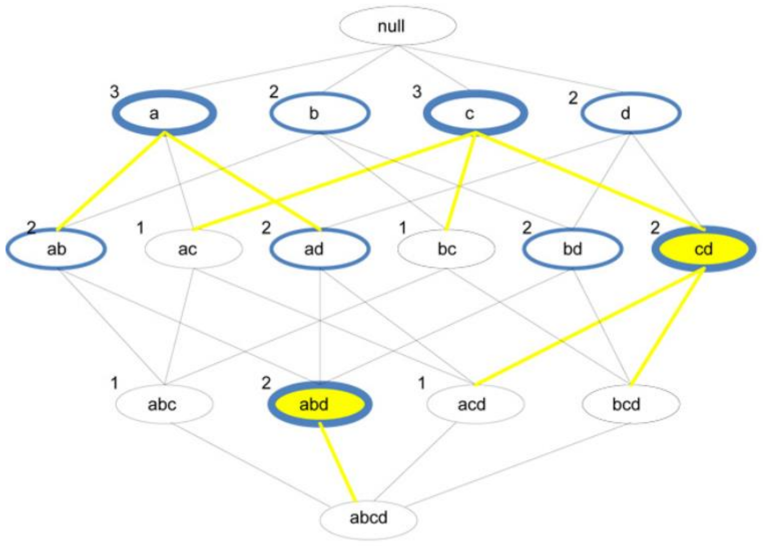
Frequent Itemset generation and rule generation are discussed in following section:

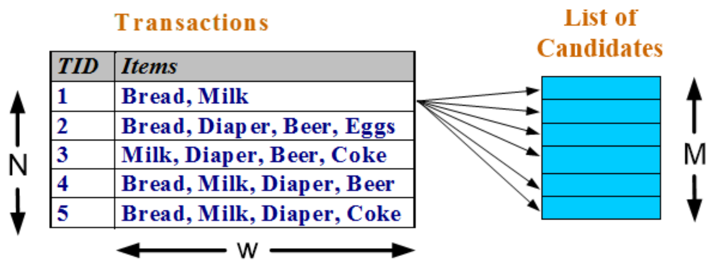
2.1 Frequent Itemset generation

Below are the steps from Apriori which are used to determine the frequent item sets of any database:

Step 1: Apply minimum support to find all the frequent sets with k items in a database.

Step 2: Use the self-join rule to find the frequent sets with k+1 items with the help of frequent k-temsets. Repeat this process from k=1 to the point when we are unable to apply self-join rule.

