

Intelligent Systems II

Exercise Sheet 1

Stuart Golodetz

January 26, 2006

1. The tables in Figure 1 show an example data set containing 16 instances over 4 attributes and classified into one of 3 categories (1-3). Use the entropy measure heuristic to construct a decision tree that is consistent with this data set. (You may find the table in Figure 2 useful.)

A	B	C	D	Cl.
N	+	H	F	1
Z	-	M	F	2
N	+	H	T	3
P	+	H	T	3
N	-	L	T	2
P	+	L	T	1
N	-	M	F	2
P	+	M	F	1

A	B	C	D	Cl.
P	-	H	F	1
N	+	M	F	1
P	-	M	T	2
Z	+	M	T	1
P	-	L	F	1
N	-	H	F	1
P	-	H	T	3
Z	-	L	F	3

Figure 1: Tables of data for question 1

b \ a	1	2	3	4	5	6	7	8	9
6	0.431	0.528	0.500	0.390	0.219	0.000			
7	0.401	0.516	0.524	0.461	0.347	0.191	0.000		
8	0.375	0.500	0.531	0.500	0.424	0.311	0.169	0.000	
9	0.352	0.482	0.528	0.520	0.471	0.390	0.282	0.151	0.000
10	0.332	0.464	0.521	0.529	0.500	0.442	0.360	0.258	0.137

Figure 2: Table of the function $-p \log_2 p$ where $p = a/b, a \leq b$.

Answer

Let c^n denote the number of examples (at the root of the current subtree on which we're working) which have $Cl. = n$ and c_i^n denote the number of examples in generated subset E_i which have $Cl. = n$ in what follows.

First of all, we need to calculate the entropy measures for each of the four attributes:

$$\begin{aligned}
 c_{(A=N)}^1 &= 3, c_{(A=N)}^2 = 2, c_{(A=N)}^3 = 1 \\
 c_{(A=Z)}^1 &= 1, c_{(A=Z)}^2 = 1, c_{(A=Z)}^3 = 1 \\
 c_{(A=P)}^1 &= 4, c_{(A=P)}^2 = 1, c_{(A=P)}^3 = 2
 \end{aligned}$$

$$\begin{aligned}
 measure_A &= \sum_{i \in \{A=N, A=Z, A=P\}} \frac{c_i^1 + c_i^2 + c_i^3}{c^1 + c^2 + c^3} H \left(\left\langle \frac{c_i^1}{c_i^1 + c_i^2 + c_i^3}, \frac{c_i^2}{c_i^1 + c_i^2 + c_i^3}, \frac{c_i^3}{c_i^1 + c_i^2 + c_i^3} \right\rangle \right) \\
 &= \frac{6}{16} H \left(\left\langle \frac{3}{6}, \frac{2}{6}, \frac{1}{6} \right\rangle \right) + \frac{3}{16} H \left(\left\langle \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right\rangle \right) + \frac{7}{16} H \left(\left\langle \frac{4}{7}, \frac{1}{7}, \frac{2}{7} \right\rangle \right) \\
 &= \frac{6}{16} (0.500 + 0.528 + 0.431) + \frac{3}{16} \log_2 3 + \frac{7}{16} (0.461 + 0.401 + 0.516) \\
 &= 1.447
 \end{aligned}$$

$$\begin{aligned}
 c_{(B=+)}^1 &= 5, c_{(B=+)}^2 = 0, c_{(B=+)}^3 = 2 \\
 c_{(B=-)}^1 &= 3, c_{(B=-)}^2 = 4, c_{(B=-)}^3 = 2
 \end{aligned}$$

$$\begin{aligned}
 measure_B &= \frac{7}{16} H \left(\left\langle \frac{5}{7}, 0, \frac{2}{7} \right\rangle \right) + \frac{9}{16} H \left(\left\langle \frac{3}{9}, \frac{4}{9}, \frac{2}{9} \right\rangle \right) \\
 &= \frac{7}{16} (0.347 + 0 + 0.516) + \frac{9}{16} (0.528 + 0.520 + 0.482) \\
 &= 1.238
 \end{aligned}$$

