Data Mining and Bioinformatics

Association Report

Saurabh Bajoria 50208005
Vidhi Shah 50207090
Sucheta Mulange 50206855

Apriori Algorithm:

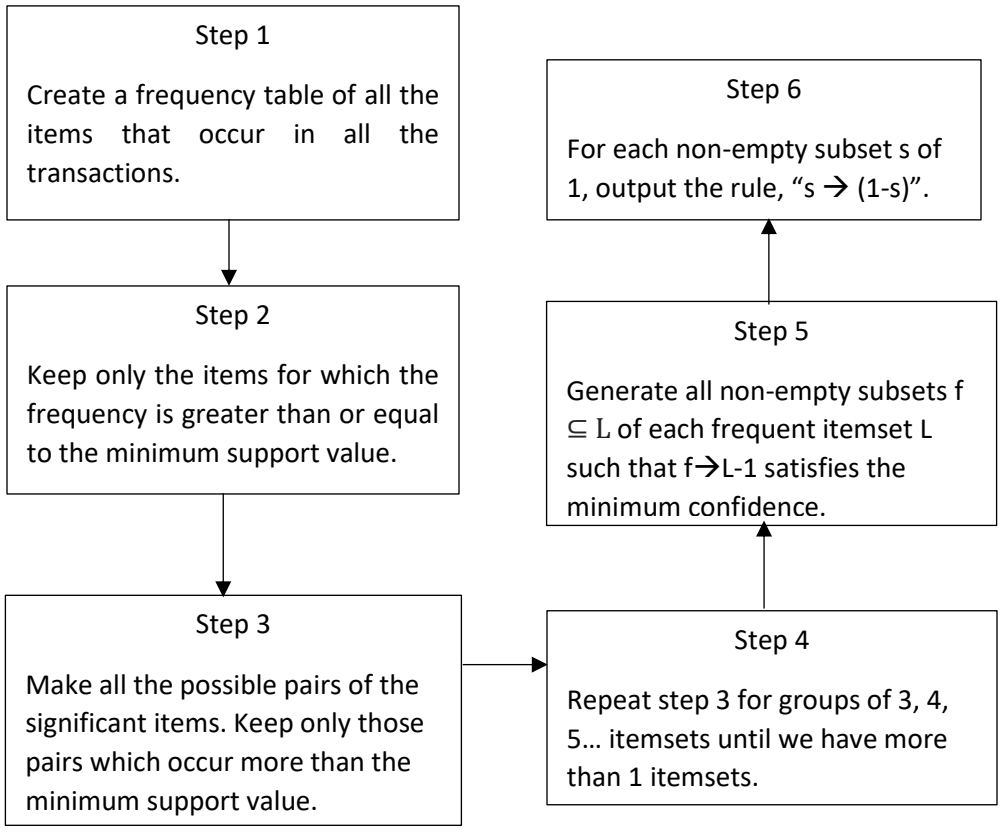
Apriori is an algorithm for frequent item set mining and association rule learning over transactional databases.

It uses a "bottom up" approach, where frequent subsets are extended one item at a time and groups of candidates are tested against the data. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database.

Terminology:

a) **Frequent Itemsets:** The sets of item which has minimum support (denoted by Li for ith-Itemset).
b) **Apriority Property:** Any subset of frequent itemset must be frequent.
c) **Join Operation:** To find L_k a set of candidate k-itemsets is generated by joining L_{k-1} with itself.

Flow of Association rule generation:



If for a frequent itemset L,|L|=k then there are 2k-1 candidate association rules.

Code Explanation:

- ## Classes:

1. **Apriori:** Public class that controls the flow of the program. Has static method main that initiates the execution of the code.

2. **Rules:** Stores the schema for all the rules. Has 3 lists, head, body and rule for storing the respective lists.

- ## Methods:

1. **length1Itemset**
   Input Parameters: ArrayList<ArrayList<String>> *data*, int *support*
   Return type: ArrayList<ArrayList<String>>

This method takes in the data read from the associationruletestdata.txt file as an ArrayList of String and support value taken from the user at runtime.
Each item in each line of the list is pushed into a HashMap of String as the key and a List as the value. If the string read is “Up”, a 1 is added to the list and if it is read “Down” then a 0 is added to the list. For all the diseases present at the end of each line, 1,2,3 and so on is added to the same list.
If the count of all the 1’s in a list is greater than the support value, then that Gene is qualified for the frequent itemset as Gene_Up otherwise it is discarded. Similarly, if the count of 0’s in a list is greater than the support value then that Gene is qualified as Gene_Down.
The qualified genes and diseases set is added to an ArrayList *freqCount* and is returned.

2. **subsequentItemsets**
   Input Parameters: ArrayList<ArrayList<String>> *first*, ArrayList<ArrayList<String>> *data*, int *support*
   Return type: ArrayList<ArrayList<String>>

This method takes in the itemset generated by the above method, data read from the association file and the support value.
It generates subsequent frequent itemsets list based on the previously generated list, i.e. it generates itemsets of length k from k-1 length frequent itemsets.
This method is called, until no frequent itemsets are found.

3. **subsets**
   Input Parameters: ArrayList<String> *list*
   Return type: ArrayList<ArrayList<String>>

This method takes in the frequent itemsets list as the input and generates all possible subsets. These subsets are used to generate association rules.