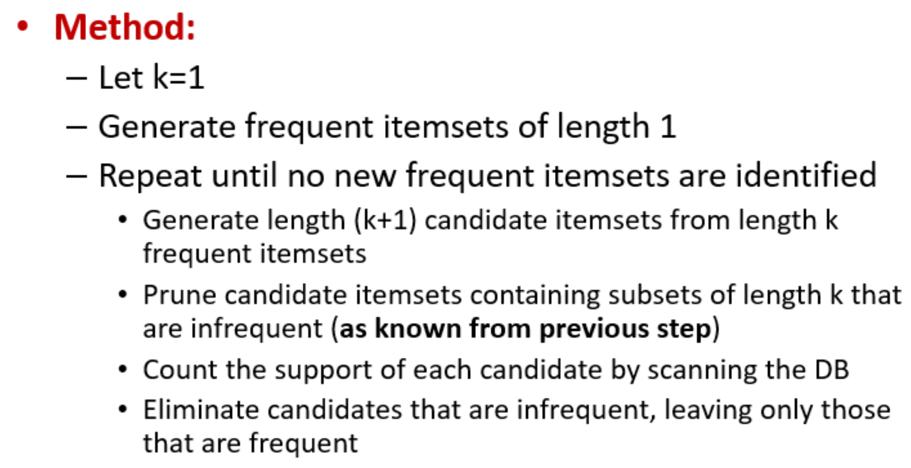




TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

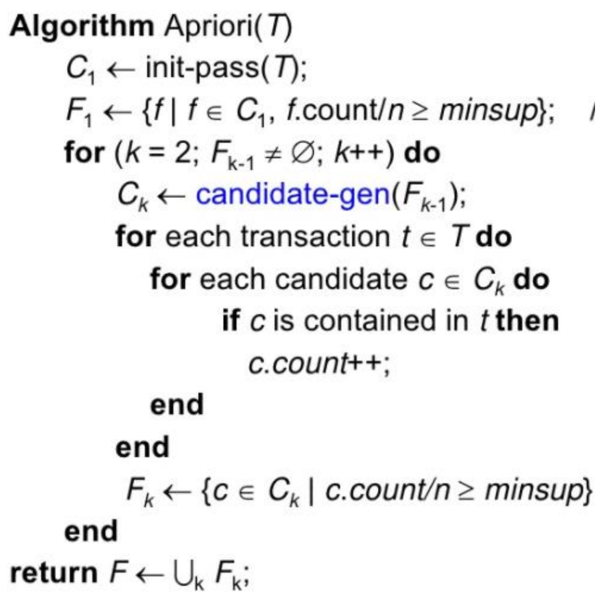
Figure 2.1-1 Frequent ItemSet Generation

Steps Frequent Itemset generation are as below:



- **Method:**
  - Let k=1
  - Generate frequent itemsets of length 1
  - Repeat until no new frequent itemsets are identified
    - Generate length (k+1) candidate itemsets from length k frequent itemsets
    - Prune candidate itemsets containing subsets of length k that are infrequent (**as known from previous step**)
    - Count the support of each candidate by scanning the DB
    - Eliminate candidates that are infrequent, leaving only those that are frequent

Below figure shows different steps of of Apriori algorithm.



```
Algorithm Apriori($T$)
    $C_1 \leftarrow$ init-pass($T$);
    $F_1 \leftarrow \{f \mid f \in C_1, f.\text{count}/n \geq minsup\}$;
    for ($k = 2$; $F_{k-1} \neq \varnothing$; $k$++) do
        $C_k \leftarrow$ candidate-gen($F_{k-1}$);
        for each transaction $t \in T$ do
            for each candidate $c \in C_k$ do
                if $c$ is contained in $t$ then
                    $c.count$++;
            end
        end
        $F_k \leftarrow \{c \in C_k \mid c.count/n \geq minsup\}$
    end
return $F \leftarrow \bigcup_k F_k$;
```

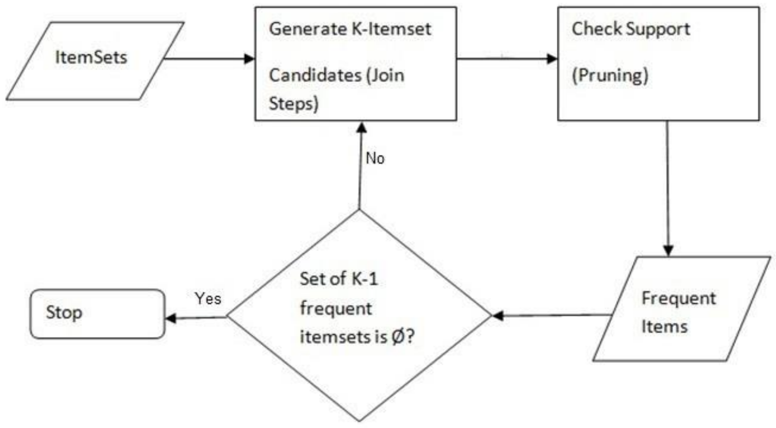
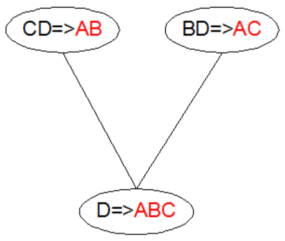


Figure 2.1-2 Flowchart showing Frequent Itemset generation Steps

2.2 Association Rules Generation

After finding frequent itemset L from previous step, association rule generation requires finding all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement. That is, Create rules from each frequent itemset using the binary partition of frequent itemsets and look for the ones with high confidence. These rules are called candidate rules.