$$\begin{aligned}
c_{(C=H)}^1 &= 3, c_{(C=H)}^2 = 0, c_{(C=H)}^3 = 3 \\
c_{(C=M)}^1 &= 3, c_{(C=M)}^2 = 3, c_{(C=M)}^3 = 0 \\
c_{(C=L)}^1 &= 2, c_{(C=L)}^2 = 1, c_{(C=L)}^3 = 1 \\
\text{measure}_C &= \frac{6}{16}H\left(\left\langle\frac{3}{6}, 0, \frac{3}{6}\right\rangle\right) + \frac{6}{16}H\left(\left\langle\frac{3}{6}, \frac{3}{6}, 0\right\rangle\right) + \frac{4}{16}H\left(\left\langle\frac{2}{4}, \frac{1}{4}, \frac{1}{4}\right\rangle\right) \\
&= 2 \times \frac{6}{16}(0.500 + 0 + 0.500) + \frac{4}{16}(0.500 + 0.500 + 0.500) \\
&= 1.125 \\
c_{(D=F)}^1 &= 6, c_{(D=F)}^2 = 2, c_{(D=F)}^3 = 1 \\
c_{(D=T)}^1 &= 2, c_{(D=T)}^2 = 2, c_{(D=T)}^3 = 3 \\
\text{measure}_D &= \frac{9}{16}H\left(\left\langle\frac{6}{9}, \frac{2}{9}, \frac{1}{9}\right\rangle\right) + \frac{7}{16}H\left(\left\langle\frac{2}{7}, \frac{2}{7}, \frac{3}{7}\right\rangle\right) \\
&= \frac{9}{16}(0.390 + 0.482 + 0.352) + \frac{7}{16}(0.516 + 0.516 + 0.524) \\
&= 1.369
\end{aligned}$$

Having done that, we note that attribute C has the lowest entropy measure and make it the splitter at the root node. We now have to do exactly the same as we just did for each of the three child nodes ($C = H$, $C = M$ and $C = L$). This is a task distinguished only by being long and boring, so I got fed up at this point and wrote a program to do it for me (a task distinguished by being long and hard, rather than long and boring, which is, in my view, infinitely preferable): the source code is shown below. The resulting tree is as shown on the attached sheet.

Listing 1: Building a decision tree

```

1 interface Classifier
  {
    int classes();
    int classify(Record r);
    String description();
  }

class Record
  {
11 private Object[] m_fields;
    private Object m_category;

    public Record(Object[] fields, Object category)
    {
      m_fields = fields;
      m_category = category;
    }

    public Object category()
    {
21 return m_category;
    }

    public Object[] fields()
    {
      return m_fields;
    }
  }

import java.util.*;

31 class Tree
  {
    abstract static class Node {}

    static class BranchNode extends Node
    {
      Classifier m_classifier;
      Node[] m_children;
    }

41 static class LeafNode extends Node

```

```

    {
        Object m_classification;

        public LeafNode(Object classification)
        {
            m_classification = classification;
        }
    }
51 BranchNode m_root;

    public Tree(Record[] records, LinkedList<Classifier> classifiers)
    {
        m_root = new BranchNode();
        build_subtree(m_root, records, classifiers);
    }

    public Object classify_record(Record r)
61 {
        return classify_against_subtree(m_root, r);
    }

    public void output()
    {
        output_subtree(m_root, 0);
    }

    // Copied from my third-year project code.
71 private static <T> T[] array_from_list(LinkedList<T> list)
    {
        // Check the preconditions.
        if(list == null || list.isEmpty()) throw new java.lang.Error();

        T[] arr = (T[]) java.lang.reflect.Array.newInstance(list.getFirst().getClass(),
                                                             list.size());

        // Copy the list elements into the array.
        int i = 0;
81 for(T t: list) arr[i++] = t;

        return arr;
    }

    private boolean build_subtree(BranchNode node, Record[] records,
                                  LinkedList<Classifier> classifiers)
    {
        HashSet<Object> categories = new HashSet<Object>();
        for(Record r: records) categories.add(r.category());
91 node.m_classifier = choose_classifier(records, classifiers, categories);

        if(node.m_classifier == null) return false;

        int childCount = node.m_classifier.classes();
        node.m_children = new Node[childCount];
        for(int i=0; i<childCount; ++i)
        {
            LinkedList<Record> childRecordsList = new LinkedList<Record>();
101 for(Record r: records)
            {
                if(node.m_classifier.classify(r) == i)
                {
                    childRecordsList.add(r);
                }
            }

            if(childRecordsList.isEmpty())
            {
111 node.m_children[i] = null;
                continue;
            }

            Record[] childRecords = array_from_list(childRecordsList);

            // If all the child records have the same category, make a leaf, else recurse.
            Object category = find_common_category(childRecords);

