In *FLAIRS'94 Florida Artificial Intelligence Research Symposium,* pp. 122-125, 1994.

# Extending ID3 Through Discretization of Continuous Inputs

Rick Bertelsen
Tony R. Martinez

Computer Science Department, Brigham Young University, Provo, Utah 84602
e-mail: rick@axon.cs.byu.edu, martinez@cs.byu.edu

**Abstract**

This paper presents a mechanism to extend ID3 by classifying real valued inputs. Real valued inputs are classified through a neural network model termed the *Competitive Classifier* (*CC*). The CC forwards discrete classification results to the ID3 system, and accepts feedback from the ID3 system. Through the use of feedback, the ID3 system guides the CC into improving classifications.

## 1 INTRODUCTION

Many problems solved by humans require the analysis of continuous or real valued inputs. In these problems, it appears that the actual real value is used only to produce a classification. The resultant classification may then be used as an input to solve the larger problem. For example, consider the problem of selecting clothing for a given day. Possible inputs to solve this problem are outlook, forecasted wind speed, and temperature. A person typically classifies the temperature from a small set of possible classifications including "hot", "cold", and "warm". This classification of temperature is often used as an input to the problem, as opposed to the actual real value of temperature. A person might classify the real valued temperature (i.e. 99.48˚F) into the class "hot", and thus select a cool shirt to wear. These classifications are based on experience. Figure 1 below shows some possible temperature values with a potential break down of classes (shown simply as class boundary lines).

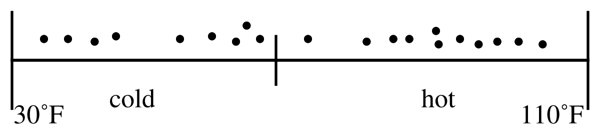


Figure 1 - Potential temperature classifications

One could spontaneously discover potential classifications using any number of clustering algorithms. However, there is no guarantee that the resulting natural classifications will assist to solve the problem. Clustering algorithms typically cluster according to statistical properties of the values of an attribute without regard to the larger problem. Humans use experience with the *actual decision (the output)* of the larger problem to form useful classifications. In this sense, experience *guides* a person in the formation of useful classifications. This paper proposes a model which feeds classifications forward to a supervised learning system, and accepts feedback from the supervised learning system to improve classifications. Figure 2 shows a potential improved classifications derived from the initial classifications given in figure 1. Learning may adjust some class boundaries, and may even create additional classes based on the problem at hand.

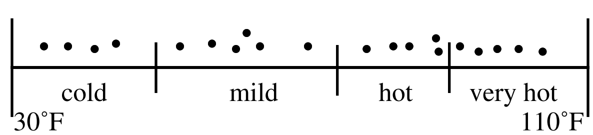


Figure 2 - Improved classifications for temperature

This paper proposes a mechanism to guide real valued inputs into useful classifications for an ID3 [Quinlan 86] learning system. This paper begins by briefly presenting the classification system used. An extension of ID3 is then presented and briefly discussed. The paper concludes discussing simulation results and future work.

## 2 THE COMPETITIVE CLASSIFIER

Guided classification is accomplished through the *Competitive Classifier (CC).* The CC accepts continuous valued inputs and produces discrete classifications as output. The classification is performed by a single layer neural network model which implements local, dynamic inhibition. The output nodes of the network implement a minimum distance activation function, and then compete through inhibition to select the final output. Through the use of local and dynamic inhibition, nodes have a variable sized region of influence.

Figure 3 shows how the CC is connected to an ID3 system. The CC forwards the classifications via a feed forward communication path. The CC is guided in its classifications via a feedback communications path. The output of the CC is consumed as part of the input to the ID3 system.

