Data Warehousing & Data Mining LAB - G2
EXPERIMENT 5

- ASHISH KUMAR
- 2K18/SE/041

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

Theory:-

Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for boolean association rule. Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

The key concept of Apriori algorithm is its anti-monotonicity of support measure. Apriori assumes that all subsets of a frequent itemset must be frequent (Apriori property). If an itemset is infrequent, all its supersets will be infrequent. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset.

The Apriori algorithm is used for mining frequent itemsets and devising association rules from a transactional database. The parameters “support” and “confidence” are used. Support refers to items’ frequency of occurrence; confidence is a conditional probability. Items in a transaction form an item set. The algorithm begins by identifying frequent, individual items (items with a frequency greater than or equal to the given support) in the database and continues to extend them to larger, frequent itemsets.

Algorithm:

Below are the steps for the apriori algorithm:
1. Determine the support of itemsets in the transactional database, and select the minimum support and confidence.
2. Take all supports in the transaction with higher support value than the minimum or selected support value.
3. Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.
4. Sort the rules as the decreasing order of lift values.

Apriori(T, ϵ)

$L_1 \leftarrow \{\text{large } 1-\text{itemsets}\}$

$k \leftarrow 2$

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 $\bigcup_k L_k$

SOURCE CODE:-

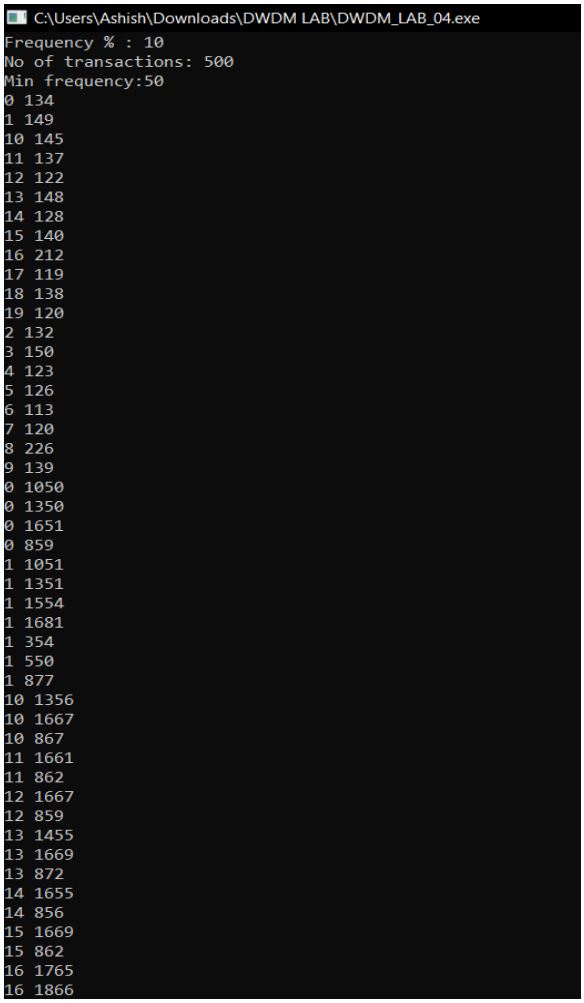
```
#include <bits/stdc++.h>
#include <map>
using namespace std;
ifstream fin;
vector < set < string > > datatable;
set < string > products;
map < string, int > freq;
vector < string > wordsof(string str) {
  vector < string > tmpset;
  string tmp = "";
  int i = 0;
  while (str[i]) {
    if (isalnum(str[i]))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



Findings and Learning:

1. We learned about association rule mining.
2. We successfully implemented and tested Apriori in C++.