```

```

    if(category != null)
    {
121      node.m_children[i] = new LeafNode(category);
    }
    else
    {
        // Our child chooses from any of our classifiers except the one we've used ourselves.
        LinkedList<Classifier> childClassifiers = new LinkedList<Classifier>();
        for(Classifier c: classifiers) if(c != node.m_classifier) childClassifiers.add(c);

        BranchNode bn = new BranchNode();
131      if(build_subtree(bn, childRecords, childClassifiers) == true)
        {
            node.m_children[i] = bn;
        }
        else
        {
            node.m_children[i] = new LeafNode(-1);
        }
    }
}

141 return true;
}

private Classifier choose_classifier(Record[] records, LinkedList<Classifier> classifiers,
                                    HashSet<Object> categories)
{
    int categoryCount = categories.size();

    double bestMetric = Double.MAX_VALUE;
    Classifier bestClassifier = null;
151
    for(Classifier classifier: classifiers)
    {
        System.out.print(classifier.description() + ": ");

        double metric = 0;

        // For each possible value of a given field.
        for(int i=0, classes=classifier.classes(); i<classes; ++i)
        {
161          int[] count = new int[categoryCount];

          // Count the number of records with the given field value and each possible outcome.
          int j = 0;
          for(Object category: categories)
          {
              for(Record r: records)
              {
                  if(classifier.classify(r) == i && r.category().equals(category)) ++count[j];
171              }

              ++j;
          }

          // Count the total number with that field value.
          double total = 0;
          for(int k: count) total += k;

          // Modify the metric.
          double h = H(count, total);
181          metric += (total / records.length) * h;

          if(i != 0) System.out.print("+ ");
          System.out.print("(" + count[i] + "/" + records.length + "*" + h + " ");
        }

        System.out.println(" = " + metric);

        if(metric < bestMetric)
        {
191          bestMetric = metric;
          bestClassifier = classifier;
        }
    }
}

```

```

    return bestClassifier;
}

private Object classify_against_subtree(Node node, Record r)
{
201   if(node instanceof LeafNode)
    {
        LeafNode ln = (LeafNode)node;
        return ln.m_classification;
    }
    else if(node instanceof BranchNode)
    {
        BranchNode bn = (BranchNode)node;
        return classify_against_subtree(bn.m_children[bn.m_classifier.classify(r)], r);
    }
211   else return null;
}

private Object find_common_category(Record[] records)
{
    if(records == null) return null;

    Object category = records[0].category();

221   for(int i=1, len=records.length; i<len; ++i)
    {
        if(!records[i].category().equals(category)) return null;
    }

    return category;
}

private double H(int[] count, double total)
{
231   double ret = 0;

    for(int c: count)
    {
        double d = c/total;
        if(d > 0) ret += -d * (Math.log(d)/Math.log(2));
    }

    return ret;
}

241 private void output_subtree(Node n, int depth)
{
    if(n != null)
    {
        for(int i=0; i<depth; ++i) System.out.print(" ");

        if(n instanceof LeafNode)
        {
251         LeafNode ln = (LeafNode)n;
            System.out.println(ln.m_classification);
        }
        else if(n instanceof BranchNode)
        {
            BranchNode bn = (BranchNode)n;
            System.out.println(bn.m_classifier.description() + "?");

            for(Node c: bn.m_children) output_subtree(c, depth+1);
        }
    }
261 }

public static void main(String[] args)
{
    Record[] records = new Record[]
    {
        new Record(new Object[] {"N", "+", "H", "F"}, new Integer(1)),
        new Record(new Object[] {"Z", "-", "M", "F"}, new Integer(2)),
        new Record(new Object[] {"N", "+", "H", "T"}, new Integer(3)),
        new Record(new Object[] {"P", "+", "H", "T"}, new Integer(3)),
        new Record(new Object[] {"N", "-", "L", "T"}, new Integer(2)),
    }
}

