PRAJAKTA DEOKULE

A1

3330

C22019221332

ID3.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Scanner;
import java.util.Vector;

public class Id3 {

   static Scanner in  = new Scanner(System.in);
   static String [][] table = null;

    public static void main(String[] args) {
        long startTime = System.currentTimeMillis();

         menu();
        ID3_calculation obj = new ID3_calculation(table);
        obj.calculate_class();
        obj.calculate_attribute();
        obj.calculate_entropy();
        obj.information_gain();

        List<Node> node = obj.getNode();
        HashMap<String,Double > information_gain = obj.getInformationGain();
        HashMap<String,String > information_gain_subAttribute =
obj.getInformationGain_of_subAttribute();

        Vector attributes = obj.getlistofAttributes();
        GenerateTree tree = new GenerateTree(attributes , node , information_gain ,
information_gain_subAttribute  );
        tree.create_tree();
       tree.Display_attribute();
       tree.display_tree();

        long endTime = System.currentTimeMillis();
      long elapsed = endTime - startTime;
            System.out.println("Time taken : " + (float)elapsed/1000 + " seconds");

    }
```

```
    public static void menu(){

    String choose="";
    System.out.println("------- Decision Tree Implementation using ID3 Algorithm ----
---");
    System.out.println("\n------- Weather DataSet----------");

            readCSV("C:/Users/admin/Desktop/dmdw lab/id3/weather1.csv");
            change_numeric_to_name();
    }


    public static void change_numeric_to_name( )
   {
            for(int j = 1 ; j < table.length; j++)
           {

                    if( Integer.parseInt(table[j][1]) >= 80  ){
                       table[j][1] = "hot";
                    }
                    else if( Integer.parseInt(table[j][1]) >= 70  ){
                       table[j][1] = "mild";
                    }
                    else if( Integer.parseInt(table[j][1]) < 70  ){
                       table[j][1] = "cool";
                    }


                    if( Integer.parseInt(table[j][2]) > 80  ){
                       table[j][2] = "high";
                    }
                    else if( Integer.parseInt(table[j][2]) <= 80  ){
                       table[j][2] = "normal";
                   }
        }
    }

    public static void readCSV(String filename){

        String csvFile = filename;
        BufferedReader br = null;
        BufferedReader pre_count = null;

        String line = "";
        String cvsSplitBy = ",";

        int row=0;
        int col=0;


        try {

            pre_count = new BufferedReader(new FileReader(csvFile));

            pre_count = new BufferedReader(new FileReader(csvFile));
```

```
            while ((line = pre_count.readLine()) != null) {

                // use comma as separator
                String[] attributes = line.split(cvsSplitBy);
                col = attributes.length - 1;
                row++;
            }

            // size of table

            table  = new String [row][col];

            int rows =0;

            br = new BufferedReader(new FileReader(csvFile));
            while ((line = br.readLine()) != null) {


                String[] attributes = line.split(cvsSplitBy);


                for(int i = 1 ; i < col+1 ; i++){
                    table[rows][i-1] = attributes[i];


                }

        }
        catch (IOException e) {
           System.out.println("File not found Exception");
         } finally {
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                  System.out.println("File not found Exception Finally");
                }
            }
        }

