01 Mar 1993

Dynamic ID3: A Symbolic Learning Algorithm For Many-valued Attribute Domains

Roger Gallion

Chaman Sabharwal
Missouri University of Science and Technology, chaman@mst.edu

Daniel C. St. Clair
Missourui University of Science and Technology

William E. Bond
Missouri University of Science and Technology

Recommended Citation

R. Gallion et al., "Dynamic ID3: A Symbolic Learning Algorithm For Many-valued Attribute Domains," *Proceedings of the ACM Symposium on Applied Computing*, pp. 14 - 20, Association for Computing Machinery (ACM), Mar 1993.
The definitive version is available at https://doi.org/10.1145/162754.162766

DYNAMIC ID3: A SYMBOLIC LEARNING ALGORITHM FOR MANY-VALUED ATTRIBUTE DOMAINS*

Roger Gallion
McDonnell Douglas Missile Systems Company
E-mail: gallion@umrgec.eec.umr.edu

Daniel C. St. Clair
University of Missouri-Rolla
Engineering Education Center
E-mail: stclair@umrgec.eec.umr.edu

Chaman L. Sabharwal
University of Missouri-Rolla
Engineering Education Center
E-mail: Chaman@umrvmb.umr.edu

W.E. Bond
McDonnell Douglas Research Laboratories

Abstract

Quinlan's ID3 machine learning algorithm induces classification trees (rules) from a set of training examples. The algorithm is extremely effective when training examples are composed of attributes whose values are taken from small discrete domains. The classification accuracy of ID3-produced trees on domains whose attributes are many-valued tends to be marginal due to the large number of possible values which may be associated with each attribute. Attempts to solve this problem by a priori grouping of attribute values into distinct subsets has met with limited success.

The dynamic ID3 algorithm improves the performance of ID3 on this type of problem by grouping many-valued attributes dynamically as the tree is built. Experimental results are provided which compare the performance of dynamic ID3 with standard ID3 and ID3 in which a priori grouping has been used.

Introduction

Quinlan's ID3 machine learning algorithm [Quinlan 1986] induces classification rules from a set of training examples, T. Training examples are taken from a domain described by an attribute space A, viz;

$$A = (A_1, A_2, ..., A_n)$$

where each A_i is an attribute with an associated set of possible values. Each training example $t_i \in T$ is an ordered tuple of values;

$$t_i = (a_{i1}, a_{i2}, ..., a_{in}, c_i),$$

where $a_{ij} \in A_j$. The classification of example t_i is c_i where $c_i \in C$, the set of all possible classifications for the domain.

The knowledge structure for ID3 classification rules is a decision tree such as the one shown in Figure 1. The rule for the path connecting node N_1 to node N_5 can be written as follows:

$$(V(N_1.a) = N_1.v_2) \wedge (V(N_3.a) = N_3.v_1) \Rightarrow N_5.C$$

The notation $N_i.a$ denotes the name of the attribute associated with node N_i while $N_i.v_k$ denotes the value associated with branch k of node N_i. This rule states that for a given set of attribute values;

If the value of attribute $N_1.a$ equals the value associated with N_1's second branch

and the value of attribute $N_3.a$ equals the value associated with N_3's first branch,

then the set of classification(s), $N_5.C$, at node N_5 apply.

Once the decision tree has been constructed, the rules its branches represent can be used to classify unseen test examples. The application of such trees include the automated design and maintenance of diagnostic and situation assessment systems [St. Clair et al 1990].

Trees constructed by ID3 often have poor classification accuracies if the attribute values come from domains with a large number of values. These "many-valued" attributes not only include attributes from continuous domains but attributes from discrete domains in which there are a large number of values. The problem arises due to the large number of possible combinations of attribute values which can occur in both training and test examples. Such a large number of possible combinations makes it likely that the exact combination of attribute values will not be found along a path in the tree when the tree is being used to classify new instances. That is, a branch may not be present for a specific attribute value. As a result, the tree may be unable to classify these instances.

