Experiment No. 7

**Title :** Implementation of Apriori Association Rule Mining Algorithm. (C,C++,Java)

**Theory :**

**Association Rule:**

An implication expression of the form X→Y, where X and Y are itemsets

Example: {Milk, Diaper}→{Beer}

**Rule Evaluation Metrics:**

Support (s): Fraction of transactions that contain both X and Y

Confidence (c) : Measures how often items in Y appear in transactions that contain X

**Mining Association Rules:**

Two-step approach:

* **Frequent Itemset Generation :** Generate all itemsets whose support ≥ minsup
* **Rule Generation :** Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset . Frequent itemset generation is still computationally expensive

**Algorithm:-**

**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
* Count the support of each candidate by scanning the DB
* Eliminate candidates that are infrequent, leaving only those that are frequent

**Frequent itemset generation:**

Scan D and count each itemset in Ck, if the count is greater than minSupp, then add that itemset to Lk.

**Candidate itemset generation:**

For k = 1, C1 = all itemsets of length = 1.

**For k > 1, generate Ck from Lk-1 as follows:**

**The join step:**

Ck= k-2 way join of Lk-1 with itself.

if both {a1,..,ak-2, ak-1} & {a1,.., ak-2, ak} are in Lk-1, then add {a1,..,ak-2, ak-1, ak} to Ck.

The items are always stored in the sorted order.

**The prune step:**

Remove {a1, ...,ak-2, ak-1, ak}, if it contains a non-frequent (k-1) subset.

**Rule Generation**

Given a frequent itemset L, find all non-empty subsets f ⊂L such that f→L – f satisfies the

minimum confidence requirement

– If {A,B,C,D} is a frequent itemset, candidate rules:

ABC→D, ABD→C, ACD→ B, BCD→A,

A→BCD, B→ACD, C→ABD, D→ABC

AB →CD, AC→ BD, AD→BC, BC→AD,

BD→AC, CD→AB,

If |L| = k, then there are 2k–2 candidate association rules (ignoring L→Øand Ø→L)

Because, rules are generated from frequent itemsets, they automatically satisfy the minimum

support threshold

**Program:**

import java.io.\*;

class Apriori

{

public static void main(String []arg)throws IOException

{

int i,j,m=0;

int t1=0;

BufferedReader b=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the number of transaction :");

int n=Integer.parseInt(b.readLine());

System.out.println("items :1--Milk 2--Bread 3--Coffee 4--Juice 5--Cookies 6--Jam");

int item[][]=new int[n][6];

for(i=0;i<n;i++)

for(j=0;j<6;j++)

item[i][j]=0;

String[] itemlist={"MILK","BREAD","COFFEE","JUICE","COOKIES","JAM"};

int nt[]=new int[6];

int q[]=new int[6];

for(i=0;i<n;i++)

{ System.out.println("Transaction "+(i+1)+" :");

for(j=0;j<6;j++)

{ //System.out.println(itemlist[j]);

System.out.println("Is Item "+itemlist[j]+" present in this transaction(1/0)? :");

item[i][j]=Integer.parseInt(b.readLine());

} }

for(j=0;j<6;j++)

{ for(i=0;i<n;i++)

{if(item[i][j]==1)

nt[j]=nt[j]+1;

}

System.out.println("Number of Item "+itemlist[j]+" :"+nt[j]);

}

for(j=0;j<6;j++)

{ if(((nt[j]/(float)n)\*100)>=50)

q[j]=1;

else

q[j]=0;

if(q[j]==1)

{t1++;

System.out.println("Item "+itemlist[j]+" is selected ");

} }

for(j=0;j<6;j++)

{ for(i=0;i<n;i++)

{

if(q[j]==0)

{

item[i][j]=0;

} } }

int nt1[][]=new int[6][6];

for(j=0;j<6;j++)

{ for(m=j+1;m<6;m++)

{ for(i=0;i<n;i++)

{ if(item[i][j]==1 &&item[i][m]==1)

{ nt1[j][m]=nt1[j][m]+1;

} }

if(nt1[j][m]!=0)

System.out.println("Number of Items of "+itemlist[j]+"& "+itemlist[m]+" :"+nt1[j][m]);

} }

for(j=0;j<6;j++)

{ for(m=j+1;m<6;m++)

{

if(((nt1[j][m]/(float)n)\*100)>=50)

q[j]=1;

else

q[j]=0;

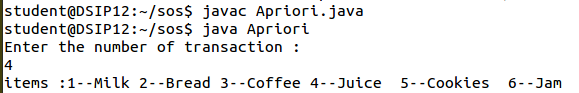
if(q[j]==1)

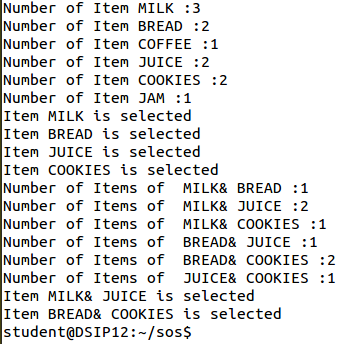
{

System.out.println("Item "+itemlist[j]+"& "+itemlist[m]+" is selected ");

} } } } }

**Output:**





**Conclusion:**

Apriori Association Rule Mining Algorithm is implemented successfully.