Experiments with Decision Tree Classifiers – Discretization of Numerical Attributes

Michael Barbieri, Kenneth Dayton, Wyatt Hebert, and Kimberly Robinson

College of Engineering and Computer Science, University of Central Florida, Orlando, FL, 32826, USA

Abstract — Classification algorithms are used in numerous applications everyday, from assigning letter grades to student student’s scores, to computerized letter recognition in mail processing. Discretization consists of applying a set of rules to reduce the number of discrete intervals from which an attribute is assigned. Discretization is generally applied to datasets whose numerical range consists of continuous values, replacing the numerical domain with a nominal domain. This paper’s primary focus is on the application of discretization algorithms on Classification and Regression Trees (CART) using Equal Width, Equal Frequency and CAIM discretization algorithms. While no method dominated others for all datasets, results show that discretization prior to classification generally decreases the processing time, while increasing the accuracy of the classifier. The data presented here further substantiates this claim.

Index Terms — Supervised discretization, CART, classification, class-attribute interdependency, continuous attributes, machine learning algorithm.

I. INTRODUCTION TO DECISION TREES

Classification algorithms are used in numerous applications everyday, from the human brain deciding the color it is seeing to automated machinery sorting apples and oranges. Classification algorithms take an input dataset and classify it according to a predetermined rule or set of rules. Machine learning algorithms are programs that a computer uses to determine the set of rules by which to classify a given dataset. This project’s primary focus is on Decision Trees, which are a particular type of classifier, and the Classification and Regression Tree (CART) algorithm, which is a particular machine learning algorithm that grows decision trees.

Datasets are comprised of a group of data points, which have values for a number of attributes. The number of attributes in a dataset is typically referred to as the dimensionality of the set, which will be discussed in more detail below. Datasets can be typically seen as an NxM matrix, with N columns, where N is the number of attributes in the set, and M rows, where M is the number of data points in the set. Attributes of datasets can be numerical or categorical. Numerical attributes are those that take numbers as descriptions. Height and weight are examples of numerical attributes. Categorical attributes are those that have words to describe them, such as color or car model. These datasets are of paramount importance when it comes to determining the usefulness of the algorithms. [4].

Machine learning algorithms come in two basic types, supervised and unsupervised. The unsupervised type takes the dataset as described above and determines how best to cluster the inputs into likely groups of common data. Supervised algorithms, on the other hand, know the classification of the input data, and learn how to best separate the dataset into the known classes. Decision Trees are supervised, and the rest of this paper will focus on supervised algorithms. [3]

A. Classifiers

A Decision Tree as a classifier can best be described as a series of if/then statements that are asked in a particular order to separate the dataset into subsets that are of a particular class. The entire training set is taken together at the top of the tree, in what is referred to as the root node. The algorithm discovers the best way to split the root, and produces two children or daughter nodes. [2]

Fig. 1. Graphical explanation of decision tree classification process. Drawing courtesy of Mark Librizzi.

When a tree node has no child nodes, it is referred to as a leaf node (See Fig. 1). If it is not, it is an internal or decision node, so named because it must be queried to discover what if/then rule must be applied to it to further classify the data in it. [3]
Decision Trees have two primary benefits over other classification systems, both stemming from the classification algorithm. The first benefit is that a Decision Tree can use either categorical or numeric data, as is and without converting categorical to numeric. The second benefit is that the actual time to classify a given data point is fairly fast. For an n-tier tree (the picture is a 2-tier tree), the maximum number of if/then statements the code must go through to classify is at most $n-1$. This results in fairly quick classifications. To contrast this, nearest neighbor and ARTMaps, two other classifying algorithm types, have to check distance calculations on all points in the training set to determine a classification. This can lead to greatly increased classification time. [3]

However, the drawbacks of Decision Trees are considerable in their own right. To create a classifier using an MLA, it is preferable to have a very large training set. However, as the training set gets extremely large, the Decision Tree growth time takes longer and longer. Our project's stated and primary purpose is to investigate what effect discretizing the training set will have on the overall computation time, accuracy, and complexity of the resulting tree. Another drawback, which compounds the first, is that if the Decision Tree needs to be recreated, it must be regrown from scratch, with, typically, a larger training set. This lends extreme importance to the investigation and discovery of discretization algorithms that will reduce the size of the training set, without losing much accuracy nor increasing complexity. [3]