Candidate rule is generated by merging two rules that share the same prefix in the rule consequent.

In adjacent figure, join(CD=>AB,BD=>AC) would produce the candidate rule D => ABC and Prune rule D=>ABC if its subset AD=>BC does not have high confidence

For finding association rules, we need to find all rules having support greater than the threshold support and confidence greater than the threshold confidence.

If X is a frequent itemset with k elements, then there are $2^k - 2$ candidate association rules.

3 Implementation Details

Our **Python programming based implementation** uses below algorithm for frequent Itemset generation. Our python programming extensively uses data structure Set. **Python uses HashMap to internally implement set, thereby, providing access of order of O(1) , provides way to count candidate item sets efficiently and reduces the number of comparisons**

1> Read data from file and save as 2D matrix. This 2D matrix contains the status of parameter (Genes and diseases in this case) for a given record.
2> Make frequency item set of length 1 from the data read as explained in 4.1 section.
3> Count the support of each candidate frequent itemset by scanning all the record in step1.

4> Using above, Prune candidate itemsets that are infrequent by comparing against minSupport threshold.
5> From remaining Itemsets of length1, Generate length2 items.
6> From above, Prune candidate itemsets of size 2 that are infrequent
7> **Also, Prune candidate itemsets containing subsets of (current_length-1) that are infrequent (as were calculated in previous step)**
8> Repeat above steps until no new frequent itemsets are identified.
9> Now using combinations utility in python, generate all combination of a frequent itemset at a time.
10> Check all these combinations against confidence threshold. If candidate rule is smaller than confidence, rule is invalid otherwise valid.

4 Implementation Result:

4.1 Frequent ItemSet generation results:

Below are frequent Itemset generated results of different lengths for various Value of MinSupport

Minsupport	len 1 FIS	len 2 FIS	len 3 FIS	len 4 FIS	len 5 FIS	len 6 FIS	len 7 FIS	len 8 FIS
30%	196	5340	5287	1518	438	88	11	1
40%	167	753	149	7	1	-	-	-
50%	109	63	2	-	-	-	-	-
60%	34	2	-	-	-	-	-	-
70%	7	-	-	-	-	-	-	-

Table 4.1-1 Count of Frequent Set Items(FIS) of differnet length with given Minsupport value

4.2 Association rule generation

4.2.1 Results for Template1:

Sample Template1 Queries	#Rules
(result11, cnt) = asso_rule.template1("RULE", "ANY", ['G59_Up'])	26
(result12, cnt) = asso_rule.template1("RULE", "NONE", ['G59_Up'])	91
(result13, cnt) = asso_rule.template1("RULE", 1, ['G59_Up', 'G10_Down'])	39
(result14, cnt) = asso_rule.template1("BODY", "ANY", ['G59_Up'])	9
(result15, cnt) = asso_rule.template1("BODY", "NONE", ['G59_Up'])	108
(result16, cnt) = asso_rule.template1("BODY", 1, ['G59_Up', 'G10_Down'])	17
(result17, cnt) = asso_rule.template1("HEAD", "ANY", ['G59_Up'])	17
(result18, cnt) = asso_rule.template1("HEAD", "NONE", ['G59_Up'])	100
(result19, cnt) = asso_rule.template1("HEAD", 1, ['G59_Up', 'G10_Down'])	24

Table 4.2-1 Association Rules count for same Template1 Queries with Support = 50% and Confidence = 70%

4.2.2 Results for Template2:

Sample Template2 Queries	#Rules
(result21, cnt) = asso_rule.template2("RULE", 3)	9
(result22, cnt) = asso_rule.template2("BODY", 2)	6
(result23, cnt) = asso_rule.template2("HEAD", 1)	117

Table 4.2-2 Association Rules count for same Template2 Queries with Support = 50% and Confidence = 70%

