

## **Experiment 10**

**Aim:** To implement the Apriori algorithm in C++.

**Theory:** Association rules are if-then statements that help to show the probability of relationships between data items within large data sets in various types of databases. Association rule mining has a number of applications and is widely used to help discover sales correlations in transactional data or in medical data sets. Association rule mining, at a basic level, involves the use of machine learning models to analyze data for patterns, or co-occurrence, in a database. It identifies frequent if-then associations, which are called association rules. An association rule has two parts: an antecedent (if) and a consequent (then). An antecedent is an item found within the data. A consequent is an item found in combination with the antecedent.

Association rules are created by searching data for frequent if-then patterns and using the criteria support and confidence to identify the most important relationships. Support is an indication of how frequently the items appear in the data. Confidence indicates the number of times the if-then statements are found true. A third metric, called lift, can be used to compare confidence with expected confidence. Association rules are calculated from itemsets, which are made up of two or more items. If rules are built from analyzing all the possible itemsets, there could be so many rules that the rules hold little meaning. With that, association rules are typically created from rules well represented in data.

Measure 1: Support. This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears. The support of an itemset X,  $\text{supp}(X)$  is the proportion of transaction in the database in which item X appears. It signifies the popularity of an itemset.

$$\text{supp}(X) = \text{Number of transaction in which X appears} / \text{Total number of transactions.}$$

If the sales of a particular product (item) above a certain proportion have a meaningful effect on profits, that proportion can be considered as the support threshold. Furthermore, we can identify itemsets that have support values beyond this threshold as significant itemsets.

Measure 2: Confidence. This says how likely item Y is purchased when item X is purchased, expressed as  $\{X \rightarrow Y\}$ . This is measured by the proportion of transactions with item X, in which item Y also appears. Confidence of a rule is defined as follows:

$$\text{conf}(X \rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)}$$

It signifies the likelihood of item Y being purchased when item X is purchased. This implies that for 75% of the transactions containing onion and potatoes, the rule is correct. It can also be interpreted as the conditional probability  $P(Y|X)$ , i.e, the probability of finding the itemset Y in transactions given the transaction already contains X. It can give some important insights, but it also has a major drawback. It only takes into account the popularity of the itemset X and not the popularity of Y. If Y is equally popular as X then there will be a higher probability that a transaction containing X will also contain Y thus increasing the confidence. To overcome this drawback there is another measure called lift.

Measure 3: Lift. This says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is. A lift value greater than 1 means that item Y is likely to be bought if item X is bought, while a value less than 1 means that item Y is unlikely to be bought if item X is bought. The lift of a rule is defined as:

$$lift(X \rightarrow Y) = \frac{supp(X \cup Y)}{supp(X) * supp(Y)}$$

This signifies the likelihood of the itemset Y being purchased when item X is purchased while taking into account the popularity of Y. If the value of lift is greater than 1, it means that the itemset Y is likely to be bought with itemset X, while a value less than 1 implies that itemset Y is unlikely to be bought if the itemset X is bought

### Algorithm:

```

Apriori( $T, \epsilon$ )
     $L_1 \leftarrow \{\text{large 1-itemsets}\}$ 
     $k \leftarrow 2$ 
    while  $L_{k-1} \neq \emptyset$ 
         $C_k \leftarrow \{c = a \cup \{b\} \mid a \in L_{k-1} \wedge b \notin a, \{s \subseteq c \mid |s| = k-1\} \subseteq L_{k-1}\}$ 
        for transactions  $t \in T$ 
             $D_t \leftarrow \{c \in C_k \mid c \subseteq t\}$ 
            for candidates  $c \in D_t$ 
                 $count[c] \leftarrow count[c] + 1$ 
             $L_k \leftarrow \{c \in C_k \mid count[c] \geq \epsilon\}$ 
         $k \leftarrow k + 1$ 
    return  $\bigcup_k L_k$ 

```

### Applications:

1. Market Basket Analysis: This is the most typical example of association mining. Data is collected using barcode scanners in most supermarkets. This database, known as the “market basket” database, consists of a large number of records on past transactions. A

single record lists all the items bought by a customer in one sale. Knowing which groups are inclined towards which set of items gives these shops the freedom to adjust the store layout and the store catalogue to place the optimally concerning one another.

2. **Medical Diagnosis:** Association rules in medical diagnosis can be useful for assisting physicians for curing patients. Diagnosis is not an easy process and has a scope of errors which may result in unreliable end-results. Using relational association rule mining, we can identify the probability of the occurrence of an illness concerning various factors and symptoms. Further, using learning techniques, this interface can be extended by adding new symptoms and defining relationships between the new signs and the corresponding diseases.
3. **Census Data:** Every government has tonnes of census data. This data can be used to plan efficient public services(education, health, transport) as well as help public businesses (for setting up new factories, shopping malls, and even marketing particular products). This application of association rule mining and data mining has immense potential in supporting sound public policy and bringing forth an efficient functioning of a democratic society.
4. **Protein Sequence:** Proteins are sequences made up of twenty types of amino acids. Each protein bears a unique 3D structure which depends on the sequence of these amino acids. A slight change in the sequence can cause a change in structure which might change the functioning of the protein. This dependency of the protein functioning on its amino acid sequence has been a subject of great research. Earlier it was thought that these sequences are random, but now it's believed that they aren't. Knowledge and understanding of these association rules will come in extremely helpful during the synthesis of artificial proteins.

