Experiment No-9

**Aim:** Implementation of FPM Association Rule Mining Algorithm using (C,C++,Java).

**Theory:**

In Data Mining the task of finding frequent pattern in large databases is very important and has been studied in large scale in the past few years. Unfortunately, this task is computationally expensive, especially when a large number of patterns exist.

The FP-Growth Algorithm, proposed by Han in, is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). In his study, Han proved that his method outperforms other popular methods for mining frequent patterns, e.g. the Apriori Algorithm and the TreeProjection . In some later works.it was proved that FP-Growth has better performance than other methods, including Eclat and Relim. The popularity and efficiency of FP-Growth Algorithm contributes with many studies that propose variations to improve his performance.This chapter describes the algorithm and some variations and discuss features of the R language and strategies to implement the algorithm to be used in R. Next, a brief conclusion and future works are proposed.

**The algorithm:**

The FP-Growth Algorithm is an alternative way to find frequent itemsets without using candidate generations, thus improving performance. For so much it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the itemset association information.

In simple words, this algorithm works as follows: first it compresses the input database creating an FP-tree instance to represent frequent items. After this first step it divides the compressed database into a set of conditional databases, each one associated with one frequent pattern. Finally, each such database is mined separately. Using this strategy, the FP-Growth reduces the search costs looking for short patterns recursively and then concatenating them in the long frequent patterns, offering good selectivity. In large databases, it’s not possible to hold the FP-tree in the main memory. A strategy to cope with this problem is to firstly partition the database into a set of smaller databases (called projected databases), and then construct an FP-tree from each of these smaller databases.

The next subsections describe the FP-tree structure and FP-Growth Algorithm, finally an example is presented to make it easier to understand these concepts.

***Apriori.java***

import java.io.\*;

import java.util.\*;

class Apriori extends Observable {

public static void main(String[] args) throws Exception {

Apriori ap = new Apriori(args);

}

private List<int[]> itemsets ;

private String transaFile;

private int numItems;

private int numTransactions;

private double minSup;

private boolean usedAsLibrary = false;

public Apriori(String[] args, Observer ob) throws Exception

{

usedAsLibrary = true;

configure(args);

this.addObserver(ob);

go();

}

public Apriori(String[] args) throws Exception

{

configure(args);

go();

}

private void go() throws Exception {

long start = System.currentTimeMillis();

createItemsetsOfSize1();

int itemsetNumber=1; //the current itemset being looked at

int nbFrequentSets=0;

while (itemsets.size()>0)

{

calculateFrequentItemsets();

if(itemsets.size()!=0)

{

nbFrequentSets+=itemsets.size();

log("Found "+itemsets.size()+" frequent itemsets of size " + itemsetNumber + " (with support "+(minSup\*100)+"%)");;

createNewItemsetsFromPreviousOnes();

}

itemsetNumber++;

}

long end = System.currentTimeMillis();

log("Execution time is: "+((double)(end-start)/1000) + " seconds.");

log("Found "+nbFrequentSets+ " frequents sets for support "+(minSup\*100)+"% (absolute "+Math.round(numTransactions\*minSup)+")");

log("Done");

}

private void foundFrequentItemSet(int[] itemset, int support) {

if (usedAsLibrary) {

this.setChanged();

notifyObservers(itemset);

}

else {System.out.println(Arrays.toString(itemset) + " ("+ ((support / (double) numTransactions))+" "+support+")");}

}

private void log(String message) {

if (!usedAsLibrary) {

System.err.println(message);

}

}

private void configure(String[] args) throws Exception

{

if (args.length!=0) transaFile = args[0];

else transaFile = "chess.dat"; // default

if (args.length>=2) minSup=(Double.valueOf(args[1]).doubleValue());

else minSup = .8;// by default

if (minSup>1 || minSup<0) throw new Exception("minSup: bad value");

numItems = 0;

numTransactions=0;

BufferedReader data\_in = new BufferedReader(new FileReader(transaFile));

while (data\_in.ready()) {

String line=data\_in.readLine();

if (line.matches("\\s\*")) continue; // be friendly with empty lines

numTransactions++;

StringTokenizer t = new StringTokenizer(line," ");

while (t.hasMoreTokens()) {

int x = Integer.parseInt(t.nextToken());

if (x+1>numItems) numItems=x+1;

}

}

outputConfig();

}

private void outputConfig() {

log("Input configuration: "+numItems+" items, "+numTransactions+" transactions, ");

log("minsup = "+minSup+"%");

}

private void createItemsetsOfSize1() {

itemsets = new ArrayList<int[]>();

for(int i=0; i<numItems; i++)

{

int[] cand = {i};

itemsets.add(cand);

}

}

private void createNewItemsetsFromPreviousOnes()

{

int currentSizeOfItemsets = itemsets.get(0).length;

log("Creating itemsets of size "+(currentSizeOfItemsets+1)+" based on "+itemsets.size()+" itemsets of size "+currentSizeOfItemsets);

HashMap<String, int[]> tempCandidates = new HashMap<String, int[]>(); //temporary candidates

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

{

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

{

int[] X = itemsets.get(i);

int[] Y = itemsets.get(j);

assert (X.length==Y.length);

int [] newCand = new int[currentSizeOfItemsets+1];

for(int s=0; s<newCand.length-1; s++) {

newCand[s] = X[s];

}

int ndifferent = 0;

for(int s1=0; s1<Y.length; s1++)

{

boolean found = false;

for(int s2=0; s2<X.length; s2++) {

if (X[s2]==Y[s1]) {

found = true;

break;