This project is being funded and supervised by the National Science Foundation, with a requirement to create a Machine Learning software repository, including implementation of various MLA's, creation or listing of datasets of interest, and helpful header files as coding aids. The code is to be submitted as suitable for MATLAB use, and as such needs to be encoded as Microsoft™ dll's due to the unfortunate overhead MATLAB imposes on normal m.file (function) implementation. This has merit for many reasons. Primarily, it creates a drag-and-drop set of dll's that anyone with MATLAB can use to run MLA research. Given that MATLAB has become a central fixture in modeling and calculation software, this creates a widely available storehouse of knowledge. Secondly, it supports the growth of the MATLAB software suite.

Classifiers are extremely powerful tools that have great utility both in research settings and in industry. The purpose of a classifier is self-explanatory: to take an object or concept and place it into a category. A human who is familiar with the objects being classified can generally do this very easily and intuitively. For instance, an experienced individual can, without explicit thinking, decide whether a certain fish is a sea bass or a salmon with near-perfect accuracy. How does a person do this? Most likely, the underlying thought process present involves an analysis of certain attributes of a given fish, such as color, length, or shape. Recognizing patterns such as these allow the human brain to make extremely accurate classification judgments.

While the human brain is remarkable in its pattern-recognition abilities, there are two key benefits to designing a machine to perform a similar function: cost and speed.

A machine requires a significant up-front cost to build, but afterward requires only the costs of operation, which are generally less than human labor. If the machine is a computer program, these costs become almost trivial.

The second benefit is speed. A human classifier has physical limitations that dictate how quickly he/she can classify large amounts of incoming data. A machine can be designed to handle bulk classification and perform this task quite rapidly.

With the advantages of building a machine to do classification tasks now apparent, the question invariably arises of how such a machine will work. A computerized machine classifier is essentially divided into five different parts, representing steps in the classification process: sensing, segmentation and grouping, feature extraction, classification, and post processing.

The sensing part is an actual physical device that extracts data from the real world and stores it electronically. In the example of fish being classified as sea bass or salmon based on their length and color, this would likely be a camera array of some sort. The segmentation and grouping step involves dividing the input into discrete parts. This is necessary because the data may arrive grouped together or even overlapping, and our classifier must recognize boundaries and classify each item as an individual data point.

The third step is the feature extraction step. Continuing with our fish example, the inputs of this step are the discrete, separated images of individual fish, and the output is a database with an entry to represent each fish and its attributes. Here is where calculations for the length and the color of the fish are made.

The fourth step is the classification step. This is where the data taken from the feature extraction step is used to make a determination of which category each data point belongs to. This is the part that our project deals with. Specifically, our project is about determining ways to speed up the process of building this fourth part of the classification system: the classifier itself. The fifth and
The final step is the post processing step. The post processing step is used to dictate an action to be taken by an overseeing machine or individual based on the classification of a given object.

The most important issue facing classifiers today is accuracy. Machine classifiers are very prone to error, and depending on the tolerance a particular application has toward error, one may be forced to invest in an extremely sophisticated classifier in order to minimize this problem.

Another major problem, specifically with classifiers constructed by machine learning algorithms such as CART, is the speed at which a classifier can be built. For large amounts of training data, CART takes an extremely long time to build a tree. The purpose of our project is to evaluate various methods of discretizing the training data so that the time can be reduced a great deal without significantly affecting the accuracy of the resulting classifier.

**B. Datasets**

The databases used by Machine Learning Algorithms to build decision trees are separated into four general columnar areas: *continuous attributes, discrete attributes, discrete targets*, and *continuous targets*. Continuous attributes are equivalent to the numerical attributes mentioned in the overview, and discrete attributes are synonymous with the categorical attributes. Discrete targets are likewise synonymous with classes, and continuous targets are used in functional regression. Discrete attributes, particularly, are represented by ordinals instead of the words, due to the high computational cost of string comparisons as opposed to pure numerical comparisons. Our project is limited in scope to continuous attributes and discrete targets.