*Work supported by the McDonnell Douglas Independent Research and Development program and Statement of Work WS-MDRL-4062 with the University of Missouri-Rolla Engineering Education Center in St. Louis. Export Authority: 22 CFR 125.4(b) (13).

ACM-SAC '93/2/93/IN, USA
© 1993 ACM 0-89791-568-2/93/0002/0014...$1.50

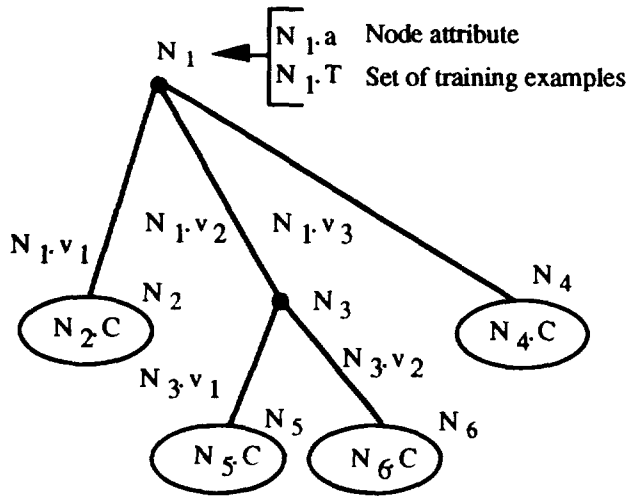

Figure 1. ID3 Decision Tree

Traditionally, datasets containing many-valued attributes have been preprocessed in order to group both training and test attribute values into a smaller number of distinct groups, thereby simplifying training and improving classification accuracy [Lebowitz 1985, Hacke et al 1992]. Another approach, called "nearest neighbor," simply selects the branch with the nearest attribute value when the tree is used for classification [St.Clair et al 1992]. The results of these efforts have provided limited improvements in classification accuracy. The algorithm described in this paper dynamically groups many-valued attributes, at each interior node, during the tree construction process. Results of numerical experiments are included which suggest the new approach often provides an improvement in classification accuracy over preprocessing approaches.

ID3

The ID3 algorithm recursively constructs a decision tree like the one shown in Figure 1. Starting at the root node, N_1, the algorithm uses the set of training examples currently available, $N_1.T = T$, to select the attribute to be used as the branching attribute, $N_1.a$. Once this attribute is chosen, each unique value of $N_1.a$ becomes a branch of the tree. In Figure 1, $N_1.a$ has three values; $N_1.v_1$, $N_1.v_2$, and $N_1.v_3$. At the end of each new branch is a new node, N_i. Node N_1 has essentially partitioned the set of training examples into three disjoint subsets. One of these subsets, $N_i.T$, is associated with each N_i.

The process is repeated at each leaf node N_i. The next branching attribute $N_i.a$ is chosen based on the subset of training examples, $N_i.T$. The set of attributes available at N_i is denoted by $N_i.A$ and is computed by $N_i.A = A - \bigcup_k \{N_k.a\}$ for all nodes N_k lying on the path connecting N_1 and N_i. A leaf node N_i is not expanded if $N_i.A = \{\}$ or if $N_i.C$ contains only a single conclusion.

At each node, the choice of branching attribute is made using the attributes in $N_i.A$ and the training examples in $N_i.T$. In order to determine which of the attributes, A_j, best partitions the training examples, ID3 uses an information-theoretic measure based on entropy [Shannon 1948, Quinlan 1986]. The objective is to build a wide-shallow tree which provides a good generalization of the training data.

Figure 2 shows pseudocode for the ID3 algorithm. The algorithm starts at the root node, N_1 and continues until no more node expansion is possible. The *group_values* procedure is shown in detail since it must be modified for dynamic data partitioning.

The recursive procedure *process_node* first determines the set of class attribute values for the set of examples stored at the current node. If the set of class values is a singleton set or if there are no remaining attributes to use as the branching attribute, then the current node is a leaf node and the procedure returns.

