**CSE 601 – Data Mining and Bioinformatics**

**Project 1: Dimensionality Reduction & Association Analysis**

**Team members:**

Kiran Ketan Nevrekar, 50336915, kiranket

Revathy Narayanan, 50336857, revathyn

Saranya Saravanan, 50314931, saravan2

**Part 2: Association Analysis**

**Objective:** The aim of this part of the project is to use Apriori algorithm to generate sets of frequent itemset that have a support value greater than the given threshold support. Use the generated frequent itemsets to generate association rules when given a query of suggested template.

**Association Analysis:** Association Analysis is a method used to discover relationships between data that are hidden in very large datasets. The relationship mentioned can be delivered in the form of frequent itemsets and/or association rules that satisfy the given minimum threshold, both of which are implemented in this project. Useful applications are in the areas of market analysis, medical analysis, web mining etc.,

**Apriori Algorithm:** Apriori Algorithm is an iterative method where k itemsets are used to find k+1 frequent itemsets given a support threshold. The result is then used for association rule generation.

**Packages used:** Pandas, Numpy, Combinations from itertools

**Frequent Itemset generation and association rule generation**

**Implementation:**

1. Import the packages necessary for handling the data and generating the combinations
2. The data is loaded into a dataframe and preprocessed.
3. The dataset is converted to the given format by appending a “G” and gene number value and stored in the transaction list.
4. To start with the candidate set is generated for length K=1 initially.
5. The main function “mainfunction”contains the :
   1. Implementation for finding the frequent itemset with length K= 1
      1. for each row in the candidateset 1, iterate through the transaction data and find the count of transactions in which the candidate is present and divide the value by the length of transactions.
      2. If this value is greater or equal to the support threshold, then it is a frequent itemset of length 1.
   2. Function call to frequentset\_k():
   3. Function call to rules\_generarion():
6. Defining function frequentset\_k():
   1. By looping through the frequent itemset generated from the previous step, a set of candidate sets are generated.
   2. The subsequent frequent itemsets of length K are generated by satisfying the support threshold.
   3. This loop continues where no more candidate sets generation are possible.
   4. The generated frequent itemsets with their length are stored in a dictionary.
7. Defining the function rules\_generarion():
   1. The list of frequent itemsets is passed as input to the function.
   2. Start from the rule whose length is greater than 1.
   3. Using the combinations function generate all the rules where the corresponding items are present together
   4. The confidence value is computed by dividing this total value by the number of transactions where the item in the left hand side of the rule is present.
   5. Store the rules that satisfy the minimum confidence threshold.
   6. This list will output the list of rules that satisfies both the minimum support threshold and minimum confidence threshold.
   7. This output can be further used for querying based on the templates specified.

**Result:** Results for support 30%, 40%, 50%, 60% and 70% are as follows,

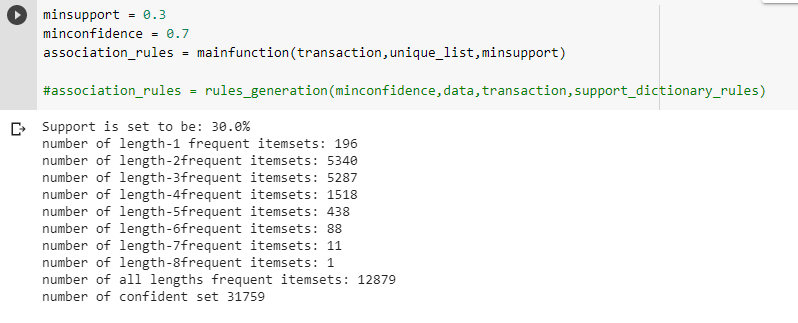


Figure:1 (support = 30%)

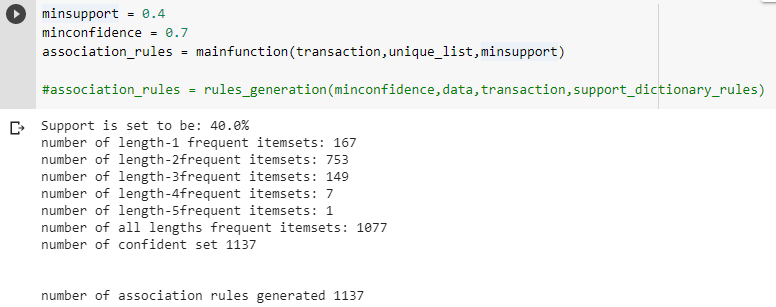


Figure:2 (support = 40%)

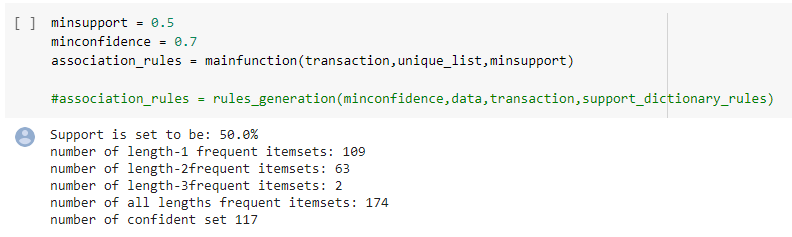


Figure:3 (support = 50%)