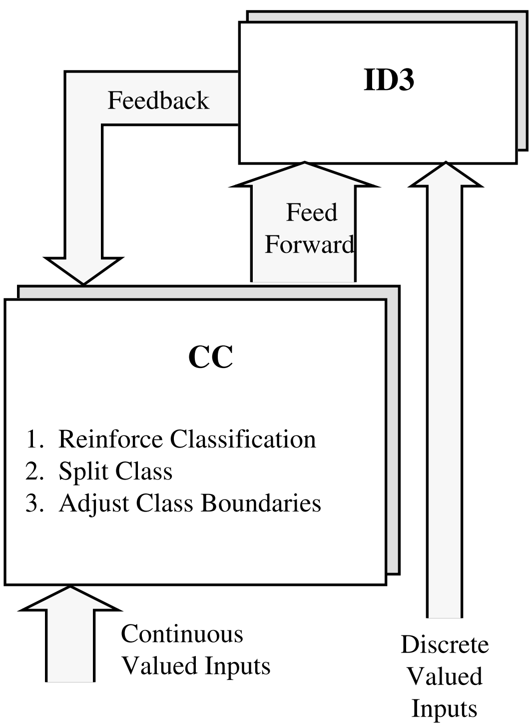


Figure 3 - The overall system

The CC is guided through feedback generated by the ID3 system. Three operations are currently supported by the CC as feedback from the ID3 system. These operations are:

- Reinforce classification.
- Split one class into two.
- Adjust classification boundaries.

*Reinforce* classifications causes the CC to move the *center* for the class to reflect a new input value correctly classified into a class. The center of a class can be thought of as a *prototype* for the class. *Split* one class into two causes the CC to allocate a new node for a particular output class. The new node is initialized by setting its prototype to the value which caused the split. Inhibition between the new class and the old class is initialized in such a way as to help preserve many of the classifications from the old class. It is important to note that some previously classified in the old class will now be misclassifed in the new class. *Adjust* classification boundaries causes the inhibition between two nodes to change. This may cause a value which had been included in a particular class to be classified in a different class. Through careful adjustment of the classification boundaries, the CC may fine-tune the classifications. By adjusting the inhibition between a new class and an older class, values misclassified by the creation of the new class can be properly classified.

## 3 EXTENSION OF ID3

The ID3 control strategy must be modified to guide the CC into useful classifications. A modified ID3 control strategy is presented below, with modifications denoted with a *. Discussion of each step is found in italics after the step.

1. Randomly select a subset of the training set. This subset is the *training window.* The training window can include the entire training set if desired.

   *This step is the same as in the original ID3 algorithm.*

2.* Coarse-code the continuous variables into natural classes.

   *This step involves an initial coarse coding of the continuous values into a number of discrete classes. Any natural clustering algorithm will suffice. Only classify the continuous variables of instances in the training window.*

3. Apply the ID3 selection criterion to each attribute and determine the root of the tree, then recursively build a decision tree from the training window.

   *This step of the algorithm remains unchanged from the original ID3 algorithm.*

4.* Iteratively apply the tree to classify instances from the training set that are not in the training window. If the training window consists of the entire training set, apply each instance of the training set to the ID3 tree.

*This step of the algorithm incorporates the feedback to the classifier. As each instance of the training set is applied to the tree created in step 3, the system senses the output. Based upon this output, the following options exist.*

**Cases:**

a) Tree gives correct output - *reinforce* classifications.

*In this case, the ID3 tree gives the correct output for the training instance and a message to reinforce the classifications giving the correct output is fed back to the CC. As previously mentioned, reinforcing of the class causes the prototype to move towards the new center of the class. Adjustments are also made to the inhibition such that class boundaries remain the same.*

b) Tree gives incorrect output -

1) If a discrete attribute in the current instance exists which is not part of the ID3 tree, add the instance to the training window.

*The current instance may be an important exception denoted by a discrete variable. Thus, in order to properly classify this instance, it is added to the training window. If the discrete variable is not useful for classifying the problem, the ID3 selection criterion will assure us that the attribute is not needed.*

2) Otherwise, for a selected classified variable attempt a second value. Allow the ID3 tree to form an output using the reclassified value.

a) If the second classification produces a correct output, *adjust* classes accordingly.

*The classified variable is selected by metrics provided by the CC. The instance tested is the same as the instance previously attempted except the one classified variable now has a different value. For example, for the classified variable temperature the value "warm" may be tried in place of the value "hot". In order to adjust class boundaries, the CC forwards the two best classifications along with a confidence metric for each classification. If the confidence of the second classification is sufficiently close to that of the first, a second classification is attempted. Confidences will be sufficiently close only when the real value lies close to the boundary between the two classes. The boundary between the proper classification and the class incorrectly selected is adjusted by changing the inhibition between the two classes.*

b) Otherwise, *split* the first class attempted.