If the node is not a leaf node, then the set of attribute values occurring in the node's training examples are stored by the *group_values* procedure. The *select_branching_attribute* procedure determines which attribute yields the greatest

```
process_node(N_k)
 N_k.C = {}                          /* Set of c_i occurring at N_k */
 N_k.N = {}                          /* Set of child nodes at N_k */
 foreach t_i ∈ N_k.T
        N_k.C = N_k.C ∪ {c_i}
 end
 if ( |N_k.A| > 0 ∧ |N_k.C| > 1 )
        group_values(N_k)
          /*Place values in groups for entropy calculation*/
        select_branching_attribute(N_k)
             /* Using entropy, select branching attribute*/
        set_child_nodes(N_k)
                        /*Build set of child nodes N_k.N*/
        foreach N_j ∈ N_k.N
                        /*Select child node to process*/
          process_node(N_j)
        end
 end
 return

group_values(N_k)
 foreach A_j ∈ N_k.A
        N_k.V_j = {}
              /* Set of branching values for A_j ∈ N_k.T */
        foreach t_i ∈ N_k.T
          N_k.V_j = N_k.V_j ∪ {a_ij}
          /*Get set of values for attribute A_j at node N_k*/
        end
 end
return
```

Figure 2. ID3 Pseudocode

information gain. This attribute will be used as the branching attribute.

Once the branching attribute has been identified, the *set_child_nodes* procedure creates a child node for each unique value of the branching attribute at the parent node. The set of training examples and the set of available attributes are generated for each child node. Finally, the *process_node* procedure is called for each child node.

Dynamic ID3

One major disadvantage of grouping the values of an attribute before training is that the groups remain fixed throughout training even though the set of data values being split at each node is constantly changing. Dynamic ID3, DID3, performs dynamic grouping of attribute values at each node. Hence, at node N_i, the set of values in $N_i.T$ is grouped and used to determine the branching attribute at that node. The rationale is to perform groupings as they are needed using only the training data applicable at the current node.

The psuedocode for DID3's *group_values* procedure is given in Figure 3. The major implementation difference between the two algorithms occurs in the *define_groups* procedure where *attribute-value* groupings are defined. In ID3, *group_values* assigns $N_k.V_j$ the set of training example values that correspond to attribute A_j which occur at node N_k. For attributes which are not many-valued, DID3 generates the same set of values that ID3 generates. For many-valued attributes, DID3 calls a procedure to group attribute values. Techniques for grouping values are described in the next section.

```
group_values(N_k)
    foreach A_j ∈ N_k.A
            N_k.V_j = { }
            if ( A_j is "many-valued" )
               define_groups(N_k)
                /*Procedure defines groups. Returns N_k.V_j*/
            else
               foreach t_i ∈ N_k.T
                  N_k.V_j = N_k.V_j ∪ {a_ij}
               /*Get set of values for each attribute at N_k*/
               end
            end
    end
return
```

Figure 3. Group_values Procedure for DID3

Another difference between ID3 and DID3 occurs when the resultant decision tree is used to classify an example. The groups formed by DID3 must be remembered since this information is required when classifications are performed by the tree. In ID3, the tree is traversed by taking the branch whose value exactly matches the example's value for the branching attribute. In DID3, branch values are groups of values as well as single values. Therefore interval and set membership as well as single value equality must be tested in DID3 trees.

DID3 Grouping Techniques

There are numerous techniques for grouping many-valued attributes. Three techniques were used in this research. Two of the techniques can be used to preprocess data for ID3. They group the data based on the values of an attribute. The third is a new technique which utilizes both attribute and classification values.

The first technique, called the bucket algorithm (BA), is an ad hoc technique which has met with surprising success. The BA uses the minimum and maximum training values of an attribute to construct N equal-sized intervals (buckets). The first and last buckets are defined as open-ended intervals in order to classify test values which may fall outside the minimum and maximum attribute values in the training set.

