**Experiment No. 9**

**Aim:** Implementation of Association Rule Mining algorithm (Apriori).

**Theory:**

Association rule mining is a technique to identify

underlying relations between different items. There are many

methods to perform association rule mining. The Apriori algorithm

is the most simple and straightforward approach. However, since

it is the fundamental method, there are many different

improvements that can be applied to it.

**Components of Apriori algorithm**

The given three components comprise the Apriori algorithm.

1. Support
2. Confidence
3. Lift

Let us take an example to understand this concept.

We have already discussed above; you need a huge database containing a large no of transactions. Suppose you have 4000 customers transactions in a Big Bazar. You must calculate the Support, Confidence, and Lift for two products, and you may say Biscuits and Chocolate. This is because customers frequently buy these two items together.

Out of 4000 transactions, 400 contain Biscuits, whereas 600 contain Chocolate, and these 600 transactions include a 200 that includes Biscuits and chocolates. Using this data, we will find out the support, confidence, and lift.

**Support**

Support refers to the default popularity of any product. You find the support as a quotient of the division of the number of transactions comprising that product by the total number of transactions. Hence, we get

**Support (Biscuits) = (Transactions relating biscuits) / (Total transactions)**

= 400/4000 = 10 percent.

**Confidence**

Confidence refers to the possibility that the customers bought both biscuits and chocolates together. So, you need to divide the number of transactions that comprise both biscuits and chocolates by the total number of transactions to get the confidence.

Hence,

**Confidence = (Transactions relating both biscuits and Chocolate) / (Total transactions involving Biscuits)**

= 200/400

= 50 percent.

It means that 50 percent of customers who bought biscuits bought chocolates also.

**Advantages of Apriori Algorithm**

It is used to calculate large itemsets.

Simple to understand and apply.

**Disadvantages of Apriori Algorithm**

Apriori algorithm is an expensive method to find support since the calculation has to pass through the whole database.

Sometimes, you need a huge number of candidate rules, so it becomes computationally more expensive.

**Code:**

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.HashMap;

import java.util.Map;

public class Apriori {

public static void main(String[] args) {

List < String[] > data = new ArrayList < > ();

data.add(new String[] {

"T100",

"I1",

"I2",

"I5"

});

data.add(new String[] {

"T200",

"I2",

"I4"

});

data.add(new String[] {

"T300",

"I2",

"I3"

});

data.add(new String[] {

"T400",

"I1",

"I2",

"I4"

});

data.add(new String[] {

"T500",

"I1",

"I3"

});

data.add(new String[] {

"T600",

"I2",

"I3"

});

data.add(new String[] {

"T700",

"I1",

"I3"

});

data.add(new String[] {

"T800",

"I1",

"I2",

"I3",

"I5"

});

data.add(new String[] {

"T900",

"I1",

"I2",

"I3"

});

List < String > init = new ArrayList < > ();

for (String[] itemSet: data) {

for (int i = 1; i < itemSet.length; i++) {

if (!init.contains(itemSet[i])) {

init.add(itemSet[i]);

}

}

}

init.sort(null);

double sp = 0.4;

int s = (int)(sp \* init.size());

Map < String, Integer > c = new HashMap < > ();

for (String i: init) {

for (String[] d: data) {

if (containsItem(d, i)) {

c.put(i, c.getOrDefault(i, 0) + 1);

}

}

}

System.out.println("C1:");

for (String key: c.keySet()) {

System.out.println("[" + key + "]: " + c.get(key));

}

System.out.println();

Map < Set < String > , Integer > l = new HashMap < > ();

for (String i: c.keySet()) {

if (c.get(i) >= s) {

Set < String > itemSet = new HashSet < > ();

itemSet.add(i);

l.put(itemSet, c.get(i));

}

}

System.out.println("L1:");

for (Set < String > key: l.keySet()) {

System.out.println(key + ": " + l.get(key));

}

System.out.println();

Map < Set < String > , Integer > pl = l;

int pos = 1;

for (int count = 2; count < 1000; count++) {

Set < Set < String >> nc = new HashSet < > ();

List < Set < String >> temp = new ArrayList < > (l.keySet());

for (int i = 0; i < temp.size(); i++) {

for (int j = i + 1; j < temp.size(); j++) {

Set < String > t = new HashSet < > (temp.get(i));

t.addAll(temp.get(j));

if (t.size() == count) {

nc.add(t);

}

}

}

Map < Set < String > , Integer > cMap = new HashMap < > ();

for (Set < String > i: nc) {

cMap.put(i, 0);

for (String[] q: data) {

Set < String > tempSet = new HashSet < > (List.of(q).subList(1,

q.length));

if (i.stream().allMatch(tempSet::contains)) {

cMap.put(i, cMap.get(i) + 1);

}

}

}

System.out.println("C" + count + ":");

for (Set < String > key: cMap.keySet()) {

System.out.println(key + ": " + cMap.get(key));

}

System.out.println();

l = new HashMap < > ();

for (Set < String > i: cMap.keySet()) {

if (cMap.get(i) >= s) {

l.put(i, cMap.get(i));

}

}

System.out.println("L" + count + ":");

for (Set < String > key: l.keySet()) {

System.out.println(key + ": " + l.get(key));

}

System.out.println();

if (l.isEmpty()) {

break;

}

pl = l;

pos = count;

}

System.out.println("Result:");

System.out.println("L" + pos + ":");

for (Set < String > key: pl.keySet()) {

System.out.println(key + ": " + pl.get(key));

