DATA MINING

ASSIGNMENT-2: MINING ASSOCIATION RULES FROM GENE EXPRESSION DATA

TEAM MEMBERS:

Srikanth Madduri Venkata	Arjun Sundaresh	Sridhar Vadlamani
Ssmaddur	arjunsun	sridharv
50134851	50169917	50168092

Implementation of Apriori Algorithm

Our implementation of the Apriori algorithm is done basically by following steps,

1. Generate frequent item set of length and filter them based on the minimum support value. Generate combinations of length > 1 one at a time and filter them at each iteration.
2. Carry on until the next iteration results in a frequent item set of size zero.

The implementation of Apriori algorithm was done over four important classes,

1. "ReadFile" Class:
   - ➢ The data is read from the file 'gene_expression.txt' and is saved in a 2-dimensional array.
   - ➢ The data is organized in such a way that it can be processed easily and this is done by changing all the cells to '0' where the value is Down and to '1' where it is UP. Similarly, the diseases 'Breast', 'Colon', 'ALL' and 'AML' are changed to 2,3,4,5 respectively.
   - ➢ The organized data is now returned.

2. "FrequentItemList "Class:
   - ➢ The main job of this class is to find the frequent items according to the given support.
   - ➢ There are two methods.

   - ➢ In the method "<u>genLengthOneFrequentItemSet</u>" we can find the list of frequent items of length one.
   - ➢ No. of up's and down's or 0's and 1's is counted vertically for a particular gene or group across all the samples and the support of up and down of each gene is calculated.
   - ➢ If the gene is found to be a frequent item, it is put in a List of strings.
   - ➢ After traversing the entire 2-dimensional array of data, we will be left with a list of strings that contains the frequent items and this list is returned.

   - ➢ In the second method, <u>getCountForListLengthN"</u> we can find the list of frequent items of length more than one.
   - ➢ In this method, we get the distinct possible combinations of itemsets from the third class i.e. generate combination.
   - ➢ We calculate the support of each and every possible combination of itemset by checking across all the samples and if it is greater than the minimum support, we classify it as frequent item and it to a list of frequent items.
   - ➢ This list of frequent items is then returned.

3. "GenerateCombination" Class:
   - ➢ In this class, we find the distinct possible combination of itemsets from frequent items of length one. The combinations we find can be of length starting from two and we will stop only if the frequent itemsets of a particular length are zero.
   - ➢ The distinct combinations of particular length are calculated from the frequent items sets of its previous length and length one.

i.e for example, if we need to generate the distinct combinations for length 4, we consider the list frequent items of length 3 and length 1.

- Thus we are eliminating the definite non-frequent combinations without even calculating the support for those combinations

4. “MainClass” Class:
   - In this class we simply create instances for each class and do the required operations.
   - We read the data from the “ReadFile” Class and pass it to “FrequentItemList “Class, where we calculate the frequent itemset for length one and return it.
   - Now we pass the results of the frequent itemset of length one to “GenerateCombination” Class so that we can find the unique combinations of itemsets for a particular length.
   - The results of this class are again passed to “FrequentItemList “Class where calculate the frequent item list of particular length more than one. The results are returned to “Main Class” where they are printed.

Implementation of Association Rule Generation

- We have designed the rule generation process in the class ‘Generate Association Rule’.
- We have maintained a Hash Map called ‘itemSetCountMap’ which stores the count of every frequent item set that is generated for the given minimum support value.
- Power sets of each of the present frequent item sets are generated and associations between them are generated.
  For example: The frequent item set (A, B, C) will generate the power set (A, B,C,AB,AC,BC,ABC). Associations are generated between items (A=>BC, B=>AC,C=>AB). The associations for (A=>B, A=>C, B=>A …) are not done at this point. They would be generated for the frequent item set (A, C) and (A, B).
- These associations are filtered using the given minimum confidence value and stored in the associations map.

Template For Queries

- **Template1**
  - The function ‘templateOne’ does a simple search for the rules though the HashMap of rules that has already been generated based on the given constraints.
- **Template 2**
  - The function ‘templateTwo’ does a count of the size of the specified part (HEAD/BODY/RULE) and checks it against the given constraint.

- **Template 3**
  - This function is used to run “AND” and “OR” logical operations on two specified queries.
  - The given two queries are run using either/both of the other two templates and runs the results though this function to return the results;

RESULTS

Results for Requirement-1:

Support – 30%:

Frequent Itemsets of Length	1	:	196
Frequent Itemsets of Length	2	:	5393
Frequent Itemsets of Length	3	:	5752
Frequent Itemsets of Length	4	:	1746
Frequent Itemsets of Length	5	:	480
Frequent Itemsets of Length	6	:	71
Frequent Itemsets of Length	7	:	3
Frequent Itemsets of Length	8	:	0
Total Frequent Itemsets		:	13641

Support – 40%:

Frequent Itemsets of Length	1	:	167
Frequent Itemsets of Length	2	:	790
Frequent Itemsets of Length	3	:	167
Frequent Itemsets of Length	4	:	7
Frequent Itemsets of Length	5	:	1
Frequent Itemsets of Length	6	:	0
Total Frequent Itemsets		:	1132

Support – 50%:

➢ Frequent Itemsets of Length	1	:	109
➢ Frequent Itemsets of Length	2	:	63
➢ Frequent Itemsets of Length	3	:	2
➢ Frequent Itemsets of Length	4	:	0
➢ Total Frequent Itemsets		:	174

Support – 60%:

➢ Frequent Itemsets of Length	1	:	34
➢ Frequent Itemsets of Length	2	:	2
➢ Frequent Itemsets of Length	3	:	0
➢ Total Frequent Itemsets		:	36

Support – 70%:

➢ Frequent Itemsets of Length	1	:	7
➢ Frequent Itemsets of Length	2	:	0
➢ Total Frequent Itemsets		:	7

Results for Requirement-2:

Part A (16 easy queries)

TEMPLATE 1:

1. RULE HAS ANY OF G6_UP	:	10
2. RULE HAS 1 OF G1_UP	:	14
3. RULE HAS 1 OF (G1_UP, G10_DOWN)	:	26
4. BODY HAS ANY OF G6_UP	:	5
5. BODY HAS NONE OF G72_UP	:	124
6. BODY HAS 1 OF (G1_UP, G10_DOWN)	:	15
7. HEAD HAS ANY OF G6_UP	:	5

8. HEAD HAS NONE OF (G1_UP, G6_UP)	126
9. HEAD HAS 1 OF (G6_UP, G8_UP)	6
10. RULE HAS 1 OF (G1_UP, G6_UP, G72_UP)	48
11. RULE HAS ANY OF (G1_UP, G6_UP, G72_UP)	50

<u>TEMPLATE 2:</u>

1. SIZE OF RULE >= 3	12
2. SIZE OF BODY >= 2	6
3. SIZE OF HEAD >= 2	6

<u>TEMPLATE 3:</u>

1. BODY HAS ANY OF G1_UP AND HEAD HAS 1 OF G59_UP	1
2. BODY HAS ANY OF G1_UP OR HEAD HAS 1 OF G6_UP	12
3. BODY HAS 1 OF G1_UP OR HEAD HAS 2 OF G6_UP	7
4. HEAD HAS 1 OF G1_UP AND BODY HAS 0 OF DISEASE	7
5. HEAD HAS 1 OF DISEASE OR RULE HAS 1 OF (G72_UP, G96_DOWN)	24
6. BODY HAS 1 of (G59_UP, G96_DOWN) AND SIZE OF RULE >=3	7