The second technique was originally used by M. Lebowitz to preprocess ID3 training data [Lebowitz 1985]. This algorithm, shown in Figure 4, groups data values for an attribute based on the distance between consecutive values. For a given attribute, Lebowitz's algorithm (LA) searches for the N-1 largest gaps between values. As with the BA method, the value of N is chosen arbitrarily. Using these values, LA then constructs N groups of values. As in the case of the bucket algorithm, each group of values is an interval. The relative gap size between two adjacent values is defined as the larger value minus the smaller value all divided by the smaller value. This approach is ill-behaved when the smaller value is near zero and may require rescaling of the data set values.

The third method of grouping attribute values is called conceptual clustering (CC). This method differs from the other methods in that: 1) it creates sets of values instead of intervals, and 2) it utilizes both attribute value and classification value. The conceptual clustering algorithm described here was developed by Dooley and St. Clair [1990] using a modification of the category utility measure found in Fisher's COBWEB learning system [Fisher 1987]. Fisher's clustering metric was originally developed as a means of predicting the basic level in human classification hierarchies. The metric groups attribute values based on intra-class similarity and inter-class dissimilarity of the objects being classified. Figure 5 provides the pseudocode for the conceptual clustering algorithm.

The Dooley and St. Clair conceptual clustering algorithm [1991] uses a revised metric, called the value clustering metric (VCM), to place attribute values into disjoint clusters. The VCM function is defined as,

$$VCM(A,I) = \sum_{j=1}^{n} P[V(A) \in I_j] * \sum_{k=1}^{a} \{P[V(C)=c_k \mid (V(A) \in I_j)]^2 - P[V(C)=c_k]^2\}$$

where A denotes an attribute, $I = \{I_j\}$ denotes the set of clusters I_j, and $P[V(A) \in I_j]$ denotes the probability that the value of attribute A is in cluster I_j. This probability is equal to the

```
define_groups(N_k)/*Lebowitz's Algorithm*/
  NUMBER_OF_INTERVALS = 6     /*N=6 was chosen here.*/
  foreach A_j ∈ N_k.A
     V = { }
     foreach t_i ∈ N_k.T
        V = V ∪ {a_ij}                    /* V = Set of A_j values */
     end
     lo = MOST_NEGATIVE_NUMBER  /*Find endpoints of*/
     hi = MOST_POSITIVE_NUMBER  /*range of A_j values*/
     if ( |V| = 1 )
        N_k.V_j = { (lo hi) }              /*Case for |A_j| = 1 value*/
     else
        V = SORT(V)                        /*Case for |A_j| > 1 value*/
        i = 0
        N = |V| - 1
        foreach v ∈ V
           if ( i > 0 )
              gap(i) = (v - last_v)/last_v
              mid(i) = v - gap(i)/2
           end
           last_v = v
           i = i+1
        end
        sorted-gaps = SORT(gap(1..N))
        if ( |V| > NUMBER_OF_INTERVALS )
           n = |V| - NUMBER_OF_INTERVALS + 1
           sorted-gaps = sorted-gaps(n..N)
        end
        mids = { }
        foreach gap(i) ∈ sorted-gaps
           mids = mids ∪ {mid(i)}          /*Set of midpoints*/
        end
        sorted-mids = SORT(mids)
        N_k.V_j = { }
        foreach mid ∈ sorted-mids
           N_k.V_j = N_k.V_j ∪ { (lo mid) }
           lo = mid
        end
        N_k.V_j = N_k.V_j ∪ { (lo hi) }
     end
  end
return
```