The following is a sample database that we will be using throughout the paper, and it will be reprinted as necessary. However, it is included here so that the explanations can be printed with the database information system.

<table>
<thead>
<tr>
<th>Data Points</th>
<th>Continuous Attributes</th>
<th>Discrete Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>Class</td>
</tr>
<tr>
<td>1</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.50</td>
</tr>
<tr>
<td>3</td>
<td>0.35</td>
<td>0.45</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>0.60</td>
</tr>
<tr>
<td>5</td>
<td>0.40</td>
<td>0.60</td>
</tr>
<tr>
<td>6</td>
<td>0.55</td>
<td>0.45</td>
</tr>
<tr>
<td>7</td>
<td>0.40</td>
<td>0.50</td>
</tr>
</tbody>
</table>

![Fig. 2. Example of a dataset with 10 datum, 2 attributes and 2 classes.](image)

X and Y are the names of the Continuous Attributes, and Class is the name of the Discrete Target. In this example, X has seven distinct values, Y has five distinct values, and Class has two distinct values. The dimensionality of a dataset is defined as the number of attributes it has; this dataset has a dimensionality of two. The number of classes a dataset has is also very important, particularly for discretization, and this set has two. Lastly, this dataset has 10 data points, labeled by simple ordinal numbers. The number of data points is of prime importance to our project in particular, as the focus of the project is discretization. The primary purpose of discretizing the database that we are working with is to reduce the number of data points without losing a similar percentage of final accuracy, and without adding more calculations than we are removing.

Datasets can be described as well-behaved or poorly behaved. Well-behaved datasets are easily separable, and sometimes visual inspection can suggest proper locations to separate the data points into their appropriate classes. A classic well-behaved dataset that is commonly used in evaluating classifiers is the Iris dataset, for classifying types of Iris flowers. The Iris flower has three different varieties. Taking the first two attributes on a pair of axes, we get the following graph:

![Fig. 3. Example of Iris database which exhibits a well behaved pattern.](image)
As can be seen, the vertical separation line will put one entire set alone, and the horizontal separator will separate the blue class from the red class. Most MLA’s will not stop at such simple divisions made in their primary phases of execution, which is why there is a regressive step that checks how important a given level of distinction is to the overall accuracy of the resulting classifier. [2]

C. Goals of Project EMD-MLR

Our project is part of a much larger project involving students of two universities and two community colleges: the University of Central Florida (UCF) and the Florida Institute of Technology (FIT), as well as Brevard and Seminole Community Colleges (BCC and SCC). This project is entitled Project EMD-MLR, which stands for Educational Materials Development through the Integration of Machine Learning Research into Senior Design Projects, and is sponsored by the National Science Foundation (NSF) and the two universities involved. In addition, it has caught the attention of Productivity Apex, an Orlando-based research and consulting firm, which may use code written for Project EMD-MLR in their commercial Data Mining Tool.

The goals of Project EMD-MLR, as stated on the official website, are as follows:

1. Develop educational materials in Machine Learning.
2. Introduce Machine Learning research into the undergraduate curriculum of the participating institutions (FIT, UCF, BCC, & SCC).
3. Disseminate the educational material produced.

We have created three discretization algorithms as MatLab dll’s: Equal Width, Equal Frequency and CAIM.

As required, we will present our final findings to the NSF advisors for inclusion as part of their sponsored studies. Our implementation of CART and the discretization algorithms under study has been greatly aided by Mingyu Zhong, a Ph.D. candidate at UCF, who has written a header file for use with writing MLA’s inside of the MATLAB .dll environment.

II. CLASSIFICATION AND REGRESSION TREES

We have chosen the CART algorithm to test our discretization algorithms due to the literature in the field that experimented with many of these discretizing algorithms on the C4.5 algorithm. [6] C4.5’s growing phase is identical to CART’s, and since the discretizing effects are seen mostly in the growing stage, we have a benchmark for our results that may not have been present had we chosen another algorithm. Our research is also specifically limited to binary splits and numerical data. Binary splits are those that produce exactly two child nodes from each parent node.