}

}

if (!found){ // Y[s1] is not in X

ndifferent++;

newCand[newCand.length -1] = Y[s1];

}

}

assert(ndifferent>0);

if (ndifferent==1) {

Arrays.sort(newCand);

tempCandidates.put(Arrays.toString(newCand),newCand);

}

}

}

itemsets = new ArrayList<int[]>(tempCandidates.values());

log("Created "+itemsets.size()+" unique itemsets of size "+(currentSizeOfItemsets+1));

}

private void line2booleanArray(String line, boolean[] trans) {

Arrays.fill(trans, false);

StringTokenizer stFile = new StringTokenizer(line, " "); //read a line from the file to the tokenizer

while (stFile.hasMoreTokens())

{

int parsedVal = Integer.parseInt(stFile.nextToken());

trans[parsedVal]=true; //if it is not a 0, assign the value to true

}

}

private void calculateFrequentItemsets() throws Exception

{

log("Passing through the data to compute the frequency of " + itemsets.size()+ " itemsets of size "+itemsets.get(0).length);

List<int[]> frequentCandidates = new ArrayList<int[]>(); //the frequent candidates for the current itemset

boolean match; //whether the transaction has all the items in an itemset

int count[] = new int[itemsets.size()]; //the number of successful matches, initialized by zeros

BufferedReader data\_in = new BufferedReader(new InputStreamReader(new FileInputStream(transaFile)));

boolean[] trans = new boolean[numItems];

for (int i = 0; i < numTransactions; i++) {

String line = data\_in.readLine();

line2booleanArray(line, trans);

for (int c = 0; c < itemsets.size(); c++) {

match = true; // reset match to false

int[] cand = itemsets.get(c);

for (int xx : cand) {

if (trans[xx] == false) {

match = false;

Break; } }

if (match) { // if at this point it is a match, increase the count

count[c]++;

} } }

data\_in.close();

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

if ((count[i] / (double) (numTransactions)) >= minSup) {

foundFrequentItemSet(itemsets.get(i),count[i]);

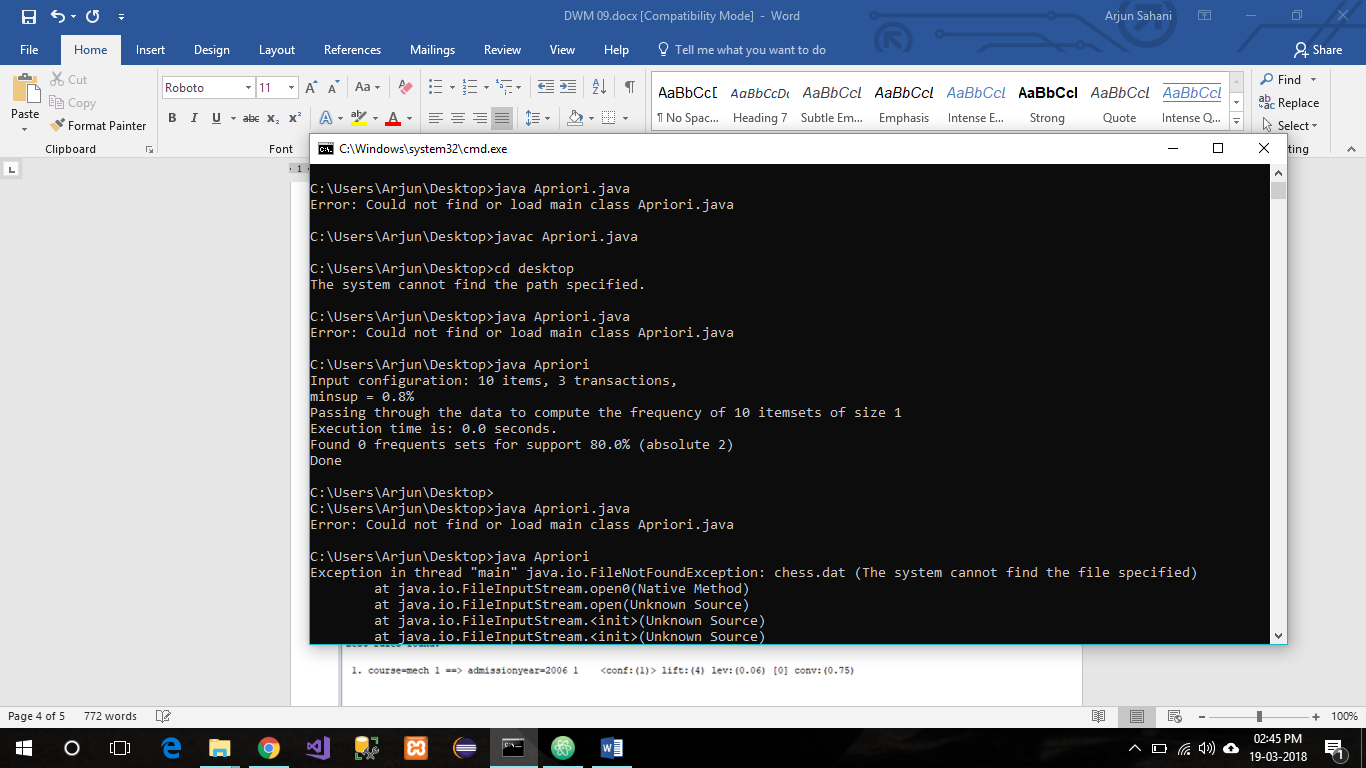
frequentCandidates.add(itemsets.get(i));

} }

itemsets = frequentCandidates;

} }

**Output:**



**Conclusion:**

Thus, FPM Association Algorithm using Java is implemented and studied successfully.