    }

}
```

Id3Calculation.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import java.text.DecimalFormat;
import java.util.HashMap;

public class ID3_calculation {

    HashMap<String,Double > information_gain = new HashMap<String,Double>();
```

```
HashMap<String,String > gain_per_subattribute = new HashMap<String,String>();
Vector attributes = new Vector();
Vector classification = new Vector();
List<Node> node = new ArrayList<>();

double morethan_onezero = 0.009;

//H(S)
double entropy =0;

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

public ID3_calculation(String[][] table) {
    this.table = table;
}

public List<Node> getNode(){
  return node;
}

public Vector getlistofAttributes(){
  return attributes;
}

public HashMap<String,Double> getInformationGain(){

    return information_gain;

}


 public  HashMap<String,String > getInformationGain_of_subAttribute(){

    return gain_per_subattribute;

}


public void calculate_attribute(){


        for(int j = 0 ; j < table[0].length-1; j++){

             if(!attributes.contains(table[0][j])){
               attributes.addElement(table[0][j]);
             }
 }}

public void calculate_class(){

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

         for(int j = (table[i].length-1) ; j < table[i].length;
```

```
                if(!classification.contains(table[i][j])){
                  classification.addElement(table[i][j]);

                }
}}}

    public void calculate_entropy(){


        List<Integer> total_classification_num = new ArrayList<>();
        double total_entropy=0;

        double total_rows  = table.length - 1;
        int count =0;

        // part 1 for H(S)
        for(int z = 0 ; z < classification.size(); z++){

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

                if(classification.get(z).toString().equals(table[i][ (table[i].length
- 1) ] )){
                 count++;
                }

             }
             total_classification_num.add(count);
             count=0;
```

```
        }
        // part 2 for H(S)

      for(int z = 0 ; z <  total_classification_num.size(); z++){

           double ps =  total_classification_num.get(z);
           double cls_entropy = -1* ( (ps/total_rows) * log(ps/total_rows,2) ) ;
           entropy = entropy + cls_entropy;
      }

      DecimalFormat df2 = new DecimalFormat(".##");
      String change = df2.format(entropy);

      if(change.contains(",")) {
         change = change.replace(",", ".");
      }

      entropy = Double.parseDouble(change);
      System.out.println( "\nEntropy of complete dataset = " + entropy+"\n");


    }

    public void information_gain(){


        HashMap<String,Integer> frequency ;
        HashMap<String,List<String> > frequency_index ;


        HashMap<String,List<classification> > classifies ;

        List<HashMap<String,List<classification> > > listofclassifies = new
ArrayList();

        List<HashMap<String,Integer>> listoffrequency = new ArrayList();

        List<HashMap<String,List<String> >> listoffrequency_index  = new ArrayList();
        // initial values
        for(int i = 0 ; i < attributes.size(); i++ ){

            classifies = new  HashMap<String,List<classification> >();
            frequency = new HashMap<String,Integer>();


            for(int j = 1 ; j < table.length; j++ ){

                if(! frequency.containsKey(table[j][i]) ){
                 frequency.put(table[j][i], 0);
               }

                if(classifies.containsKey(table[j][i]) ){

                    List<classification> temp = classifies.get(table[j][i]);
                        int flag =0;


                if(flag == 0){

                        ,1));
```

```
t                       p.add(new classification( table[j][table[j].length-1]

e                       classifies.put(table[j][i],  temp );

m                       }



                }
                 else{

                     List<classification> temp = classifies.get(table[j][i]);

                     if(temp == null){
                         temp = new ArrayList<>();
                         temp.add(new classification( table[j][table[j].length-1]
,1));
                         classifies.put(table[j][i],  temp );
                      }}}
             listofclassifies.add(classifies);
             listoffrequency.add(frequency);
         }
```

```
        // pre calculated


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


            List<String> attri = new ArrayList<>();
            frequency_index = new HashMap<String,List<String>>();
            for(int j = 1 ; j < table.length; j++ ){

                if(!attri.contains(table[j][i])){
                  attri.add(table[j][i]);
                }


                    List<classification> ty =
node.get(j).getClassifies().get(parts.get(z));

                    // initial value
                    int cc = ty.get(0).getRepetition();
                    String str = ty.get(0).getClassification_attributes();

                     for(int q = 0 ; q < ty.size(); q++){

                        if(q >= 1 ){
                         if(cc == ty.get(q).getRepetition()){

if(addd.contains(ty.get(q).getClassification_attributes())){

                                for(int k =0 ; k < ty.size() ; k++){


if(addd.contains(ty.get(k).getClassification_attributes())){

                                 }

                                }

                        else if(cc < ty.get(q).getRepetition()){

                            cc = ty.get(q).getRepetition();
                            str = ty.get(q).getClassification_attributes();
                            addd.add(str);

if(gain == 0.0){
                  gain = morethan_onezero;
                  morethan_onezero = morethan_onezero - 0.001;
                }


                information_gain.put(attributes.get(i).toString(),
                System.out.println(attributes.get(i).toString()+ "
```

```
    static double log(double x, int base)
{
    return (double) (Math.log(x) / Math.log(base));
}

}
```

Node.java

```
import java.util.HashMap;
import java.util.List;


public class Node {

String attribute;
List<String> listofattribute;
HashMap<String,Integer> frequency;
HashMap<String,List<String> > frequency_index ;
HashMap<String,List<classification> > classifies ;

    public Node(String attribute, List<String> listofattribute, HashMap<String,
Integer> frequency, HashMap<String, List<String>> frequency_index, HashMap<String,
List<classification>> classifies) {
        this.attribute = attribute;
        this.listofattribute = listofattribute;
        this.frequency = frequency;
        this.frequency_index = frequency_index;
        this.classifies = classifies;
    }

    public String getAttribute() {
        return attribute;
    }

    public void setAttribute(String attribute) {
        this.attribute = attribute;
    }

    public List<String> getListofattribute() {
        return listofattribute;
    }


    public void setFrequency_index(HashMap<String, List<String>> frequency_index) {
        this.frequency_index = frequency_index;
    }

    public HashMap<String, List<classification>> getClassifies() {
        return classifies;
    }

}
```