```

```

271     new Record(new Object[] { "P", "+", "L", "T"}, new Integer(1)),
        new Record(new Object[] { "N", "-", "M", "F"}, new Integer(2)),
        new Record(new Object[] { "P", "+", "M", "F"}, new Integer(1)),
        new Record(new Object[] { "P", "-", "H", "F"}, new Integer(1)),
        new Record(new Object[] { "N", "+", "M", "F"}, new Integer(1)),
        new Record(new Object[] { "P", "-", "M", "T"}, new Integer(2)),
        new Record(new Object[] { "Z", "+", "M", "T"}, new Integer(1)),
        new Record(new Object[] { "P", "-", "L", "F"}, new Integer(1)),
        new Record(new Object[] { "N", "-", "H", "F"}, new Integer(1)),
        new Record(new Object[] { "P", "-", "H", "T"}, new Integer(3)),
281     new Record(new Object[] { "Z", "-", "L", "F"}, new Integer(3))
    };

LinkedList<Classifier> classifiers = new LinkedList<Classifier>();
classifiers.add(new Classifier()
{
    public int classes() { return 3; }

    public int classify(Record r)
291     {
        if(r.fields()[0].equals("N")) return 0;
        else if(r.fields()[0].equals("Z")) return 1;
        else if(r.fields()[0].equals("P")) return 2;
        else throw new Error();
    }

    public String description() { return "A"; }
});
classifiers.add(new Classifier()
301 {
    public int classes() { return 2; }

    public int classify(Record r)
    {
        if(r.fields()[1].equals("+")) return 0;
        else if(r.fields()[1].equals("-")) return 1;
        else throw new Error();
    }

    public String description() { return "B"; }
311 });
classifiers.add(new Classifier()
{
    public int classes() { return 3; }

    public int classify(Record r)
    {
        if(r.fields()[2].equals("H")) return 0;
        else if(r.fields()[2].equals("M")) return 1;
        else if(r.fields()[2].equals("L")) return 2;
321     else throw new Error();
    }

    public String description() { return "C"; }
});
classifiers.add(new Classifier()
{
    public int classes() { return 2; }

    public int classify(Record r)
331     {
        if(r.fields()[3].equals("F")) return 0;
        else if(r.fields()[3].equals("T")) return 1;
        else throw new Error();
    }

    public String description() { return "D"; }
});

341 Tree tree = new Tree(records, classifiers);
tree.output();

for(Record r: records)
{
    System.out.println(tree.classify_record(r));
}

```

```
}  
}  
}
```

For what it's worth, the output from the program (including the calculated metrics) is as follows:

```
A: 6/16*1.4591479170272448 + 3/16*1.584962500721156 + 7/16*1.3787834934861758  
  = 1.4475787161706355  
B: 7/16*0.863120568566631 + 9/16*1.5304930567574824 = 1.238517593173985  
C: 6/16*1.0 + 6/16*1.0 + 4/16*1.5 = 1.125  
D: 9/16*1.224394445405986 + 7/16*1.5566567074628228 = 1.3697591850558521  
A: 3/6*0.9182958340544896 + 0/6*0.0 + 3/6*0.9182958340544896 = 0.9182958340544896  
B: 3/6*0.9182958340544896 + 3/6*0.9182958340544896 = 0.9182958340544896  
D: 3/6*0.0 + 3/6*0.0 = 0.0  
A: 2/6*1.0 + 2/6*1.0 + 2/6*1.0 = 1.0  
B: 3/6*0.0 + 3/6*0.0 = 0.0  
D: 4/6*1.0 + 2/6*1.0 = 1.0  
A: 1/4*0.0 + 1/4*0.0 + 2/4*0.0 = 0.0  
B: 1/4*0.0 + 3/4*1.584962500721156 = 1.188721875540867  
D: 2/4*1.0 + 2/4*1.0 = 1.0
```

```
C?  
D?  
  1  
  3  
B?  
  1  
  2  
A?  
  2  
  3  
  1
```

```
1  
2  
3  
3  
2  
1  
2  
1  
1  
1  
1  
2  
1  
1  
1  
1  
3  
3
```

Note that the values calculated for the first few are the same as the ones we calculated by hand.