Figure 4. Lebowitz's Method

number of instances in cluster I_j divided by the total number of training instances available at the current node. The value $P[V(C) = c_k \mid V(A) \in I_j]$ is the conditional probability that the classification is c_k given that the value of attribute A is in cluster I_j. This is equal to the number of instances in cluster j with classification c_k divided by the number of instances in cluster I_j. The value $P[V(C) = c_k]$ is the probability that c_k is the class. This value is equal to the number of training instances at the current node having classification c_k divided by the total number of training instances at the current node.

```
define_groups(N_k)/*Conceptual Clustering Algorithm*/
  foreach A_j ∈ N_k.A
     V = { }
     h-vcm = 1
     clstr(1) = {a_1j}
                  /*Value of A_j in t_1 ∈ N_k.T forms first cluster*/
     c_set = {clstr(1)}/*Put first cluster in set of clusters*/
     foreach t_i ∈ N_k.T where i ≠ 1
        foreach clstr(m) ∈ c_set - {clstr(1)}
          if (VCM (A_j, (c_set - {clstr(m)}) ∪{clstr(m) ∪ {a_ij}})
                > VCM(A_j, (c_set - {clstr(h-vcm)}) ∪
                {clstr(h-vcm) ∪ {a_ij}}))
             h-vcm = m
          end
        end
        if (VCM (A_j, (c_set ∪ {{a_ij}})) >
                VCM (A_j, (c_set - {clstr(h-vcm)}) ∪
                {clstr(h-vcm) ∪ {a_ij}}))
           c_set = c_set ∪ {{a_ij}}
        else
           clstr(h-vcm) = clstr(h-vcm) ∪ {a_ij}
                        /*Add new cluster to set of clusters*/
        end
        N_k.V_j = c_set
     end
  end
```

Figure 5. Conceptual Clustering Method

Note that clusters represent a set of values which do not necessarily form a continuous interval. One interesting aspect of producing clusters, rather than intervals, is that when the tree is used to classify a new example, the attribute value may not exactly match any of the values in any of the clusters. When this occurs, a nearest-neighbor approach is used to select the cluster with which to identify the test instance. The values in each of the clusters are examined, and the cluster containing the value with the smallest absolute difference is selected.

Numerical Experiments

To evaluate the performance of the DID3 algorithm, experiments were conducted which compared the new algorithm's results against that of ID3 and the nearest neighbor version of ID3 (ID3nn). This latter algorithm, developed by St. Clair et al [1992], constructs ID3 trees in the usual way. However, when using a decision tree to classify data and a test case attribute does not exactly match the value of the corresponding attribute in the tree, the branch having the closest value is selected. St. Clair et al have shown that this method can provide significant improvement in classification accuracy.

Datasets were chosen for the experiments which would provide results from a wide range of domains. These datasets represent real domains and include noise. Each dataset had one or more continuous-valued attributes. Table I provides a summary of

these datasets. The Average Train/Class value in Table I indicates the average number of training examples for each classification in each set of training data. Assuming classification instances are uniformly distributed throughout the training data, Average Train/Class values provide a rough measure of the amount of training data available for each classification. This value does not necessarily predict algorithm performance since some concepts are easier to learn than others.

Nondestructive Evaluation (NDE) is performed on metallic and composite structures such as those found in aircraft in order to identify defects that are not visible on the surface. Each NDE instance contains one hundred attribute values and a classification value. There are ten possible classification values: nine different defects and no defect. Attributes represent time with attribute values corresponding to the strength of the ultrasonic waveform reflected from the material being tested. This data was obtained from NDE test engineers at Douglas Aircraft Company [Amirfathi et al 1991, Bond et al 1992].

Dataset	# Attributes	# Class.	# Instances	Average Train/Class
Breiman	21	3	399	100
CPU	6	6	209	26
Glass	9	6	214	23
NDE	100	10	180	14
Imports	24	7	205	22
Iris	4	3	150	38
Thyroid	20	23	80	3

Table I. Dataset Summary

Glass left at the scene of a crime can often be used as evidence if it can be correctly classified. Each instance in this dataset, from B. German, Central Research Establishment, Berkshire, is composed of nine numeric attributes: the refractive index and eight numeric attributes measuring weight expressed as a percentage of an oxide in the glass. The objective is to classify the type of glass. There are six glass types, such as vehicle glass, building glass, etc. The dataset exhibits very low correlation between individual attributes and classification type [Sabharwal et al 1992].

