

```

1  import java.io.BufferedReader;
2  import java.io.File;
3  import java.io.FileNotFoundException;
4  import java.io.FileReader;
5  import java.io.IOException;
6  import java.lang.reflect.Array;
7  import java.util.*;
8  import java.util.Map.Entry;
9
10
11  public class Decision_Tree
12  {
13
14      static int columb_count=0;
15      static int row_count=0;
16      static private BitSet columns;
17      static private BitSet rows;
18
19
20      static String targetValue;
21      static HashMap<String , Integer> tvalue= new HashMap<String, Integer>();
22
23      /*
24      static String value1=null;
25      static String value2=null;
26      static int vall_count=0;
27      static int val2_count=0;
28      */
29      static File file=new File("C:\\Users\\Tanvi\\Desktop\\MCA IV sem\\Data
Mining\\Decision_Tree\\data.txt");
30      static double Targetentropy=0.0;
31
32
33
34      public static void main(String[] args) throws IOException
35      {
36
37          // TODO Auto-generated method stub
38          // fetch total number of rows and columns in table.(done)
39          // get target class name, value1, value2 and entropy of that class.(done)
40          FileReader fileReader = new FileReader(file);
41          BufferedReader breader = new BufferedReader(fileReader);
42          String line;
43          int counter = 0;
44
45          ArrayList<ArrayList<String>> data = new ArrayList<ArrayList<String>>();
46
47          while ((line = breader.readLine()) != null)
48          {
49              String[] cols = line.split("\\\\|");
50
51              ArrayList<String> k= new ArrayList<String>();
52              Collections.addAll(k,cols);
53              data.add(k);
54              if (counter == 0)
55              {
56                  columb_count = cols.length;
57                  targetValue = cols[cols.length - 1];
58              }
59              else
60              {
61                  if(!tvalue.containsKey(cols[cols.length-1]))
62                  {
63                      tvalue.put(cols[cols.length-1], 0);
64
65                  }
66                  /*
67                  if(value1==null)
68                  {

```

```

69         value1=cols[cols.length-1];
70     }
71     else
72     {
73         if(!cols[cols.length-1].equalsIgnoreCase(value1))
74         {
75             value2=cols[cols.length-1];
76         }
77     }*/
78 }
79 counter++;
80 }
81
82
83 row_count = counter;
84 rows= new BitSet(row_count);
85 columns= new BitSet(columb_count);
86 columns.set(columb_count-1);
87
88 //printvalue
89 System.out.println("val:"+tvalue);
90 for (ArrayList<String> k:data)
91 {
92     for (String n:k)
93     {
94         System.out.print(" "+n);
95     }
96     System.out.println();
97 }
98
99
100 // fill attribute value and create the treenode based on that
101
102
103 TNode head=new TNode();
104 TNode kk=head;
105
106 ArrayList<Attribute> att= new ArrayList<Attribute>();
107
108 boolean once=true;
109 int counter2 =0;
110 while((kk=checktree(head)).getIndex() !=-2)
111 {
112     System.out.println("\n\n\n *** start ***");
113     if(counter2==3)
114         System.exit(0);
115
116     counter2++;
117
118
119     System.out.println("tree :"+kk);
120     // change data to point only to our required data.
121     ArrayList<ArrayList<String>> data1= required_data(data, kk, att);
122     System.out.println(data1);
123     // calculate value of P and N
124     calculate_val1_and_val2(data1);
125     System.out.println("value :"+tvalue);
126     // Calculate class Entropy
127     ArrayList<String>tname =new ArrayList<String>();
128     tname.addAll(tvalue.keySet());
129     Targetentropy=entropy(tvalue.get(tname.get(0)), tvalue.get(tname.get(1)));
130     System.out.println("target: "+Targetentropy);
131     //calculate entropy,gain for each attribute
132     att=attributevalue(data1, kk);
133
134     //calculate entropy of each class provided in the att list
135     att= calculate_attribute_entropy(att);
136     System.out.println(att);
137

