package astro;

import java.io.\*;

import java.util.\*;

public class AlgoAprioriHT extends Observable

{

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

{

new AlgoAprioriHT (args);

}

private List<int[]> itemsets ;

private String transaFile;

private int numItems;

private int numTransactions;

private double minSup;

private boolean usedAsLibrary = false;

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

{

usedAsLibrary = true;

configure(args);

this.addObserver(ob);

go();

}

public AlgoAprioriHT(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 = "E:\\data.csv"; // 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());

//log(x);

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

}

}

outputConfig();

data\_in.close();

}

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[]>();

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];

}

}

// we have to find at least 1 different, otherwise it means that we have two times the same set in the existing candidates

assert(ndifferent>0);

if (ndifferent==1) {

// HashMap does not have the correct "equals" for int[] :-(

// I have to create the hash myself using a String :-(

// I use Arrays.toString to reuse equals and hashcode of String

Arrays.sort(newCand);

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

}

}

}

//set the new itemsets

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

//put the contents of that line into the transaction array

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

// load the transaction file

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

boolean[] trans = new boolean[numItems];

// for each transaction

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

// boolean[] trans = extractEncoding1(data\_in.readLine());

String line = data\_in.readLine();

line2booleanArray(line, trans);

// check each candidate

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

match = true; // reset match to false

// tokenize the candidate so that we know what items need to be

// present for a match

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

//int[] cand = candidatesOptimized[c];

// check each item in the itemset to see if it is present in the

// transaction

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]++;

//log(Arrays.toString(cand)+" is contained in trans "+i+" ("+line+")");

}

}

}

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));

}

//else log("-- Remove candidate: "+ Arrays.toString(candidates.get(i)) + " is: "+ ((count[i] / (double) numTransactions)));

}

//new candidates are only the frequent candidates

itemsets = frequentCandidates;

}

}