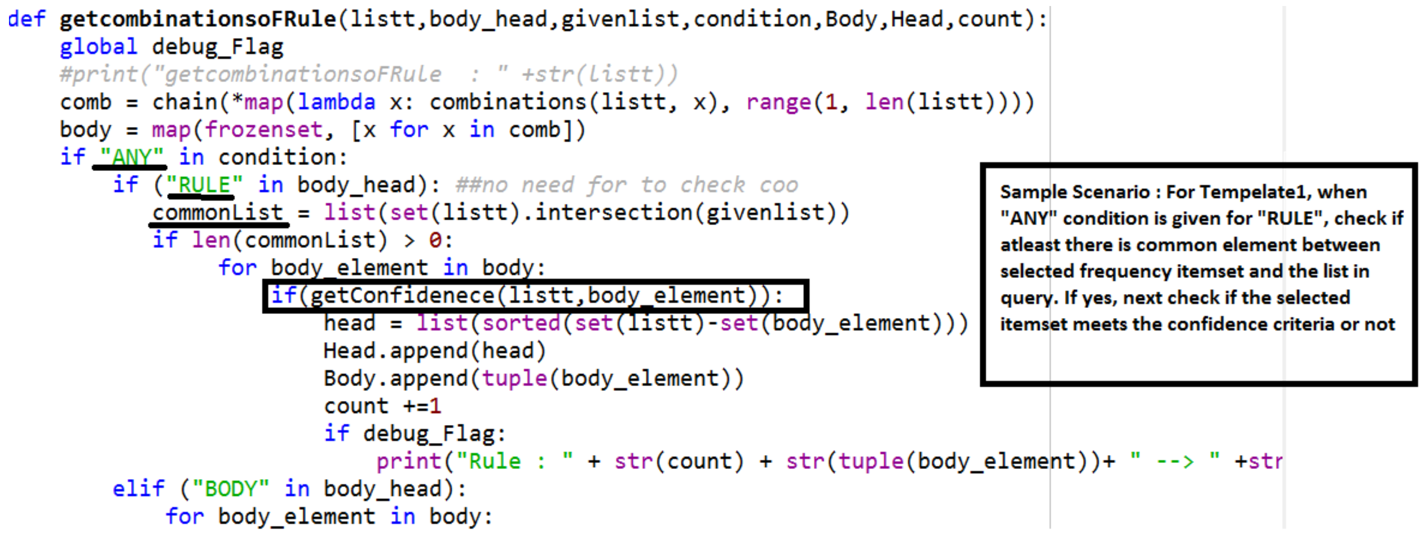
4.2.3 Results for Template3:

Sample Template3 Queries	#Rules
(result31, cnt) = asso_rule.template3("1or1", "BODY", "ANY", ['G10_Down'],"HEAD", 1, ['G59_Up'])	24
(result32, cnt) = asso_rule.template3("1and1", "BODY", "ANY",['G10_Down'], "HEAD", 1, ['G59_Up'])	1
(result33, cnt) = asso_rule.template3("1or2", "BODY", "ANY", ['G10_Down'],"HEAD", 2)	11
(result34, cnt) = asso_rule.template3("1and2", "BODY", "ANY",['G10_Down'], "HEAD", 2)	0
(result35, cnt) = asso_rule.template3("2or2", "BODY", 1, "HEAD", 2)	117
(result36, cnt) = asso_rule.template3("2and2", "BODY", 1, "HEAD", 2)	3

Table 4.2-3 Association Rules count for same Template3 Queries with Support = 50% and Confidence = 70%