4. **template1**
   Input Parameters: String *part*, int *i*, String *itemlist*, ArrayList<Rules> *rule*
   Return Type: ArrayList<String>

This method takes in 4 inputs, *part* which states which part of the rule needs to be queried, *itemlist* is the gene set which need to be checked, i is the occurrence count and *rule* is the ArrayList which has all the rules.
If the part is equal to "BODY" then the body part of each rule is checked for all the genes present in the itemlist. If the count of occurrence is equal to the value i, then that rule is added to the queryResult list.
Similarly, for part= "RULE" and "HEAD".

5. **template2**
   Input Parameters: String *part*, int i, ArrayList<Rules> *rule*
   Return type: ArrayList<String>

This method takes in 3 inputs, *part* which states which part of the rule needs to be queried, i is the occurrence count and *rule* is the ArrayList which has all the rules.
If the part is equal to "BODY" then the body part of each rule is checked for the number of genes. If that count is greater than or equal to i, then that rule is added to the queryResult list.
Similarly, for part="RULE" and "HEAD".

6. **template3**
   Input Parameters: String *s*, ArrayList<Rules> *rule*
   Return type: ArrayList<String>

This method takes in the whole query as a string *s* and splits it using " "(space).
If the query is an OR of 1 and 1, then *template1* is called twice and the lists obtained are ORed and returned. Similarly, if the query is an AND of 1 and 2, then *template1* and *template2* are called once each, and the lists obtained are ANDed and returned. Similarly, for all the other cases.

7. **main**

This method reads the data from a file "association.txt" into an ArrayList<ArrayList<String>> *data*.
It also asks for confidence and support values from the user.
Frequent itemset of length1 is generated using *length1Itemset* and all the subsequent itemsets are generated using *subsequentItemsets*.
For all the Frequent itemsets, rules are generated using *subsets* and added to an ArrayList of type Rules.
Rules is a class that has 3 Lists in it, Body, Head and Rule, each maintaining respective lists.
After all the rules are generated, user is asked for queries. User can input queries using any format out of template1, template2 or template3. The user can also end the execution of the program using "EXIT".

Results:

Task 1:

1) Support= 70%

number of length-1 frequent itemsets: 7
number of length-2 frequent itemsets: 0
number of all lengths frequent itemsets: 7

2) Support= 60%

number of length-1 frequent itemsets: 34
number of length-2 frequent itemsets: 2
number of length-3 frequent itemsets: 0
number of all lengths frequent itemsets: 36

3) Support= 50%

number of length-1 frequent itemsets: 109
number of length-2 frequent itemsets: 63
number of length-3 frequent itemsets: 2
number of length-4 frequent itemsets: 0
number of all lengths frequent itemsets:174

4) Support= 40%

number of length-1 frequent itemsets: 167
number of length-2 frequent itemsets: 753
number of length-3 frequent itemsets: 149
number of length-4 frequent itemsets: 7
number of length-5 frequent itemsets: 1
number of all lengths frequent itemsets: 1077

5) Support= 30%

number of length-1 frequent itemsets: 196
number of length-2 frequent itemsets: 5340
number of length-3 frequent itemsets: 5287
number of length-4 frequent itemsets: 1518
number of length-5 frequent itemsets: 438
number of length-6 frequent itemsets: 88
number of length-7 frequent itemsets: 11
number of length-8 frequent itemsets: 1
number of all lengths frequent itemsets: 12879

Task 2:

Template 1:

Support= 50%, Confidence= 70%

a) ("RULE", "ANY", ['G59_Up'])

Rule Count: 26

b) ("RULE", "NONE", ['G59_Up'])

Rule Count: 91

c) ("RULE", 1, ['G59_Up', 'G10_Down'])

Rule Count: 39

d) ("BODY", "ANY", ['G59_Up'])

Rule Count: 9

e) ("BODY", "NONE", ['G59_Up'])

Rule Count: 108

f) ("BODY", 1, ['G59_Up', 'G10_Down'])

Rule Count: 17

g) ("HEAD", "ANY", ['G59_Up'])

Rule Count: 17

h) ("HEAD", "NONE", ['G59_Up'])

Rule Count: 100

i) ("HEAD", 1, ['G59_Up', 'G10_Down'])

Rule Count: 24

Template 2:

Support= 50%, Confidence= 70%

a) ("RULE", 3)

Rule Count: 9

b) ("BODY", 2)

Rule Count: 6

c) ("HEAD", 1)

Rule Count: 117

Template 3:

Support= 50%, Confidence= 70%

a) ("1or1", "BODY", "ANY", ['G10_Down'], "HEAD", 1, ['G59_Up'])

Rule Count: 24

b) ("1and1", "BODY", "ANY", ['G10_Down'], "HEAD", 1, ['G59_Up'])

Rule Count: 1

c) ("1or2", "BODY", "ANY", ['G10_Down'], "HEAD", 2)

Rule Count: 11

d) ("1and2", "BODY", "ANY", ['G10_Down'], "HEAD", 2)

Rule Count: 0

e) ("2or2", "BODY", 1, "HEAD", 2)

Rule Count: 117

f) ("2and2", "BODY", 1, "HEAD", 2)

Rule Count: 3

References:

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