CONCEPT FORMATION FROM VERY LARGE TRAINING SETS

Richard A. O'Keefe*

Department of Artificial Intelligence
University of Edinburgh, Edinburgh

Abstract

This paper proposes an alternative to Quinlan's algorithm for forming classification trees from large sets of examples. My algorithm is guaranteed to terminate. Quinlan's algorithm is usually faster.

I. The Nature of the Problem.

We have a population of objects which we want to classify into two** groups. We have a set of attributes, each with a snail finite number of distinct values, and a set of examples whose attributes have been measured and which have already been classified. Our goal is to find a rule, based on these examples, which we can use to classify other members of the population.

In general this will lead us to statistical methods such as cluster analysis (Everitt, 1974, Kendall, 1975, Sturt, 1981a, Sturt, 1981b). The larger our collection of examples, the more likely it is that some of them are misclassified. (Sturt, 1981a) provides an excellent illustration of how improving the fit of a rule to the training set (beyond a certain point) can make It perform worse on the rest of the population.

Even so, there are interesting tasks where the domain is formal, and we can be sure that we have all the information we need and that our classifications are correct. There are, however, interesting problems where the domain is formal rather than real, and we can be sure that all the relevant information is available and our classifications are correct. Chess positions and algebraic equations are two such domains.

*This work was supported by a Commonwealth Scholarship. Computing was done on the SERC Dec-10 under grant CR/C/20826

**ID3 and ray algorithm are both presented in terms of two categories. The information heuristic generalises to any fixed number of categories, and so do the algorithms.

The input to a concept formation algorithm, then, is

a set of attributes A_j, $j = 1..M$

attribute j has values $1..|A_j|$

a set of examples E_i, $i = 1..N$

for each example, its classification C_{i0} in $\{1,2\}$

and its attributes C_{ij} in $\{1..|A_j|\}$

The training set can thus be represented by an $N \times (M+1)$ integer matrix.

II. The Form of a Rule.

(Quinlan, 1979, Sturt, 1981a) and this paper represent a discriminant function as a **decision tree**. The description here is slightly different to make the algorithm clearer.

Each node of a decision tree has three labels:

```
        E-Set(Node) is a set of examples
        Class(Node) is empty, 1, 2, or mixed
        Attrb(Node) is undefined, 1, ..., or M
and some sons
        Son(Node,i) for i = 1..|A_{Attrb(Node)}|
```

We use a decision tree to classify examples by applying the following algorithm to its attribute measurements:

```
set Node = the root of the tree.
while Class(Node) is mixed do
    set k = the value of attribute Attrb(Node).
    set Node = Son(Node, k).
od.
if Class(Node) is empty, fail.
report Class(Node) as the classification.
```

(Quinlan, 1979) suggests returning an arbitrary classification when Class(Node) is empty. Whether guessing is a good idea or not depends on what you want to do with the rule. Note that Class(Node) is redundant; it is empty, 1, 2, or mixed according as E-Set(Node) is empty, all class 1, all class 2, or some of each.

III. Building a Rule.

The algorithm for building a decision tree is the usual one:

```
create a tree with one Node.
set E-Set(Node) = the entire training set.
set Class(Node) according to E-Set(Node).
set Attrb(Node) = undefined.

while there is a Node with Class(Node) = mixed
      and Attrb(Node) = undefined do
    select an attribute $A_j$ not tested by any
          ancestor of this Node.
    set Attrb(SubNode) = j.
    for k = 1 .. $|A_j|$ do
        create a new SubNode.
        add SubNode as Son(Note, k).
        set E-Set(SubNode) = the examples in
            E-Set(Node) whose $A_j$ value is k.
        set Class(SubNode) from E-Set(SubNode).
        set Attrb(SubNode) = undefined.
    od.
od.
```

It is clear that this algorithm must terminate, that each mixed Node will finally be expanded, and that the order of expansion doesn't matter. The only room for choice in this algorithm is the selection of an attribute. (Quinlan, 1979) suggests using the information heuristic to choose an attribute. The entropy of a set of examples is defined as

$$H = - p1.\log'(p1) - p2.\log'(p2)$$
$$\log'(0) = 0,\ \log'(X) = \log_2 (X) \text{ for } X > 0.$$

where p1 (p2) is the proportion of examples in class 1 (class 2)*. This is, roughly speaking, the number of binary tests needed to discriminate between the two classes. If we test attribute A_j, the average entropy of the subtrees is

$$H_j = (\text{sum over } k = 1..|A_j| \text{ of } H_{jk})/|A_j|$$
$$H_{jk} = -p_{jk1}.\log'(p_{jk1}) - p_{jk2}.\log'(p_{jk2})$$
$$p_{jkc} = \text{the proportion of E-Set(Node) with } C_j = k \text{ and } C_0 = c$$

The information heuristic is to select the attribute with minimum H_j. It is a heuristic because we are assuming that the training set is typical of the population. For 2 classes, H'_j can be computed in

$$o(2.M'.N') \text{ time and } o(2.K') \text{ space,}$$
$$\text{where } K' = \text{sum over } j = 1..M' \text{ of } |A_j|$$

The same space can be used for all attribute selections, since K' can never exceed its initial value.