```

```

138 // max gain index
139 int index=0;
140 double gain=0.0;
141 for(Attribute a1:att)
142 {
143     if(gain<a1.getGain())
144     {
145         gain=a1.getGain();
146         index=a1.getIndex();
147     }
148 }
149 System.out.println("index:"+index);
150 // set the index value in BITSET col
151 BitSet temp=kk.getCol();
152 temp.set(index);
153 temp.set(columb_count-1);
154 kk.setCol(temp);
155 // create node for Index Attribute
156 kk.setName(data.get(0).get(index));
157 kk.setIndex(index);
158 for(Attribute a1:att)
159 {
160     if(a1.getIndex()==index)
161     {
162         for(attvalue a2:a1.getValues())
163         {
164             TNode a=new TNode();
165             a.setCol(temp);
166             a.setName(a2.getName());
167
168             if(a2.getEntropy()==0&&(a2.getVal1().size()==0||a2.getVal2().size()==0))
169             {
170                 a.setLeaf(true);
171                 a.setIndex(a2.getIndex());
172                 if(a2.getVal1().size()==0&&a2.getVal2().size()!=0)
173                 {
174                     a.setOutput(tname.get(1));
175                 }
176
177                 if(a2.getVal1().size()!=0&&a2.getVal2().size()==0)
178                 {
179                     a.setOutput(tname.get(0));
180                 }
181             }
182             kk.getNext().put(a2.getName(), a);
183         }
184     }
185 }
186 }
187 }
188 }
189
190
191 ArrayList<TNode>que= new ArrayList<TNode>();
192 que.add(head);
193 int size=1;
194 // print tree :
195 while(que.size()!=0)
196 {
197     System.out.println("\n\n");
198     int next=0;
199     ArrayList<TNode> k= new ArrayList<TNode>();
200
201     for(int i=0;i<size;i++)
202     {
203         for(String s:que.get(i).getNext().keySet())
204         {
205             k.add(que.get(i).getNext().get(s));
206

```

```

207         next++;
208     }
209     que.addAll(k);
210 }
211
212
213
214     for(int i=0;i<size;i++)
215     {
216         if(que.get(0).isLeaf())
217             System.out.print("\t"+que.get(0).getName()+"(output:"+que.get(0).getOutput()+")");
218         else
219             System.out.print("\t"+que.get(0).getName());
220
221
222         que.remove(0);
223     }
224     size=next;
225
226
227 }
228
229
230
231
232
233
234
235 }
236
237
238
239 private static ArrayList<ArrayList<String>>
required_data(ArrayList<ArrayList<String>> data,TNode t,ArrayList<Attribute> att)
{
    // TODO Auto-generated method stub
    if(t.getName()==null)
    {
        return data;
    }
    ArrayList<ArrayList<String>> data1 = new ArrayList<ArrayList<String>>();
    data1.add(data.get(0));
    String name=t.getName();
    for(Attribute a1:att)
    {
        for(attvalue a2:a1.getValues())
        {
            if(name==a2.getName())
            {
                for(Integer k1:a2.getVal1())
                {
                    data1.add(data.get(k1));
                }
                for(Integer k2:a2.getVal2())
                {
                    data1.add(data.get(k2));
                }
            }
        }
    }
    return data1;
}

```

```

275
276 private static void calculate_val1_and_val2(ArrayList<ArrayList<String>> data) {
277     // TODO Auto-generated method stub
278     //clear tvalue
279     for(String k1:tvalue.keySet())
280     {
281         tvalue.put(k1, 0);
282     }
283     int counter=0;
284     for(ArrayList<String> a1:data)
285     {
286         if(counter==0)
287         {
288             counter++;
289             continue;
290         }
291         int value=tvalue.get(a1.get(columb_count-1));
292         tvalue.put(a1.get(columb_count-1), value+1);
293     }
294 }
295
296
297
298
299 private static ArrayList<Attribute>
calculate_attribute_entropy(ArrayList<Attribute> att) {
300     // TODO Auto-generated method stub
301     for(Attribute a1:att)
302     {
303
304         ArrayList<String>tname =new ArrayList<String>();
305         tname.addAll(tvalue.keySet());
306         int val1_count=tvalue.get(tname.get(0));
307         int val2_count=tvalue.get(tname.get(1));
308
309
310         double entropy1=0.0;
311         for(attvalue a2:a1.getValues())
312         {
313
314             int val1=a2.getVal1().size();
315             int val2=a2.getVal2().size();
316
317             a2.setEntropy(entropy(val1, val2));
318
319             double k1=(double)val1+val2;
320
321             double k2=(double)val1_count+val2_count;
322
323             double k3=(k1/k2)*a2.getEntropy();
324
325             entropy1=entropy1+k3;
326         }
327         // attribute entropy
328         a1.setClassentropy(entropy1);
329         // gain
330         a1.setGain((Targetentropy-entropy1));
331     }
332
333
334
335     return att;
336 }
337
338
339
340 public static TNode checktree(TNode head)
341 {
342