}

for (Set < String > lSet: pl.keySet()) {

List < String > cList = new ArrayList < > (lSet);

List < Set < String >> subsets = generateSubsets(cList);

double mmax = 0;

for (Set < String > a: subsets) {

Set < String > b = new HashSet < > (cList);

b.removeAll(a);

int sab = 0;

int sa = 0;

int sb = 0;

for (String[] q: data) {

Set < String > tempSet = new HashSet < > (List.of(q).subList(1,

q.length));

if (a.stream().allMatch(tempSet::contains)) {

sa++;

}

if (b.stream().allMatch(tempSet::contains)) {

sb++;

}

if (cList.stream().allMatch(tempSet::contains)) {

sab++;

}

}

double temp = (double) sab / sa \* 100;

if (temp > mmax) {

mmax = temp;

}

System.out.println(a + " -> " + b + " = " + temp + "%");

System.out.println(b + " -> " + a + " = " + (double) sab / sb \* 100 +

"%");

}

int curr = 1;

System.out.print("Choosing: ");

for (Set < String > a: subsets) {

Set < String > b = new HashSet < > (cList);

b.removeAll(a);

int sab = 0;

int sa = 0;

int sb = 0;

for (String[] q: data) {

Set < String > tempSet = new HashSet < > (List.of(q).subList(1,

q.length));

if (a.stream().allMatch(tempSet::contains)) {

sa++;

}

if (b.stream().allMatch(tempSet::contains)) {

sb++;

}

if (cList.stream().allMatch(tempSet::contains)) {

sab++;

}

}

double temp = (double) sab / sa \* 100;

if (temp == mmax) {

System.out.print(curr + " ");

}

curr++;

temp = (double) sab / sb \* 100;

if (temp == mmax) {

System.out.print(curr + " ");

}

curr++;

}

System.out.println();

}

}

private static boolean containsItem(String[] itemSet, String item) {

for (int i = 1; i < itemSet.length; i++) {

if (itemSet[i].equals(item)) {

return true;

}

}

return false;

}

private static List < Set < String >> generateSubsets(List < String > originalSet) {

List < Set < String >> subsets = new ArrayList < > ();

int n = originalSet.size();

for (int i = 0; i < (1 << n); i++) {

Set < String > subset = new HashSet < > ();

for (int j = 0; j < n; j++) {

if ((i & (1 << j)) > 0) {

subset.add(originalSet.get(j));

}

}

subsets.add(subset);

}

subsets.remove(0); // Remove the empty set

return subsets;

}

}

**Output:**

C1:

[I1]: 6

[I2]: 7

[I3]: 6

[I4]: 2

[I5]: 2

L1:

[I1]: 6

[I2]: 7

[I3]: 6

[I4]: 2

[I5]: 2

C2:

[I1, I2]: 4

[I1, I3]: 4

[I1, I4]: 1

[I2, I3]: 4

[I1, I5]: 2

[I2, I4]: 2

[I2, I5]: 2

[I3, I4]: 0

[I3, I5]: 1

[I4, I5]: 0

L2:

[I1, I2]: 4

[I1, I3]: 4

[I2, I3]: 4

[I1, I5]: 2

[I2, I4]: 2

[I2, I5]: 2

C3:

[I2, I4, I5]: 0

[I1, I2, I3]: 2

[I1, I2, I4]: 1

[I1, I2, I5]: 2

[I1, I3, I5]: 1

[I2, I3, I4]: 0

[I2, I3, I5]: 1

L3:

[I1, I2, I3]: 2

[I1, I2, I5]: 2

C4:

[I1, I2, I3, I5]: 1

L4:

Result:

L3:

[I1, I2, I3]: 2

[I1, I2, I5]: 2

[I1] -> [I2, I3] = 33.33333333333333%

[I2, I3] -> [I1] = 50.0%

[I2] -> [I1, I3] = 28.57142857142857%

[I1, I3] -> [I2] = 50.0%

[I1, I2] -> [I3] = 50.0%

[I3] -> [I1, I2] = 33.33333333333333%

[I3] -> [I1, I2] = 33.33333333333333%

[I1, I2] -> [I3] = 50.0%

[I1, I3] -> [I2] = 50.0%

[I2] -> [I1, I3] = 28.57142857142857%

[I2, I3] -> [I1] = 50.0%

[I1] -> [I2, I3] = 33.33333333333333%

[I1, I2, I3] -> [] = 100.0%

[] -> [I1, I2, I3] = 22.22222222222222%

Choosing: 13

[I1] -> [I2, I5] = 33.33333333333333%

[I2, I5] -> [I1] = 100.0%

[I2] -> [I1, I5] = 28.57142857142857%

[I1, I5] -> [I2] = 100.0%

[I1, I2] -> [I5] = 50.0%

[I5] -> [I1, I2] = 100.0%

[I5] -> [I1, I2] = 100.0%

[I1, I2] -> [I5] = 50.0%

[I1, I5] -> [I2] = 100.0%

[I2] -> [I1, I5] = 28.57142857142857%

[I2, I5] -> [I1] = 100.0%

[I1] -> [I2, I5] = 33.33333333333333%

[I1, I2, I5] -> [] = 100.0%

[] -> [I1, I2, I5] = 22.22222222222222%

Choosing: 2 4 6 7 9 11 13

**Conclusion:** Thus, in this experiment, we have implemented Association Rule Mining algorithm using Apriori algorithm.