5 Brief Code Overview

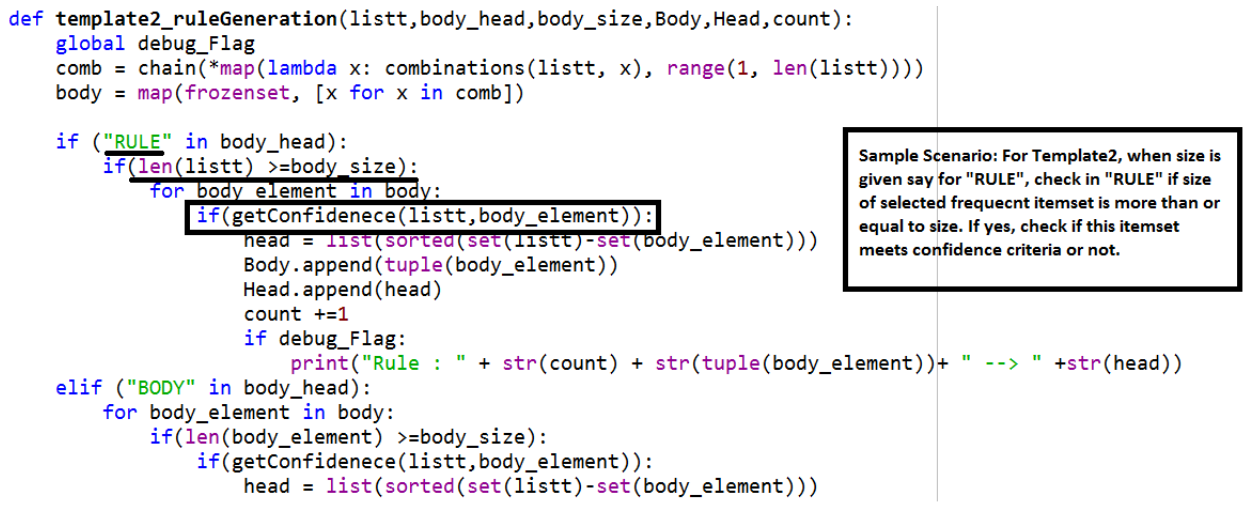
5.1 Handling Template1 Queries:



```
def getcombinationsoFRule(listt,body_head,givenlist,condition,Body,Head,count):
    global debug_Flag
    #print("getcombinationsoFRule  : " +str(listt))
    comb = chain(*map(lambda x: combinations(listt, x), range(1, len(listt))))
    body = map(frozenset, [x for x in comb])
    if "ANY" in condition:
        if ("RULE" in body_head): ##no need for to check coo
            commonList = list(set(listt).intersection(givenlist))
            if len(commonList) > 0:
                for body_element in body:
                    if(getConfidenece(listt,body_element)):
                        head = list(sorted(set(listt)-set(body_element)))
                        Head.append(head)
                        Body.append(tuple(body_element))
                        count +=1
                        if debug_Flag:
                            print("Rule : " + str(count) + str(tuple(body_element))+ " --> " +str
        elif ("BODY" in body_head):
            for body_element in body:
```

Sample Scenario : For Tempelate1, when "ANY" condition is given for "RULE", check if atleast there is common element between selected frequency itemset and the list in query. If yes, next check if the selected itemset meets the confidence criteria or not

