Applications of Genetic Programming to Building Decision Trees

Cezary Z Janikow
Dept. of Mathematics and Computer Science
University of Missouri - St. Louis
St. Louis, Missouri 63121
janikow@umsl.edu

Sandhya Malatkar
Dept. of Mathematics and Computer Science
University of Missouri - St. Louis
St. Louis, Missouri 63121
Sm9r6@mail.umsl.edu

Contact Author
Cezary Z Janikow
Janikow@umsl.edu

Keywords
Genetic Programming, Constrained Genetic Programming, Decision Trees, Classification, Data Mining

Conference
GEM'11

ABSTRACT

Decision Trees generated with recursive local partitioning methods are widely used in classification problems. In most cases, sub-optimal yet efficient axis-parallel partitioning generates trees of sufficient quality. More complex partitioning is possible, yet it is less efficient and generally not cost effective. Genetic Programming methods have also been used to generate decision trees. In this case, which does not rely on the standard recursive partitioning but rather manipulates explicit trees, the major challenge is to maintain valid trees. This paper uses Constrained Genetic Programming – a method allowing automatic control of the evolved trees. The method is used to evolve axis-parallel, and then also more powerful trees, using the standard Iris dataset. The results show that this method can be used to easily evolve different kinds of trees, and that axis-parallel and also oblique trees offer sufficient quality for this dataset.

INTRODUCTION

Finding a good classifier for a given database is a very common problem in data mining or machine learning. There are several approaches to solve this classification problem, such as rules, neural networks, and decision trees.

ID3 and C4.5 are the two most known top-down partitioning methods for building decision trees [9]. Any such method needs a method to partition the space, using some partitioning quality measure such as entropy. Most partitioning methods work locally, that is partition the space recursively guided by local metrics, and only then possibly apply some global metrics to avoid overfit. If the partitioning is done on one attribute at a time, the resulting partition is axis-parallel. If each partitioning test has binary outcome, the resulting tree is binary.

Decision tree should also be prevented from overfitting to match the training data. This can be accomplished by imposing some constraints on the trees, or by validating the trees with a separate validation data set.

Recently, evolutionary methods, especially Genetic Programming, have been used for evolving decision trees [1][2][3][11]. In this case, decision trees are evolved by explicit operations of candidate decision trees. Then, a major challenge is how to perform such operations and yet preserve the constraints of the trees being used, such as maintaining only axis-parallel or linear trees [1][2][3].

In this paper, we use a Genetic Programming system called CGP, which allows tree manipulations to be easily controlled. We use CGP to evolve axis-parallel, then linear, then oblique trees.

Decision Trees

Decision tree learning is a supervised learning method, using already classified training data. A supervised learning algorithm analyzes the training data and produces a classifier, usually some representation coupled with an inference rule, which can be used to classify new previously unseen data.

Decision trees, especially axis-parallel recursive partitioning methods, are extremely popular due to efficiency, good sub-optimal performance, and ability to easily explain its internals to any observer.

In axis-parallel decision trees, each node of the tree is either an 'if' test or a classification leaf. If it is a test, one attribute is tested against a constant, producing binary outcome (binary tree). Such a tree is shown in Figure 1 (the tree I sbinary, plus the test is shown as a separate node). The testing continues recursively and locally, until a decision node is reached. Classifying new data is simple – observing the features of a datum, follow the tests until a leaf-decision is reached. Such trees can be built very efficiently, yet exhibit very good near-optimal performance, especially when coupled with some method to prevent overfit [4][9].

In oblique decision trees, the test is extended to test two attributes at a time. This corresponds to finer partitioning of the universe of discourse. This is illustrated in Figure 2. Oblique trees can be extended to testing arbitrary linear combinations of attributes. Such decision trees are called linear, and they are illustrated in Figure 3.

Oblique and linear decision trees perform finer partitioning, but sometime lead to overfit rather than to improved classification accuracy. Moreover, finding the optimal test in each node is a much more complex task now, usually not giving enough benefits for the extra costs.

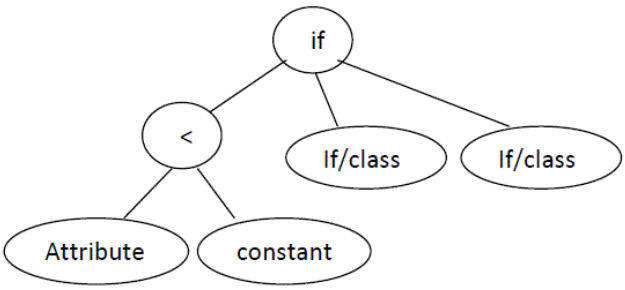

Figure 1. Axis-parallel binary decision tree.