*this Is where the generalisation to more than two categories is made.

IV. Quinlan's Iterative Dichotomiser

The algorithm described above is normally regarded as suitable only when all the examples (and the rule) can be fitted into main memory. Real problems sometimes don't fit. (Quinlan, 1979) used chess end-games. His algorithm makes concept formation practical for such problems. In outline, it is

```
select a sample of the training set.
do
    Infer a rule from the sample.
    check the rule against the whole training set.
    exit when there are no mistakes.
    select a new sample from the old sample and
    the exceptions which were just discovered,
od.
```

In practice, ID3 works very well. Each iteration requires only one pass over the training set, and only changes data in main memory. If ID3 terminates, it has indeed found a rule. However, ID3 needs _some_ of the examples to be in memory. A rule which will fit in memory may be missed because the examples it is inferrable from will not fit. It would be better if none of the examples had to be kept in memory. The resampling method as describe in (Quinlan, 1979) may fail to realise that no possible sanple will work, and loop forver.

It would be acceptable if ID3 spotted that it couldn't cope and stopped. In practice a limit on the size of the tree or the number of iterations is enough. However, there is no guarantee that a problem which exceeds these limits doesn't have a simple rule.

V. Rule Building is like Radix Sort.

Quinlan made two contributions to concept formation: the information heuristic, and his iterative sampling method. We can retain the former, and look for another way of handling large training sets.

The only difference between the Concept Formation algorithm and radix sort is that the sons of a node nay test different attributes, while in a left-to-right radix sort they must test the sane attribute. We adapt the data structure of radix sort (a pair of randomly accessed files) to the abstract algorithn of Section 3. This is the only new idea in this paper. As it keeps no examples in memory, if it fails to find a rule, that Is a guarantee that the information heuristic will never generate a rule that fits in memory. As it adds no extra choices to the abstract algorithm, it must terminate.

We hold the examples in two files, which are always permutations of each other. An example set is represented by a triple

<WhichFile, First, Next>

where the WhichFile field says which of the two files the set may be found in, First is the index of the first example in this set, and Next is the index of the first example following this set. An empty set is represented by a triple with First ■ Next.

Selecting an Attrb for a Node requires one pass over E-Set(Node), where we calculate the counts from which the p_{jkc} are derived. We initialise the E-Sets for the sons as follows:

```
set <filel.First,Next> « E-Set(Node).
set file2 - other_file(filel).
set first - First.
for k - 1,.|Attrb(Node)| do
    set E-Set(Son(Node,k)) - <file2 ,first,first>.
    set first    first+Nk.
od.
```

where Mk, the number of examples with attribute value k, was determined in the first pass. We then make one more pass over the data, doing

```
for place = First..Next-1 do
    read an example from filel(place).
    set k = the attribute value.
    set <file2,first,last> - E-Set(Son(Node,k)).
    write the example to file2(last).
    set E-Set(Son(Node,k)) - <file2 ,first,last+l>.
od •
```

As we permute the examples into the same range of records, the E-Set labels of all existing nodes remain valid.

Main memory is needed to hold the counts for attribute selection (less than 500 cells for Quinlan's examples) and to hold the rule. If the rule will not fit in main memory, this algorithm will stop. We are assured that the algorithm could not find any simpler rule.

This algorithn nay take as many passes over the training set as there are levels in the decision tree, so the total cost of finding a rule is proportional to the product

(depth of decision tree) x
(total number of attribute values) x
(number of examples in training set)

A pass of my algorithm reads a file twice and writes another once. A pass of ID3 reads the data file once. When ID3 discovers a simple rule, it usually takes fewer passes than the modified radix sort. But the point of the modified radix sort is to cope with *any* sort of data. I doubt that it can be done nore cheaply.

REFERENCES

Everitt, 3 S. Cluster Analysis. : Heinemann 1974.

Kendall, *U* Multivariate Analysis. : Charles Criffin &'co 1975.

Quinlan, J.R. Discovering Rules by Induction from Large Collections of Examples, pages 168-201. Edinburg"h "University pYess," 1979.

Sturt, E. Computerized Construction in Fortran of a Discriminant Function for Categorical Data. Applied Statistices 1981,20. 213-222.

Sturt, E. An Algorithm to Construct a Discriminant Function in Fortran for Categorical Data. Applied Statistics, 1981 , 30), 313-325. This is the'source'code" of Algorithm AS 165.