*If the second classification results in an incorrect output, or the confidences mentioned above are not sufficiently close, split the first class attempted. Place the current real value at the center of the new class. Set the inhibition of the new class to neighboring classes very low, resulting in a narrow new class.*

5.* After all instances of the training set have been tested-

**Cases:**

a) If some instance has been added to the training window, then **throw out the current tree, keep the current classifications**, go to step 3.

*This step is same as the original ID3, except the current classifications are not discarded. The current classifications are easily held over since the CC is separate from the ID3 system. The guided classifications have been modified to solve the problem, thus these classifications may still be useful.*

b) If the training window is the same, iterate through the training set until the classifications do not significantly improve.

## 4 SIMULATION RESULTS

Simulation of the above algorithm show both the potential of the technique, along with the need for future work. The simulation uses the algorithm stated above, and allowed the training window to be either the entire instance set, or a subset of the instance set. The coarse coding is one of two possible techniques: The first merely initializes the prototypes in the CC to random values in the proper range of the particular attribute The second technique initializes the prototypes in the CC to equidistant points in the range of the particular attribute. Due to the difficulty in reproducing results with random initial points, the latter method is typically used.

Using the University of California at Irvine hepatitis dataset some preliminary results are obtained. With two equidistant initial coarse coded classes, the algorithm is able to construct a tree and properly classify 58% of the instances. In ten iterations the algorithm constructs a tree to properly classify 72% of the instances. Further iterations give slight improvements over this result. The size of the tree increases somewhat over the course of the ten iterations, but not significantly. Typically tree growth is associated with node creations. When only two initial classes are used, the CC usually creates one additional class for each possible attribute, but rarely are two classes created for any attribute.

When the simulation uses three or more equidistant initial classes, the results are comparable to those obtained with two initial classes. However there is an increased tendency to create new classes, thus causing the tree to grow.

In some of the simulations, random initial prototypes are used. As expected, these results varied greatly from trial to trial. While random starting points are difficult to reproduce, they are useful. In simulations where only two initial classes are formed, and the initial prototypes are near the minimum and maximum value for the attribute, results were much better. In some cases the classifying ability varies from near 50% initially to over 80% after ten iterations.

These results are better in some cases than results obtained using equidistant initial prototypes. The explanation appears to be in the number of new classes created for an attribute. When the random initial prototypes are near the extremes for the class, fewer new classes are created. The only learning in the CC is reinforcement of classes, and adjustment of inhibition creating an adjustment in the boundary between the class. Thus the data is separated in one of the two classes by a threshold defined by the inhibition between the two classes.

Creating a new class does not always create a useful class for the task of discriminating in the tree. Several methods are used to determine which attribute should be selected in which to create a new class. The best in terms of adding discriminatory power to the tree is to select the continuous valued attribute which corresponds to the last node visited in the tree. If the last node is a discrete valued attribute, simply continue up the tree until the node corresponds to a continuous valued attribute. This method however, may create a class which adds little discriminatory power to the tree. Research is on going in order to methods to detect when a class is not useful and merging that class with another class. Another method used to select the class in which the new node is created is to select the continuous attribute which has the lowest value for the selection criterion being used. One final method selects the attribute which has the lowest confidence forwarded from the CC. Confidences are based on a combination of the closeness of the value being classified to the nearest prototype, the number of classes for the attribute, and the value range of the attribute. Values near a prototype are more confident than values near the boundary

## 5 CONCLUSIONS AND FUTURE RESEARCH

This abstract has presented a method of extending ID3 to accept continuous valued inputs. Current research includes:

- More empirical testing.
- Better methods to determine which attributes and classes within attributes should be split.
- Augmenting the CC with the ability to sense when to merge classes and efficient methods to merge classes.
- Extend the methodology to other machine learning models which do not support continuous valued inputs.

## Bibliography

**Quinlan J.R. (1986)** Induction of Decision Trees. *Machine Learning*, 1:81-106, Kluwer Academic Publishers, Boston.

This research was supported in part by grants from Novell, Inc. and WordPerfect Corporation.