### **Source Code:**

```
#include <bits/stdc++.h>
#include <map>
using namespace std;
ifstream fin;
double minfre;
vector<set<string> > datatable;
set<string> products;
map<string, int> freq;
vector<string> wordsof(string str)
{
    vector<string> tmpset;
    string tmp = "";
    int i = 0;
    while (str[i])
```

```

{
    if (isalnum(str[i]))
        tmp += str[i];
    else
    {
        if (tmp.size() > 0)
            tmpset.push_back(tmp);
        tmp = "";
    }
    i++;
}
if (tmp.size() > 0)
    tmpset.push_back(tmp);
return tmpset;
}

string combine(vector<string> &arr, int miss)
{
    string str;
    for (int i = 0; i < arr.size(); i++)
        if (i != miss)
            str += arr[i] + " ";
    str = str.substr(0, str.size() - 1);
    return str;
}

set<string> cloneit(set<string> &arr)
{
    set<string> dup;
    for (set<string>::iterator it = arr.begin(); it != arr.end(); it++)
        dup.insert(*it);
    return dup;
}

set<string> apriori_gen(set<string> &sets, int k)
{
    set<string> set2;
    for (set<string>::iterator it1 = sets.begin(); it1 != sets.end(); it1++)
    {
        set<string>::iterator it2 = it1;
        it2++;
        for (; it2 != sets.end(); it2++)
        {
            vector<string> v1 = wordsof(*it1);

```

```

vector<string> v2 = wordsof(*it2);
bool alleq = true;
for (int i = 0; i < k - 1 && alleq; i++)
if (v1[i] != v2[i])
alleq = false;
if (!alleq)
continue;
v1.push_back(v2[k - 1]);
if (v1[v1.size() - 1] < v1[v1.size() - 2])
swap(v1[v1.size() - 1], v1[v1.size() - 2]);
for (int i = 0; i < v1.size() && alleq; i++)
{
string tmp = combine(v1, i);
if (sets.find(tmp) == sets.end())
alleq = false;
}
if (alleq)
set2.insert(combine(v1, -1));
}
}
return set2;
}
int main()
{
fin.open("apriori.in");
cout << "Frequency % .:";
cin >> minfre;
string str;
while (!fin.eof())
{
getline(fin, str);
vector<string> arr = wordsof(str); //taking data from file ,
set<string> tmpset;
for (int i = 0; i < arr.size(); i++)
tmpset.insert(arr[i]);
datatable.push_back(tmpset);
for (set<string>::iterator it = tmpset.begin(); it != tmpset.end(); it++)
{
products.insert(*it);
freq[*it]++;
}
}

```

```

}
fin.close();
cout << "No of transactions: " << datatable.size() << endl;
minfre = minfre * datatable.size() / 100;
cout << "Min frequency:" << minfre << endl;
queue<set<string>::iterator> q;
for (set<string>::iterator it = products.begin(); it != products.end(); it++)
if (freq[*it] < minfre)
q.push(it);
while (q.size() > 0)
{
products.erase(*q.front());
q.pop();
}
for (set<string>::iterator it = products.begin(); it != products.end(); it++)
cout << *it << " " << freq[*it] << endl;
int i = 2;
set<string> prev = cloneit(products);
while (i)
{
set<string> cur = apriori_gen(prev, i - 1);
if (cur.size() < 1)
break;
for (set<string>::iterator it = cur.begin(); it != cur.end(); it++)
{
vector<string> arr = wordsof(*it);
int tot = 0;
for (int j = 0; j < datatable.size(); j++)
{
bool pres = true;
for (int k = 0; k < arr.size() && pres; k++)
if (datatable[j].find(arr[k]) == datatable[j].end())
pres = false;
if (pres)
tot++;
}
if (tot >= minfre)
freq[*it] += tot;
else
q.push(it);
}
}


```

```

while (q.size() > 0)
{
    cur.erase(*q.front());
    q.pop();
}
for (set<string>::iterator it = cur.begin(); it !=
cur.end(); it++)
    cout << *it << " " << freq[*it] << endl;
prev = cloneit(cur);
i++;
}
}

```

## Output:



```

Anurags-MacBook-Air:DWDM_LAB jarvis$ ./ap
Frequency % :10
No of transactions: 9
Min frequency:0.9
I1 6
I2 7
I3 6
I4 2
I5 2
I1 I2 4
I1 I3 4
I1 I4 1
I1 I5 2
I2 I3 4
I2 I4 2
I2 I5 2
I3 I5 1
I1 I2 I3 2
I1 I2 I4 1
I1 I2 I5 2
I1 I3 I5 1
I2 I3 I5 1
I1 I2 I3 I5 1
Anurags-MacBook-Air:DWDM_LAB jarvis$

```

## Findings and Learnings :

1. We learned about association rule mining and familiarized ourselves with the test.arff dataset.
2. We implemented and tested Apriori in C++.