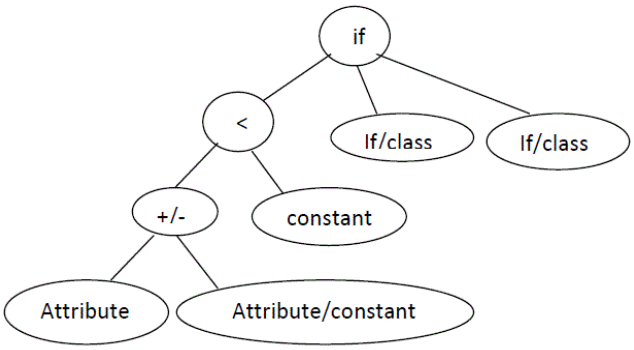

Figure 2. Oblique binary decision tree.

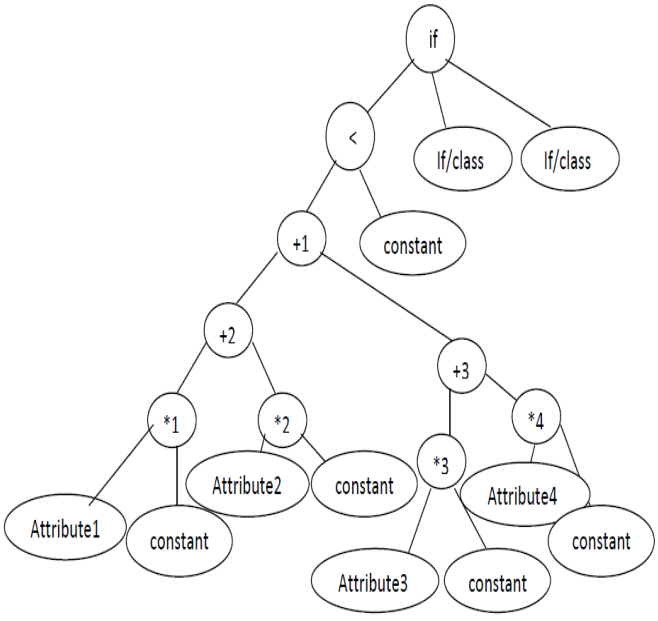

Figure 3. Linear binary decision tree.

An example of a test in an axis-parallel tree, Figure 1, is "if intake temp < 100 then … else …". An example test in an oblique decision tree, Figure 2, is "if intake temp – output temp < 0 then … else …". Note that Figure 2 allows this test to be on just one attribute and 2 constants, reducing the tree to axis-parallel.

Tests in Figure 3 are more complex. In general, linear trees allow any linear combination of attributes for a test. However, to simplify processing while knowing we will deal with problems involving four attributes, we build the test on linear combination of four attributes. This is accomplished by using the specially designated + and * functions: +1 is an addition function that allows its two arguments to be addition functions only; +2 is an addition function that allows its two arguments to be multiplications of two attributes and constants only; +3 is similar on another two attributes. The multiplication functions are similarly constrained to build the tests as shown in Figure 3. In short, these restrictions guarantee that each tree will be linear using all four attributes.

Genetic Programming

Genetic programming (GP) is a technique based on the biological evolution similar to Genetic Algorithms (GA). GP maintains a population of independent solutions, each solution being a tree. GP solves problems by first sampling some potential solutions, using the population, and then mixing the solutions using simulated mutation and crossover while guided by Darwinian selective pressure (the better the solution, the more likely to be maintained and used). This process take place over a number of generations – each generation produces new population of solutions.

When GP is applied to a problem when potential solution is a classifier, the classifier's quality can be determined by measuring the number of errors this classifier makes while classifying the training data.

One obvious problem with applying GP to building decision trees is that the simulated mutation and crossover may not produce desired trees [1][2][3]. For example, if the GP is evolving trees as in Figure 1, after a number of operations the tree may look like that of Figure 2, or possibly even worse as a tree without proper tests or proper decision nodes.

In our study, we will use Constrained GP (CGP) to maintain the desired properties for the trees.