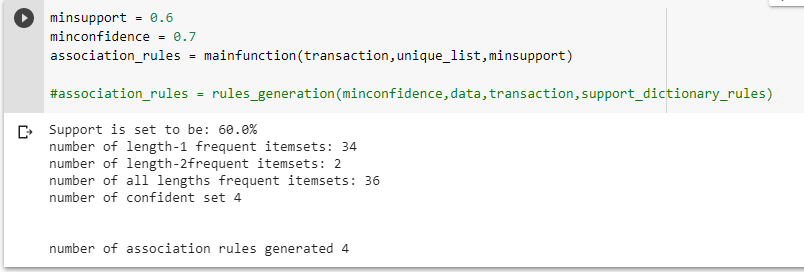


Figure:4 (support = 60%)

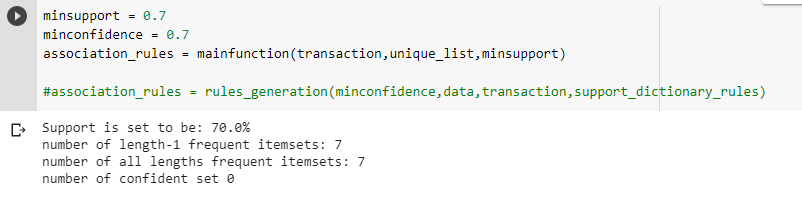


Figure:5 (support = 70%)

**Implementation of the templates:**

**Template 1:{RULE|HEAD|BODY} HAS ({ANY|NUMBER|NONE}) OF (ITEM1, ITEM2, ..., ITEMn)**

1. The function call\_template1() will give a prompt for the user to input the query.
2. call\_template1() calls the function template1() which takes parameters, association rules and query. And this returns the valid rules that satisfies the query conditions that are passed.
3. The query that is passed as the input is divided into three parts type1, type2 and query. Type1 will contain values like RULE|HEAD|BODY, type2 will contain values like ANY|NONE|NUMBER and query contains a set of items that are to be searched in the rule set.
4. ANY means if any of the items passed in the input is a part of the rule|head|body, NONE means none of the items passed in input are a part of the rule|head|body. NUMBER would be the exact number of times the items occur in the rule|head|body.
5. As per this combination of the input passed, it checks for this combination in the rules generated. Everytime a rule from the ruleset matches, it increments a variable called ‘confidence\_ct’ by 1. That corresponding rule from the ruleset is then appended to ‘output\_rules’. Thus once the entire ruleset is traversed for the combination input, confidence\_ct will contain the value of the number of rules that are matching and valid and output\_rules will contain all the valid rules. Thus the output of template1 is the count of the valid rules and output\_rules which is a set of the valid rules.

**Template 2:SizeOf({HEAD|BODY|RULE}) ≥ NUMBER**

1. The function call\_template2() will give a prompt for the user to input the query.
2. call\_template2() also calls the function template2() which in turn stores the output in a variable named ‘output’. Also returns the rules generated and their count.
3. template2() function takes the input from call\_template2(). This function has variable count which increments according to the loop it enters.
4. Template 2 has three combinations totally, each case is directly handled by if and else set of statements.
5. The RULE, HEAD or BODY is split and stored in a list accordingly and increment the counter and append the list to a final output list if its length is greater than or equal to the number given in the query input.
6. For any query(RULE|BODY|HEAD, size), return all the rules with (RULE|HEAD|BODY)'s size greater than or equal to the given size and increments the count value which is then passed back to call-template3() function to print the final output.

**Template 3: Any combined templates using AND or OR. For**

1. The function call\_template3() will give a prompt for the user to input the query.
2. call\_template3() also calls the function template3() and stores the output in a variable named ‘output’. Also returns the rules generated and their count.
3. The template 3 has 6 combinations totally, where each case is handled by an if statement.
4. The major query is split with “|” as delimiter and the list at index[0]

is again split using “,” as a delimiter to get the operator and the choice of two templates.

1. Call the function for template 1 or template 2 with respective query based on the choice of first template entered by the user. Perform this same step for the choice of the second template as well.
2. Now based on the operator given by the user set operation “and” or “or” is performed.
3. The result count and Association rule list from here is stored to a variable and a list respectively and outputted.

**Results:**

**Template 1:**

**1. RULE;ANY;G59\_Up**

Count of the query returned: 26

**2. RULE;NONE;G59\_Up**

Count of the query returned: 91

**3. RULE;1;G59\_Up,G10\_Down**

Count of the query returned: 39

**4. HEAD;ANY;G59\_Up**

Count of the query returned: 17

**5. HEAD;NONE;G59\_Up**

Count of the query returned: 100

**6. HEAD;1;G59\_Up,G10\_Down**

Count of the query returned: 24

**7. BODY;ANY;G59\_Up**

Count of the query returned: 9

**8. BODY;NONE;G59\_Up**

Count of the query returned: 108

**9. BODY;1;G59\_Up,G10\_Down**

Count of the query returned: 17

**Template 2:**

**1. RULE;3**

Count of the query returned: 9

**2. HEAD;2**

Count of the query returned: 3

**3. BODY;1**

Count of the query returned: 117

**Template 3:**

**1. 1,or,1|HEAD;ANY;G10\_Down|BODY;1;G59\_Up**

Count of the query returned: 16

**2. 1,and,1|HEAD;ANY;G10\_Down|BODY;1;G59\_Up**

Count of the query returned: 0

**3. 1,or,2|HEAD;ANY;G10\_Down|BODY;2**

Count of the query returned: 13

**4. 1,and,2|HEAD;ANY;G10\_Down|BODY;2**

Count of the query returned: 0

**5. 2,or,2|HEAD;1|BODY;2**

Count of the query returned: 117

**6. 2,and,2|HEAD;1|BODY;2**

Count of the query returned: 6