The CPU dataset contains 209 instances indicating the relative CPU performance of computers [Ein-Dor and Feldmesser 1987]. Instances are described by six numeric (integer) attributes and an associated relative CPU performance value. Relative CPU performance is represented by 116 integer values. Since the number of classifications is extremely large in comparison to the number of instances (209) in the dataset, the original classifications were grouped into 6 interval superclasses. The attribute correlation between individual attributes and CPU performance are very high.

The Iris flower dataset was developed by R. A. Fisher [1936]. It contains measurements of petal length, petal width, sepal length, and sepal width, from 50 Iris flowers each of species Virginica, Versicolor, and Setosa. Fisher used it to develop statistical methods of classification. Certain attributes, i.e. petal width, are more discriminating than others. This dataset was selected due to its wide use in the literature.

The Imports dataset, compiled by J.C. Schlimmer of Washington State University, consists of 205 samples of imported automobiles and seven insurance risk classification symbols. Symbol values are integers, ranging from a value of +3 [very risky] to a value of -3 [very safe]. The risk classification symbol is associated with eleven discrete and thirteen ordinal attributes. This dataset was selected for the large numbers of attributes and unique attribute values it contains.

The Thyroid dataset, from the Garvan Institute of Medical Research in Sydney, contains diagnoses of thyroid conditions based upon seventeen discrete attributes and three continuous valued attributes. This dataset was included because it contains examples where not all attribute values are known. For example, certain tests may not have been performed on the patient. These unknown attribute values are associated with an unique symbol to permit the decision tree to be constructed.

The Breiman dataset is based on a waveform recognition problem described by Breiman et al [1984]. Each waveform is generated by a function which randomly combines two of three given, triangular waveforms and adds random N(0,1) noise. The classification of the resulting waveform is determined by the two input waveforms chosen for each of the training examples constructed. Thus, three different classifications of waveforms are produced. For each classification, 133 waveform samples were generated for a total of 399 samples. Each sample is described by 21 attributes which represent the magnitude of the waveform at a point in time.

Experiments were conducted with each dataset using the V-fold cross-validation technique described in Breiman et al [1984]. In the V-fold cross-validation implementation used, each dataset was randomly partitioned into four subsets of roughly equal size. In each experiment, one of the subsets was used for testing while the other three were used for training. This was repeated four times with a different subset being used as a test set each time. Breiman et al indicate this approach works well for small datasets.

Table II shows the average % Correct for each set of V-fold tests for each algorithm and for each dataset. The first two rows in the table were obtained from ID3 and ID3nn. Results listed under Preprocessing were achieved by preprocessing the continuous attributes in each dataset and then applying ID3 to the resulting data. Bucket6 data was created by preprocessing the data using the bucket method with N = 6. Lebo6 and Lebo10 denote Lebowitz's method using 6 and 10 intervals respectively. Cluster denotes results from conceptual clustering. Dynamic Processing denotes the results obtained from the DID3 algorithm using the associated grouping procedures.

The average for each method across all datasets, Ave., indicates that dynamic processing, in general, produces higher percentages of correct results. Cluster is the notable exception to this set of results. The next best set of results is obtained