5.2 Handling Template2 Queries

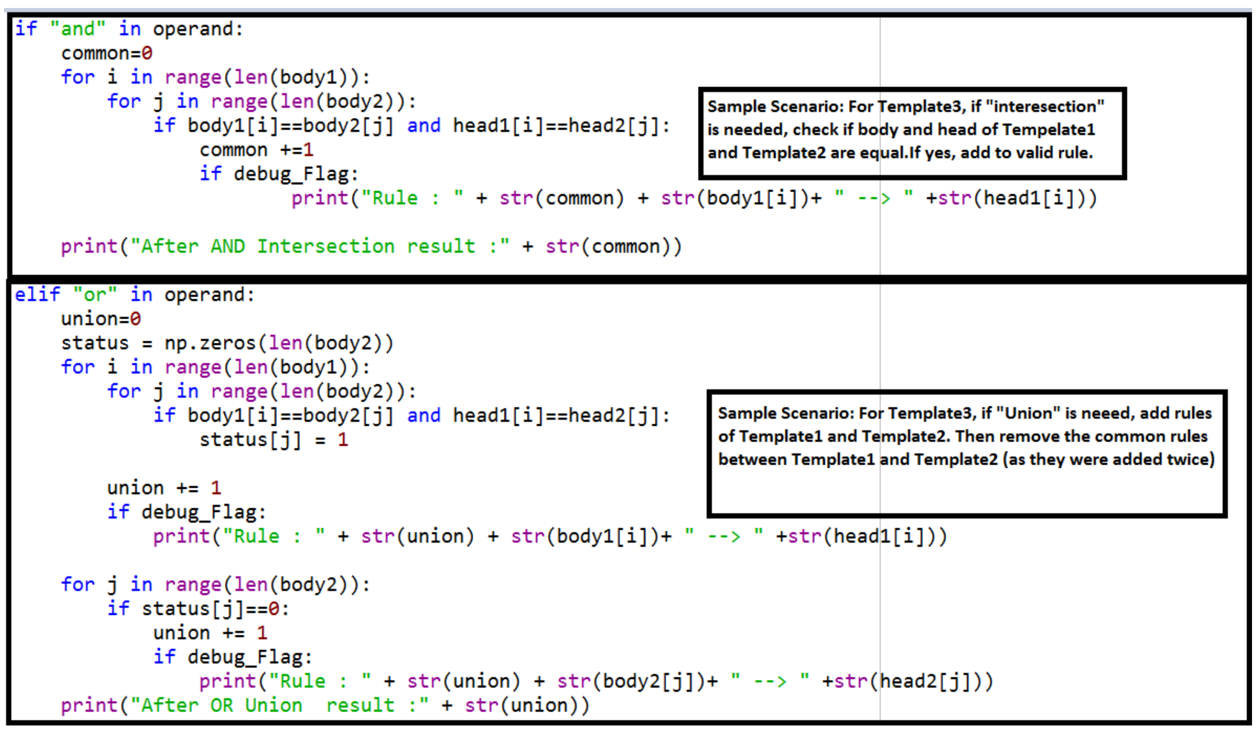


```
def template2_ruleGeneration(listt,body_head,body_size,Body,Head,count):
    global debug_Flag
    comb = chain(*map(lambda x: combinations(listt, x), range(1, len(listt))))
    body = map(frozenset, [x for x in comb])

    if ("RULE" in body_head):
        if(len(listt) >=body_size):
            for body_element in body:
                if(getConfidenece(listt,body_element)):
                    head = list(sorted(set(listt)-set(body_element)))
                    Body.append(tuple(body_element))
                    Head.append(head)
                    count +=1
                    if debug_Flag:
                        print("Rule : " + str(count) + str(tuple(body_element))+ " --> " +str(head))
    elif ("BODY" in body_head):
        for body_element in body:
            if(len(body_element) >=body_size):
                if(getConfidenece(listt,body_element)):
                    head = list(sorted(set(listt)-set(body_element)))
```

Sample Scenario: For Template2, when size is given say for "RULE", check in "RULE" if size of selected frequecnt itemset is more than or equal to size. If yes, check if this itemset meets confidence criteria or not.

5.3 Handling Template3 Queries



```
if "and" in operand:
    common=0
    for i in range(len(body1)):
        for j in range(len(body2)):
            if body1[i]==body2[j] and head1[i]==head2[j]:
                common +=1
                if debug_Flag:
                    print("Rule : " + str(common) + str(body1[i])+ " --> " +str(head1[i]))

    print("After AND Intersection result :" + str(common))
elif "or" in operand:
    union=0
    status = np.zeros(len(body2))
    for i in range(len(body1)):
        for j in range(len(body2)):
            if body1[i]==body2[j] and head1[i]==head2[j]:
                status[j] = 1

        union += 1
        if debug_Flag:
            print("Rule : " + str(union) + str(body1[i])+ " --> " +str(head1[i]))

    for j in range(len(body2)):
        if status[j]==0:
            union += 1
            if debug_Flag:
                print("Rule : " + str(union) + str(body2[j])+ " --> " +str(head2[j]))
    print("After OR Union  result :" + str(union))
```

Sample Scenario: For Template3, if "interesection" is needed, check if body and head of Tempelate1 and Template2 are equal.If yes, add to valid rule.

Sample Scenario: For Template3, if "Union" is neeed, add rules of Template1 and Template2. Then remove the common rules between Template1 and Template2 (as they were added twice)

6 References

https://en.wikipedia.org/wiki/Association_rule_learning

https://en.wikipedia.org/wiki/Apriori_algorithm