```

```

343 Queue<TNode> bb= new LinkedList<TNode>();
344 bb.add(head);
345 TNode kk=null;
346 System.out.println("tree1");
347 System.out.println("size: "+bb.size());
348 while(bb.size()!=0)
349 {
350     kk=bb.poll();
351
352     if(kk.getIndex()==-1)
353     {
354         System.out.println("return kk");
355         return kk;
356     }
357     for(String s:kk.getNext().keySet())
358     {
359         bb.add(kk.getNext().get(s));
360     }
361 }
362
363 return new TNode(-2);
364 }
365
366
367 public static ArrayList<Attribute> attributevalue(ArrayList<ArrayList<String>>
data, TNode t)
368 {
369     //if (!rows.get(counter))
370         //continue;
371     row_count=data.size();
372     int counter =0;
373     BitSet col=t.getCol();
374     col.set(columb_count-1);
375     ArrayList<Attribute> ans= new ArrayList<Attribute>();
376     while (counter<columb_count)
377     {
378         if(col.get(counter))
379         {
380             counter++;
381             continue;
382         }
383         Attribute a1= new Attribute();
384         a1.setName(data.get(0).get(counter));
385         a1.setIndex(counter);
386         ArrayList<attvalue> values= new ArrayList<attvalue>();
387         int counter2 =1;
388
389         while(counter2<row_count)
390         {
391
392             String target=data.get(counter2).get(columb_count-1);
393             String value=data.get(counter2).get(counter);
394             boolean a2=false;
395             // previously seen value in the respective attribute (counter2) (done)
396             ArrayList<String>tname =new ArrayList<String>();
397             tname.addAll(tvalue.keySet());
398             String value1=tname.get(0);
399             String value2=tname.get(1);
400             for(attvalue a:values)
401             {
402                 if(a.getName().equalsIgnoreCase(value))
403                 {
404                     a2=true;
405                     if(target.equalsIgnoreCase(value1))
406                     {
407                         a.getVal1().add(counter2);
408                     }
409                     if(target.equalsIgnoreCase(value2))
410                     {

```

```

411         a.getVal2().add(counter2);
412     }
413 }
414
415
416     }
417
418     // new value in the attribute.+ add the new attvaluein values list (done)
419     if(a2==false)
420     {
421         attvalue a4= new attvalue();
422         a4.setName(value);
423         a4.setIndex(0);
424
425         if(target.equalsIgnoreCase(value1))
426         {
427
428             a4.getVal1().add(counter2);
429         }
430         if(target.equalsIgnoreCase(value2))
431         {
432
433             a4.getVal2().add(counter2);
434         }
435
436         values.add(a4);
437     }
438     counter2++;
439 }
440
441     a1.setValues(values);
442     ans.add(a1);
443     counter++;
444     //System.exit(0);
445 }
446
447 /*
448 for(Attribute at:ans)
449 {
450     System.out.println("\n\n\nname: "+at.getName());
451     for(attvalue ki:at.getValues())
452     {
453         System.out.println(ki);
454     }
455 }
456
457 */
458
459 System.out.println("attribute: "+ans);
460 return ans;
461 }
462
463
464
465
466
467 public static double entropy(int a,int b)
468 {
469     if(a==0||b==0)
470         return 0;
471     if(a==b)
472         return 1;
473
474     double ans=0.0;
475     int n=a+b;
476     double a1=(double)a/(double)n;
477     double a2=logb(a, n);
478     ans=ans+(-1*a1*a2);
479     a1=(double)b/(double)n;

```

```
480         a2=logb(b, n);
481         ans=ans+(-1*a1*a2);
482         return ans;
483     }
484
485     public static double logb(double a, double b)
486     {
487         if (a == 0)
488             return 0;
489         double k1=Math.log(a)/Math.log(2); // gives log a base 2
490         double k2=Math.log(b)/Math.log(2); // gives log n base 2
491         return k1-k2; // log(a/b)=log a- log b
492     }
493
494
495 }
496
```