	Brem	Comp	Glass	Impt	Iris	NDE	Thyrd	Ave.	sd
ID3	33.3	58.4	15.0	5.8	76.0	5.0	60.0	*36.2*	*28.9*
ID3nn	67.9	72.7	45.8	31.7	92.7	42.2	76.3	*61.3*	*21.8*
Preprocessing									
Bucket6	85.0	41.2	47.2	75.2	88.0	63.4	66.3	*66.6*	*17.8*
Lebo6	91.2	19.2	23.4	66.9	43.9	86.1	68.8	*57.1*	*28.8*
Lebo10	91.0	60.8	21.5	74.8	55.9	85.6	70.0	*65.6*	*23.1*
Cluster	82.4	78.4	36.4	30.3	90.6	57.8	5.0	*54.4*	*31.7*
Dynamic Processing									
Bucket6	88.7	71.3	63.6	77.2	94.7	61.7	71.3	*75.5*	*12.3*
Lebo6	94.2	64.2	52.9	76.7	72.7	84.5	75.0	*74.3*	*13.4*
Lebo10	92.2	68.4	60.3	76.7	77.9	75.6	75.0	*75.1*	*9.7*
Cluster	67.9	78.5	34.6	31.2	92.0	42.2	56.3	*57.5*	*23.1*
Ave.	79.4	61.3	40.0	54.6	78.4	60.4	62.4	*62.4*	*13.6*
sd	17.8	17.5	15.8	25.5	16.2	24.2	20.1	19.6	

Table II. Average % Correct (%)

from Bucket6 and Lebo10 preprocessing. The ID3 algorithm with nearest neighbor evaluation, ID3nn does almost as well as the best preprocessing approaches. The high values for standard deviation, sd, indicate a wide variability in the outputs produced across all datasets. The sd values for dynamic processing are generally smaller than for the other techniques.

The two rows at the bottom of the table show the average % Correct and standard deviation on each dataset by all of the algorithms. The dynamic processing versions of Bucket6 is the only technique which produces results consistently above the average produced by all algorithms for each dataset. Dynamic Lebo10 is a close second.

The results of clustering are much poorer than originally anticipated. It was expected that the use of classification to group data values would increase the average % Correct. Table II indicates that this was, in general, not the case.

It was anticipated that % Correct values produced by ID3 might indicate the type of performance to be expected from the other algorithms. Examination of the correlation coefficients (cc) between ID3 % Correct and the % Correct for each of the other algorithms showed that the highest correlations occurred between ID3 and the dynamic versions of Lebo10 (cc = 0.81) and Lebo6 (cc = 0.65). The correlation between ID3 and ID3nn was only 0.52. The correlation between ID3 %Correct and Dynamic Bucket6 % Correct was only 0.12 even though the Dynamic Bucket6 algorithm had the highest average accuracy of all algorithms over all datasets!

Table III provides another type of evaluation of the trees formed by each of the methods. It is % Generalization where;

$$\%\text{Generalization} = \left(1 - \frac{\#\text{ leaf nodes}}{\#\text{ training examples}}\right) \times 100.$$

If each training instance is thought of as a rule, % Generalization indicates the % reduction in the number of training rules that is produced by the decision tree.

As would be expected, a high % Generalization does not indicate a high % Correct. In fact, the dynamic versions of Bucket6 and Lebo10 produce the highest % Correct averages but produce relatively low % Generalization values. The low values for ID3 and ID3nn indicate that ID3 only reduced the original datasets by about 27%. The average reduction over all datasets and all methods is 59.2%.

The correlation coefficient between % Correct and % Generalization for all methods and all datasets was 0.27 indicating no significant correlation between these values. For ID3 alone, the correlation between % Correct and % Generalization was an impressive 0.98! For ID3 and dynamic Lebo10, the % Correct and % Generalization correlation was 0.78. All other correlations were significantly smaller.

Conclusions

Experimental results suggest that the Bucket6 and Lebo10 versions of the new Dynamic ID3 algorithm (DID3) produce decision trees with higher classification accuracies over a range of complex datasets. At the same time, the trees produced by these algorithms are less complex than those produced by ID3 but somewhat more complex then the trees produced by preprocessing approaches. Although more difficult to implement, the new algorithm is effective in domains whose set of attributes consist of both discrete and continuous attributes.

Acknowledgement

The authors would like to thank University of California - Irvine for use of its Repository of Machine Learning Data.