A. Growing the Tree

The basic idea of the growing phase of CART is as follows: it begins at the root node, which contains the entire training data set. For this node, it must decide where to split the data in the node to create the two child nodes. It bases its decision where to split based on purity of the resulting nodes (data containing less variety of classes) in the child nodes. This ensures the most accurate and efficient tree possible. Once CART decides on a split, the two children are created, and then this process is repeated iteratively on the children. CART continues until
the stopping conditions, discussed later, are reached. [5]

CART chooses its split values by a mathematical process, whereby it checks every single possible split it can make, and discovers what the data points in each child node would be, and then discovers how closely each node is to being a single class. Typically, the term used to describe how wholly a node is of a single class is the purity of the node. The impurity, then, is how far the node is from being of a single class. It is possible to devise a function that can be applied to a node to measure its impurity. Such a function must satisfy the following requirements:

1. It must be at a minimum (typically zero) when the node is pure,
2. It must be at a maximum when the nodal class probability (defined as the probability that a given data point in the node is of a particular class) of each class is equal to 1/J, where J is the number of classes in the training set, and
3. It must be symmetric.

Two measures are typically used to measure impurity. They are Shannon’s Entropy function [9] (1) and the Gini function (2).

\[
i(t) = \sum_{j=1}^{J} - p(j \mid t) \log_2(p(j \mid t)) \quad (1)
\]

\[
i(t) = 1 - \sum_{j=1}^{J} p^2(j \mid t) \quad (2)
\]

The CART growing phase first calculates the impurity of the current node under consideration. Then, for each attribute, it verifies all the possible split values. Typically, these potential split values are the midpoints of all the distinct values of the examined attribute. In a set of values \{0,0.1,2,3,4,4.5,6\}, the possible split values are 0.5, 1.5, 2.5, 3.5, 4.5, and 5.5. For each possible split value, it calculates the impurity of each daughter node.
using the same impurity function as before. It then averages the impurity of the daughter nodes, and evaluates the change in impurity $\Delta(s, t)$. [5]

$$\Delta(s, t) = i(t) - p(t_L) \cdot i(t_L) - p(t_R) \cdot i(t_R)$$ (3)

The algorithm continues until it reaches the stopping criteria for the splitting process listed below.

1) If the node is all of one class, it is a leaf.
2) If a nodal impurity is less than or equal to a specified impurity, it is a leaf.
3) If all the attributes of the data points in a node are identical, but the classes are different, it will be labeled a leaf.

**B. Pruning**

After the tree is fully grown, the pruning phase takes over. Pruning is the process by which the algorithm checks to see how the tree could be optimized. Pruning is the process of removing leaf nodes from the tree to attempt to make the tree both smaller and more resistant to outlier effects, as discussed above. Two criteria are used to decide on optimal Decision Tree size: Number of Leaves and the Error created within the Tree by the Pruning process. One equation used to represent this phenomenon is the cost complexity equation:

$$CostComplexity = R(t) + \alpha(\frac{1}{T})$$ (4)

Here, $\alpha$ is simply a parameter. It typically represents the complexity cost of larger numbers of leaves. $R(t)$ is the resubstitution error, or classification error in the tree after the pruning is done. T-bar is the number of leaves remaining in the tree after the pruning action. This equation creates a series of lines with the Cost as the dependent variable and $\alpha$ as the independent variable (Fig 5). For our implementation, the program mathematically selects nodes for pruning, and removes all of their children nodes. The equation used in the algorithm is:

$$ErrorMinimization = \frac{R(t) - S(t)}{N(t)-1}$$ (5)

$R(t)$ is the error being added to the system by the pruning step, $S(t)$ is the current error in the leaves resulting from that node, and N is the number of leaves removed by the Pruning step. The numerator of the equation is the change in Error being made by pruning, and the denominator is the complexity reduction. CART checks all decision nodes, calculates this Error Minimization, and removes the node or nodes whose Error is minimized, as minimum error would correspond to the pruning of a node which will result in lowest new error over the largest leaf count reduction. The algorithm saves the tree then iterates again, with the new, pruned tree. This continues again until the tree is reduced all the way to just the root node. At this point, the algorithm takes all of the pruned trees and tests them on valid data outside of the training set. This is done to verify the utility of the resulting trees as classifiers. [2]