2. Consider the entropy measure function

$$H(\langle P_1, \dots, P_n \rangle) = \sum_{i=1}^n -P_i \log_2 P_i$$

(a) For the case where $n = 2$ we can re-write this equation as a function of one variable:

$$h(p) = -p \log_2 p - (1 - p) \log_2(1 - p)$$

Write down expressions for the first and second derivatives of h , and hence show that h has a maximum value when $p = \frac{1}{2}$. (You may wish to recall that the derivative of a product of functions is given by $(uv)' = u'v + uv'$, and that $\frac{d \log x}{dx} = \frac{1}{x}$.)

Answer

We first observe that the given equation for the derivative of $\log x$ is only true for base e (i.e. for $\ln x$), so we have to do a bit more work:

$$\log_2 x = \ln x / \ln 2$$

$$\Leftrightarrow \frac{d \log_2 x}{dx} = \frac{1}{x \ln 2}$$

Given this, we compute:

$$\begin{aligned} h'(p) &= -\left(p \times \frac{1}{p \ln 2} + \log_2 p\right) - \left((1 - p) \times \frac{-1}{(1-p) \ln 2} - \log_2(1 - p)\right) \\ &= -\left(\frac{1}{\ln 2} + \log_2 p\right) - \left(\frac{-1}{\ln 2} - \log_2(1 - p)\right) \\ &= -\frac{1}{\ln 2} - \log_2 p + \frac{1}{\ln 2} + \log_2(1 - p) \\ &= \log_2(1 - p) - \log_2 p \\ &= \log_2\left(\frac{1-p}{p}\right) \end{aligned}$$

$$\begin{aligned} h''(p) &= \frac{p}{(1-p) \ln 2} \times \frac{p \cdot -1 - (1-p) \cdot 1}{p^2} \\ &= \frac{-p}{p^2(1-p) \ln 2} \\ &= \frac{-1}{p(1-p) \ln 2} \end{aligned}$$

Now h has a turning point where $h'(p) = 0$, i.e. where:

$$\begin{aligned} \log_2\left(\frac{1-p}{p}\right) &= 0 \\ \Leftrightarrow \frac{1-p}{p} &= 1 \\ \Leftrightarrow 1 - p &= p \\ \Leftrightarrow p &= \frac{1}{2} \end{aligned}$$

To show the turning point is a maximum, we classify it using the sign of $h''\left(\frac{1}{2}\right)$:

$$h''\left(\frac{1}{2}\right) = \frac{-1}{\frac{1}{2}\left(1-\frac{1}{2}\right) \ln 2} = -5.77 \dots < 0$$

Turning points where the second derivative is negative are maxima, thus h has a maximum value when $p = \frac{1}{2}$.

(b) Show that

$$H\left(\left\langle\frac{1}{n}, \dots, \frac{1}{n}\right\rangle\right) = \log_2 n$$

Answer

$$\begin{aligned} & H\left(\left\langle\frac{1}{n}, \dots, \frac{1}{n}\right\rangle\right) \\ &= \sum_{i=1}^n -\frac{1}{n} \log_2 \frac{1}{n} \\ &= \left(-\frac{1}{n} \log_2 \frac{1}{n}\right) \sum_{i=1}^n 1 \quad \{\text{this line is obvious, but is included for completeness}\} \\ &= \left(-\frac{1}{n} \log_2 \frac{1}{n}\right) n \quad \{\text{ditto}\} \\ &= -1 \log_2 \frac{1}{n} \\ &= \log_2 \left(\frac{1}{n}\right)^{-1} \\ &= \log_2(n) \quad \square \end{aligned}$$

3. We claim that using a linear activation function in a neural network will always produce outputs that are linear sums of their inputs. Use an inductive argument to prove this assertion.

