CSE 601 Data Mining

HomeWork 2

Mining Association Rules From Gene Expression Data

Amit Nihalani (50133954)
Harsh Harwani (50134630)
Siddharth Ghodke (50134394)

Apriori Algorithm:

- In the first step of the Apriori Algorithm a list of features of size 1 with their respective count in the data is generated.
- This frequent item set of size 1 is passed to the apriori algorithm and apriori algorithm computes the frequent itemsets and their count of various sizes till the count of frequent itemset becomes zero.
- In every step of the apriori algorithm a candidate set C is generated from L. for example A candidate set C2 is generated by joining all the results from L1 to L1. For every join result we also compute all the subsets and check whether those subsets belong to L1 , this is the use of the apriori property if the subset is infrequent its superset will also be infrequent so we prune such results and return C2.
- Now the support count for the pruned result is calculated and is compared against the minimum support which generates L2.
- Again L2 is passed and C3 is produced using the same process. This process goes on till the size of L set is not zero.
- At Every complete iteration of the algorithm we get frequent itemset of size k+1.

Time Complexity:

- In order to find frequent itemsets using the brute force approach we would have to generate 2^d possible subsets
- Considering M candidates and N transactions the complexity would be o(NM) which is very expensive since M=2^d.
- Now using Apriori principle we reduce the number of candidates set in every iteration by pruning the candidates which do not satisfy the apriori property.
- The size of N decreases every iteration as the number of frequent itemsets increases.
- Also we store the candidates in a efficient data structure such as HashMap which makes searching the candidates and maintaining their count efficient.
- Now in case of apriori suppose the input transaction is N and candidate is R,the complexity of size i would be O(R^i) and time for calculating support

can be done in O(n). So the time complexity would be O[(R + N) + (R^2 + N) + (R^3 + N) ...] = O[MN + (R^1+R^2+ ... R^M)] = O(MN+ (1-R^M)/(1-R))

Implementation Details:

- In the implementation we had three goals in mind:
  - Reducing the candidate itemsets by using pruning techniques using the apriori property.
  - Reduce the number of transactions.
  - Reduce the number of comparisons to generate the frequent item datasets.
- First step was to create a list of 204 features,2(Up and Down) each for first 100 features and 4 features for last row of disease.
- we used a List<Map<Set<String>,Integer>> to store the datalist , the use of map was made in order to retrieve sets and their counts efficiently.
- Second step was to compute and compare the count for each feature and keep features with count>=support value and keep repeating till feature list will be zero.
- Now for every step of the apriori candidate set is generated ,which is done by making a self join of L set, this function is upper bounded by complexity O(n^2).
- Once a candidate set is produced, the candidate set is scanned and subsets are produced the subsets take O(2^n) time . We prune the results using apriori property if any of the subset is not present in L we remove that from C.
- Now again we generate L from C by counting the support of C and removing any set that has support less than minimum support. As the datalist is stored in a Map we can get through the map and check if set is present which takes O(n) time.
- Once the frequent item list is generated we find the frequent associations which is done by creating a power set of every item in the frequent item list. Complexity of this step is O(2^n)
- Now we go through every subset of the every frequent item set and find support(subset)/support(Lset) >= confidence we add the rule s->(l-s). The

overall time complexity of this step is O(2^n*n), but as the number of n is small in this step the code runs efficiently.

- Once the association rules are formed, we return a List<String> which contains the associations(For example Gene10_Up->Gene1_Down)
- We pass this List<String> to the function that handle template queries.

Results:

Part-1:

Confidence>=70 for all the results

Frequent ItemSet Count	Support>= 30	Support>= 40	Support>= 50	Support>= 60	Support>= 70
1	195	167	109	34	7
2	5330	753	63	2	0
3	5272	149	2	0	0
4	1498	7	0	0	0
5	423	1	0	0	0
6	82	0	0	0	0
7	10	0	0	0	0
8	1	0	0	0	0
9	0	0	0	0	0
Total size	12811	1077	174	36	7
Association Count	30830	1137	117	4	0
Execution	23.159	1.411	0.363	0.124e	0.72

Time(seconds)					

Part-2:

The number of association rules for support>=50% and confidence>=70 is **117**
The code prints the association rules for support>=50% and confidence>=70.

Part-3:

Template queries and their counts are as follows:

Queries	Count
RULE HAS (ANY) of (Gene1_UP)	13
RULE HAS (NONE) of (Gene1_UP)	104
RULE HAS (1) of (Gene1_UP,Gene10_DOWN)	24
BODY HAS (ANY) of (Gene1_UP)	7
BODY HAS (NONE) of (Gene1_UP)	110
BODY HAS (1) of (Gene1_UP,Gene10_DOWN)	15
HEAD HAS (ANY) of (Gene1_UP)	6
HEAD HAS (NONE) of (Gene1_UP)	111

HEAD HAS (1) of (Gene1_UP,Gene10_DOWN)	13
SizeOf(RULE) >= 2	117
SizeOf(BODY) >=2	6
SizeOf(HEAD) >= 2	3
BODY HAS (ANY) of (Gene1_UP) OR HEAD HAS (1) of (Gene59_UP)	23
BODY HAS (ANY) of (Gene1_UP) AND HEAD HAS (1) of (Gene59_UP)	1
BODY HAS (ANY) of (Gene1_UP) OR SizeOf(HEAD) >= 2	10
BODY HAS (ANY) of (Gene1_UP) AND SizeOf(HEAD) >= 2	0
SizeOf(BODY) >= 1 OR SizeOf(HEAD) >= 2	117
SizeOf(BODY) >= 1 AND SizeOf(HEAD) >= 2	3

Conclusion: we have generated the frequent item sets using the apriori algorithm and tried to implement it efficiently. We have made the code as readable and efficient by breaking the into various classes and printing out the results.

We are including a readme.txt with our code which would help you test our code.