![Fig. 1. Cost complexity function with various alpha values. The best complexity is the minimum of this function.](image)

This may seem strange, but quite often the growing algorithm will fit the training data too explicitly. This leads to greater error in practical use, as it will likely have attempted to include outliers in its rules, and pruning tends to reduce overstepping error. This is of extreme importance in highly overlapped or poorly behaved datasets. This is also another reason that discretization will probably not create too much error overall. The algorithm includes this self-checking feature, which removes leaves from the tree. Discretization, properly done, will do the same.

**III. DISCRETIZATION ALGORITHMS**

The purpose of this project is to evaluate the capabilities of the discretization algorithms with the CART Decision Tree Machine Learning Algorithm. We have done this by a variety of methods. Also of primary importance to us is the time saving we will encounter in the growing phase of CART, and thus we measured the total time in growing and pruning each tree, with the full dataset used as a control. Accuracy, of course, is vital to the project, so we include such concerns in our final report on the topic. We have, therefore, two primary measures of how well our discretizers perform: time taken and accuracy. We employ standard statistical methods, determining means...
and standard deviations, and then graphing the results for final presentation.

A. Equal Width

The Equal Width and Equal Frequency algorithms are among the most basic of discretization algorithms. Because of their very fast processing time, and ease of implementation, they are ideal for datasets where comprised accuracy is tolerable. The Equal Width algorithm works by dividing the data set into intervals of equal width, where width is defined over the attribute you wish to discretize the data set on. Each interval is given the value of the attribute that each data point in the interval has is set to the same number. This algorithm is implemented by evaluating the lower and upper bounds of the continuous attribute, and dividing the attribute into \( M/3C \) units, where \( M \) is the number of datapoints and \( C \) is the number of classes in the dataset.

B. Equal Frequency

The Equal Frequency algorithm works by dividing the data set into intervals of equal frequency, where frequency of an interval is defined as the number of data points in the interval. Then, in each interval, the value of the attribute that each data point in the interval has is set to the same number. The advantage of Equal Frequency over Equal Width is that it performs much better on “clumped” data, such as those seen in signal processing applications.

The main advantages of these algorithms as a whole are that they are easy to implement and they are fast. Both algorithms have a simplified computational complexity of \( O(n) \) (not including sorting) which is quite fast when compared to the other, more complex algorithms.

The major disadvantage of these algorithms is also rooted in their simplicity. They are unsupervised and do not take any extra steps to minimize error. The total error of these algorithms is therefore expected to be significantly higher than that of some of the other supervised algorithms. The total error is a measure of the continuous-valued error in training data caused by discretization. It is equal to the sum of the partial errors of each data point. The partial error of a data point is defined as the difference between the original value of the discretized attribute for that data point and the new, discretized value.

B. CAIM

The CAIM discretization algorithm was first reported in a paper by Lukasz A. Kurgan and Krzysztof J. Cios entitled, “Discretization Algorithm that Uses Class-Attribute Interdependence Maximization.” [7] The premise behind discretizing a data set based on class attribute interdependence is simple; the more values of an attribute are related to a certain class, the easier it will be for the decision tree to quickly and correctly classify the data set. The CAIM algorithm was chosen due to its high performance relative to other supervised discretizers (as evident in the literature).

While CAIM internally is one of the more complex of the discretization algorithms that we will be implementing, its function is still the same as the rest. It takes a set of continuous training data and significantly reduces the number of different values that data assumes. It automatically chooses the number of intervals and their boundaries based on a criterion which attempts to group elements of the same class together.