Answer

We'll assume we've got a linear activation function and use induction over the layer number to show that the output is a linear sum of the inputs:

Base Case (Input Layer)

Each input is clearly a linear sum of the inputs. Specifically, if the inputs are $input_1, \dots, input_n$, then (clearly):

$$input_i = 1 \times input_i + \sum_{j=0, j \neq i}^n 0 \times input_j$$

Inductive Hypothesis

The values produced by the layer k are each a linear sum of the inputs.

Inductive Step

We want to show that the hypothesis implies that the values produced by layer $k + 1$ are each a linear sum of the inputs.

Let the values produced by a layer m be denoted $a_{m,i}$ for various i . Then since each value produced by layer k is a linear function of the inputs:

$$\forall i \exists b_1, \dots, b_n \cdot a_{k,i} = \sum_{j=1}^n b_j \times input_j$$

Now if $W_{(k,j),(k+1,i)}$ is the weight on the link from node j of layer k to node i of layer $k + 1$, then:

$$\begin{aligned}
 a_{k+1,i} &= g\left(\sum_j W_{(k,j),(k+1,i)} a_{k,j}\right) \\
 &= g\left(\sum_j W_{(k,j),(k+1,i)} \left[\sum_{p=1}^n b_{p,j} \times input_p\right]\right) \quad \{\text{for some } b_{p,j}\} \\
 &= g\left(\sum_{p=1}^n \left[\left(\sum_j W_{(k,j),(k+1,i)} b_{p,j}\right) \times input_p\right]\right) \\
 &= g\left(\sum_{p=1}^n [f(p) \times input_p]\right) \quad \{\text{where } f(p) = \sum_j W_{(k,j),(k+1,i)} b_{p,j}\}
 \end{aligned}$$

Now if $g(x)$ is a linear activation function then it's of the form $qx + r$ for some q, r . So:

$$\begin{aligned}
 &g\left(\sum_{p=1}^n [f(p) \times input_p]\right) \\
 &= q \sum_{p=1}^n (f(p) \times input_p) + r \\
 &= \sum_{p=1}^n (F(p) \times input_p) + r \quad \{\text{where } F(p) = q \times f(p)\}
 \end{aligned}$$

This is quite definitely a linear sum of the inputs, so the values produced by layer $k + 1$ are each a linear sum of the inputs.

The output layer is as much a layer as any other, so the values it produces (the outputs of the network) must be linear sums of the inputs.

4. A simple neural network consists of two inputs, two hidden nodes, and one output node (cf. slide 8 in lecture slides set 2). The weights are all initially 1, and you are asked to compute the updated weights using one epoch (i.e. pass from right to left) of the Delta rule after an input of $[-0.5, 0.25]$ is applied in the following two cases.
 - (a) Using a step activation function, the learning rate set to $\rho = 0.5$, thresholds initially 0, and required output 1.
 - (b) Now using the same starting conditions and learning rate, but assume that the activation functions are all sigmoids with no thresholds and required output 0.5. (Recall that for a sigmoid we have $g(x) = 1/(1 + e^{-x})$ and that $g'(x) = g(x)(1 - g(x))$.)

Was choosing the same initial weight values throughout a sensible move?

Answer

See separate sheets for diagrams and answers. Choosing the same initial weight values throughout was sensible from the point of view of making it easier to calculate the new values: a lot of them are the same as each other. Other than that, though, it seems to have been rather ineffective: the weights haven't really changed an awful lot.

5. Consider the Umbrella World example from the book and notes. A new guard starts, and observes the sequence of umbrellas over the first few days given in Figure 3. (The same conditional probabilities apply as before, as also summarised in the figure.) What do you estimate the probability to be that (a) it is raining on day 4 and (b) it was raining on day 2?

Day	1	2	3	4	R_{t-1}	$P(R_t)$	R_t	$P(U_t)$
Umbrella	Yes	No	Yes	Yes	T	0.7	T	0.9
					F	0.3	F	0.2

Figure 3: Umbrella World Example

Answer

See separate sheet.