GenerateTree.java

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Vector;


public class GenerateTree {


    Vector listofattributes;
    List<Node> node;
    HashMap<String,Double > information_gain;
    HashMap<String,String > information_gain_subAttribute;
    TreeNode treenode = new TreeNode();
    HashMap <String , List<String>>  Main_Attribute = new HashMap <String ,
List<String>>();
    List<String> sort;


    public GenerateTree( Vector listofattributes, List<Node> node ,
HashMap<String,Double > information_gain , HashMap<String,String >
information_gain_subAttribute ){

        this.listofattributes = listofattributes;
        this.node = node;
        this.information_gain = information_gain;
        this.information_gain_subAttribute = information_gain_subAttribute;
    }

    public void Display_attribute() {

        System.out.println("-.....");
        System.out.println("\nAttributes of Weather dataset\n");

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

            System.out.print((i+1)+" "+sort.get(i)+" ");

        }

        System.out.println("\n-.....");
    }

    public void create_tree() {


        System.out.println("-.....");

     System.out.println("\nGenerating Tree.....\n");
        sort = new ArrayList();
        double aa =0;
        String loc ="";
```

```
        // for sorting , finding root
        for(int i = 0 ; i < information_gain.size(); i++ ){

            for(String jj:  information_gain.keySet() ){

               if(!sort.contains(jj)){

                   if(aa < information_gain.get(jj) ){

   aa = information_gain.get(jj);

                       loc = jj;
                   }

               }
           }

            if(!loc.equals("")){
sort.add(loc);
            }

          aa=0;
          loc = "";

        }


       System.out.println(sort);
       // get all main or sub attribute stored
        for(int i = 0 ; i < node.size() ; i++){

            String main = node.get(i).getAttribute();
            List<String> rel = node.get(i).getListofattribute();
            Main_Attribute.put(main, rel);

        }


       // set root
       treenode.Set_root(sort.get(0));
       int count =1;
       // traverse and create tree
       for(int i = 0 ; i < sort.size()-1 ; i++){

           String parent =  sort.get(i);
           List<String> rel = Main_Attribute.get(parent);
           List<String> child = new ArrayList<>();
           // setting childs
            for(int j =0 ; j < rel.size(); j++ ){
                    if(this.information_gain_subAttribute.get(rel.get(j)).equals("0")
){

child.add(information_gain_subAttribute.get((rel.get(j)+"1")));
                    }
                    else{
```

```
                    if(count < sort.size()-1 ){
                    child.add(sort.get(count));
                    count++;
                    }
                    else{
          child.add(information_gain_subAttribute.get((rel.get(j))));
                    }

public void display_tree(){

    System.out.println("\n\n------Visualizing Decision Tree---------");
    System.out.println("\n\nRoot of Tree ==> "+ treenode.get_root() + "\n");
       System.out.println(treenode.get_root());
    gofor_child(treenode.get_root() , "" );

}


public int gofor_child(String parent, String space) {

   List<String> rel = treenode.getRelation(parent);
     List<String> child = treenode.getChild(parent);
   if(child == null) {
         return 0;
   }
   else {
         int c = rel.size();

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


               System.out.println(space+" "+rel.get(i)+":");
               System.out.println(space+"   "+child.get(i));

               if(treenode.getChild(child.get(i)) == null )  {}
               else {

                     String temp = space+"     ";
                     gofor_child( child.get(i) ,  temp );
               }

         }
```

Output:

```
<WEATHERDATASET>:
outlook                 temperature             humidity                windy                   play
-------------------------------------------------------------------------------------------------------------
sunny                   85                      85                      FALSE                   no
sunny                   80                      90                      TRUE                    no
overcast                83                      86                      FALSE                   yes
rainy                   70                      96                      FALSE                   yes
rainy                   68                      80                      FALSE                   yes
rainy                   65                      70                      TRUE                    no
overcast                64                      65                      TRUE                    yes
sunny                   72                      95                      FALSE                   no
sunny                   69                      70                      FALSE                   yes
rainy                   75                      80                      FALSE                   yes
sunny                   75                      70                      TRUE                    yes
overcast                72                      90                      TRUE                    yes
overcast                81                      75                      FALSE                   yes
rainy                   71                      91                      TRUE                    no
```

```
------- Decision Tree Implementation using ID3 Algorithm -------

------- Weather DataSet -------

Entropy of complete dataset = 0.94

Information Gain

outlook = 0.246
temp = 0.029
humidity = 0.152
wind = 0.048
-----------------------------------

Generating Tree ....

[outlook, humidity, wind, temp]
-----------------------------------

Attributes of Weather dataset

1 outlook 2 humidity 3 wind 4 temp
-----------------------------------
```

```
------Visualizing Decision Tree------


Root of Tree ==> outlook

outlook
 sunny:
   humidity
      high:
        no
      normal:
        yes
 overcast:
   yes
 rainy:
   wind
      FALSE:
        yes
      TRUE:
        no
Time taken : 0.169 seconds
```