<table>
<thead>
<tr>
<th>Class</th>
<th>Interval</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_i )</td>
<td>( q_{i1} ) ... ( q_{ir} ) ... ( q_{in} )</td>
<td>( M_{+i} )</td>
</tr>
<tr>
<td>( C_{r} )</td>
<td>( q_{r1} ) ... ( q_{r1} ) ... ( q_{rn} )</td>
<td>( M_{+r} )</td>
</tr>
</tbody>
</table>

\( \text{Interval Total} = M_{+1} \ ... \ M_{+n} \ ... \ M_{+r} \)

The table in Fig. 6 represents the discretization quanta matrix for a single attribute of the data set. The value \( q_{ir} \) represents the number of continuous data points belonging to class \( C_i \), that are to be discretized in the interval \( (d_{r-1}, d_r] \).

\[
p_{ir} = p(C_i, D_r | F) = \frac{q_{ir}}{M} \tag{6}
\]

\[
p_{i+} = p(C_i) = \frac{M_{+i}}{M} \tag{7}
\]

The discretization criterion for CAIM for class \( C \), discretization scheme \( D \), and attribute value \( F \) is shown in (9).

\[
p_{+r} = p(D_r | F) = \frac{M_{+r}}{M} \tag{8}
\]
\[
CAIM(C, D | F) = \sum_{r=1}^{n} \frac{\max_{i} q_{ir}^2}{M + r} \frac{M + r}{n}
\]  

(9)

\(n\) = number of intervals in discretization scheme \(D\)
\(r\) = variable that iterates through all the intervals
\(\max_{i} q_{ir}\) = the maximum value of all \(q_{ir}\) values (among all classes)
\(M + r\) = the total number of values (among all classes) that are within \((d_{r-1}, d_{r}]\)

IV. RESULTS

We judge the performance of a discretizer by several methods including: **speedup** of non-discretized versus discretized data, and **accuracy** of non-discretized versus discretized dataset. The speedup of applying the discretization algorithm can be summed as follows,

\[
Speedup = \frac{Time\ to\ classify\ non \text{-} discretized\ dataset}{Time\ to\ classify\ discretized\ dataset + Time\ to\ discretize\ \ data}
\]

(10)

Because discretization can reduce the dimensionality during induction, it can drastically decrease the processing required during the classification induction phase. We applied the aforementioned three discretizing algorithms to the following databases which were obtained from the UCI Repository Of Machine Learning Databases and Domain Theories. [11]. These databases include those used in the CAIM paper [7] as well as five additional databases which have a large number of numerical attributes and a large number of cases.

<table>
<thead>
<tr>
<th>Database</th>
<th>#case</th>
<th>#attribute</th>
<th>#class</th>
</tr>
</thead>
<tbody>
<tr>
<td>ann-thyroid</td>
<td>7200</td>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>pendigit</td>
<td>10992</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>satellite</td>
<td>6435</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>optdigit</td>
<td>5620</td>
<td>64</td>
<td>10</td>
</tr>
<tr>
<td>waveform</td>
<td>5000</td>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>abalone</td>
<td>4177</td>
<td>8</td>
<td>*</td>
</tr>
<tr>
<td>segmentation</td>
<td>2310</td>
<td>19</td>
<td>7</td>
</tr>
<tr>
<td>vowel</td>
<td>990</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>vehicle</td>
<td>846</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>pima-diabetes</td>
<td>768</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>ionosphere</td>
<td>351</td>
<td>34</td>
<td>2</td>
</tr>
<tr>
<td>heart</td>
<td>270</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>iris</td>
<td>150</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Fig. 7. Databases and summary of attributes used for our CART discretization performance.

We compared trees created with undiscretized data, versus those which were discretized before being classified by CART for all three algorithms: Equal Width, Equal Frequency, and CAIM. The following sets of charts show the results of our testing on the Iris, Heart and PIMA datasets.

Fig. 8. Iris database leaf node comparison.

Fig. 9. Iris databases accuracy comparison.
Fig. 10. Iris databases performance comparison (time).

Fig. 11. Heart database leaf node comparison.

Fig. 12. Heart databases accuracy comparison.

Fig. 13. Heart databases accuracy comparison.

Fig. 14. PIMA database leaf node comparison.

Fig. 15. PIMA databases accuracy comparison.
Lastly, we would like to acknowledge and thank all the professors and friends who have supported us, given us permissions to use their work, and in general been there to answer our questions.

REFERENCES