	Breim	Comp	Glass	Impt	Iris	NDE	Thyrd	Ave.	sd
ID3/ID3nn	32.3	46.1	7.2	3.4	55.4	3.7	41.3	*27.0*	*22.0*
Preprocessing									
Bucket6	69.5	70.3	45.6	54.8	78.2	56.3	66.3	*63.0*	*11.2*
Lebo6	89.6	72.4	75.2	51.7	88.0	76.7	69.6	*74.7*	*12.7*
Lebo10	85.5	52.6	65.1	51.9	82.7	68.9	61.3	*66.9*	*13.3*
Cluster	76.2	55.4	70.2	90.7	85.8	91.9	95.8	*80.9*	*14.4*
Dynamic Processing									
Bucket6	67.6	42.4	37.2	51.6	77.1	49.1	62.1	*55.3*	*14.2*
Lebo6	84.6	41.0	39.1	50.7	73.6	61.5	61.3	*58.8*	*16.7*
Lebo10	73.2	40.7	29.4	47.0	60.0	44.8	52.9	*49.7*	*14.1*
Cluster	99.0	76.9	79.0	90.9	91.8	91.9	89.6	*88.4*	*7.8*
Average	71.0	54.4	45.5	49.6	75.0	54.8	64.1	59.2	11.1
sd	21.4	13.2	25.1	27.8	12.4	29.7	17.0	20.9	

Table III. % Generalization

References

Amirfathi, M., Morris, S., O'Rorke, P.O., St. Clair, D.C., and Bond, W.E., *Pattern Recognition for Nondestructive Evaluation*, **IEEE Aerospace Applications Conference**, Feb. 3-8, 1991.

Bond, W.E., St. Clair, D.C., Amirfathi, M.M., Merz, C.J., and Aylward, S., *Neural Network Analysis of Nondestructive Evaluation Patterns*, **Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing**, Vol. II, ACM Press, March 1992, pp. 643-650.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J., **Classification and Regression Trees**, Wadsworth & Brooks/Cole, 1984.

Dooley, J.F. and St. Clair, D.C., Personal discussions on using category utility to cluster continuous-valued attributes, McDonnell Douglas Research Laboratories, 1990.

Ein-Dor, E., and Feldmesser, F., *Linear Regression Predictions of Relative CPU Performance*, **Communications of the ACM**, Vol 4, 1987, pp. 308-317.

Fisher, D.H., *Knowledge Acquisition Via Incremental Conceptual Clustering*, **Machine Learning**, Vol. 2, 1987, pp. 139-172.

Fisher, R.A., *The Use of Multiple Measurements in Taxonomic Problems*, **Annals of Eugenics**, Vol. 7, 1936, pp. 57-90.

Hacke, K.R., St. Clair, D. C., and Sabharwal, C. L., *Incremental Learning of Numeric Clusters in Classifier Systems*, **Advances in Logic Programming and Automated Reasoning**, ed. R.W. Wilkerson, Ablex Publishing Co., 1992. (in press)

Lebowitz, M., *Categorizing Numeric Information for Generalization*, **Cognitive Science**, Vol. 9, 1985, pp. 285-308.

Quinlan, J.R., *Induction of Decision Trees*, **Machine Learning**, Vol. 1, 1986, pp. 81-106.

Sabharwal, C. L., Hacke, K.R., and St. Clair, D. C., *Formation of Clusters and Resolution of Ordinal Attributes in ID3 Decision Trees*, **Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing**, Vol. I, ACM Press, March 1992, pp. 590-597.

Shannon, C.E., *A Mathematical Theory of Communications*, **Bell Systems Journal**, Vol. 77, 1948, pp. 379-423.

St. Clair, D.C., Bond, W.E., Rigler, A.K., and Aylward, S., *An Evaluation of Learning Performance in Backpropagation Neural Networks and Decision-Tree Classifier Systems*, **Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing**, Vol. II, ACM Press, March 1992, pp. 636-642.

St. Clair, D. C., Sabharwal, C. L., Hacke, K., and Bond, W. E., *Using Decision-Tree Classifier Systems to Extract Knowledge from Databases*, **Proceedings of the Fifth Conference on Artificial Intelligence for Space Applications**, May 1990.