Constrained GP

We use the CGP 2.1.1 [5]. CGP is a methodology to process both strong constraints and weak constraints in GP. Strong constraints are those that absolutely have to be satisfied, such as '<' node (test) must test an attribute against a constant if evolving axis-parallel trees. Weak constraints are not used here – they also allow weights to be assigned to different arguments. CGP uses very simple constraints, called first order, which only allow constraining a node and one of its children at a time. That is, we can restrict '<' to have an attribute on the left and a constant on the right, but we cannot restrict it to use only one attribute and one constant.

CGP 2 uses data typing for constraints, which can be used here easily – using types we can easily restrict nodes such as '<' to use any attribute on the left and any constant on the right by assigning one type to attributes and another type to constants. Once typing is assigned, and validity rules are entered into the system, CGP guarantees to only evolve valid trees, and does it with minimal overhead [5].

Empirical Study

Experimental Setup

The function sets are presented in Table 1, 2 and 3, and the terminal set is presented in Table 4 – the terminals correspond to the attributes of the Iris data set. In addition, real number terminals are also used. The GP parameters are presented in Table 5. All experiments were conducted and averaged using 10-fold cross-validation.

Table 1. Function Set for Axis Parallel Decision Trees

Function	No. of Arguments	Data Type	Return Type
If	3	Boolean, Class type, Class type	Class type
<	2	Real, Real	Boolean

Table 2. Function Set for Oblique Decision Trees

Function	No. of Arguments	Data Type	Return Type
If	3	Boolean, Class type, Class type	Class type
<	2	Real, Real	Boolean
+	2	Real, Real	Real

Table 3. Function Set for Linear Decision Trees

Function	No. of Arguments	Data Type	Return Type
If	3	Boolean, Class type, Class type	Class type
<	2	Real, Real	Boolean
+1	2	Real, Real	Real
+2	2	Real, Real	Real
+3	2	Real, Real	Real
*1	2	Real, Real	Real
*2	2	Real, Real	Real
*3	2	Real, Real	Real
*4	2	Real, Real	Real

To prevent GP from producing large trees, with possible unexpressed subtrees, we use the same method as used in [2] – we add penalty based on the number of nodes in the tree. We have tested various penalty levels and we have found that penalty for exceeding 40 nodes to be the best performer here.

Table 4. Terminal Set

Terminal	Return Type
X1 (Petal Length)	Real
X2 (Petal Width)	Real
X3 (Sepal Length)	Real
X4 (Sepal Width)	Real

Table 5. GP Parameter Settings

Parameter	Setting
Population Size	500
Number of Generations	50
Fitness Cases	150 (Depends on the database we use)
Selection operator	Tournament Selection of size = 7
Crossover rate	0.9
Mutation rate	0.05
Tree type	Half and half

GP vs. CGP

First, we compared performance of standard GP against that of CGP, while evolving axis-parallel decision trees. While CGP can use constraints to evolve only trees as those of Figure 1, standard GP will evolve any tree made of the labels. Such trees were subject to evaluation given carefully designed extended interpretations. Figure 4 shows the results - it is clear that the trees produced using GP i.e. without constraints are not as good as the trees evolved with. CGP produced only valid axis-parallel, as illustrated below:

If(X2 <1.93543) then setosa,

Else If(X1 <1.57859) then versicolor else virginica

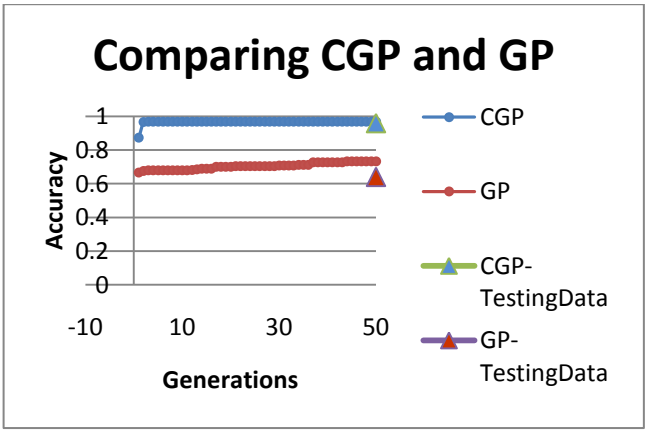

Figure 1. Comparing GP and CGP performance with axis-parallel decision tree.

Moreover, CGP produced small and high quality trees, Table 6 compares those generated here with those produced with C4.5 as reported in [4].

Table 6. Axis Parallel Decision Trees Comparison

Algorithm	Predicted Accuracy on testing data	Number of tree nodes
C4.5	94.0%	5.0
CGP	96.0%	5.0

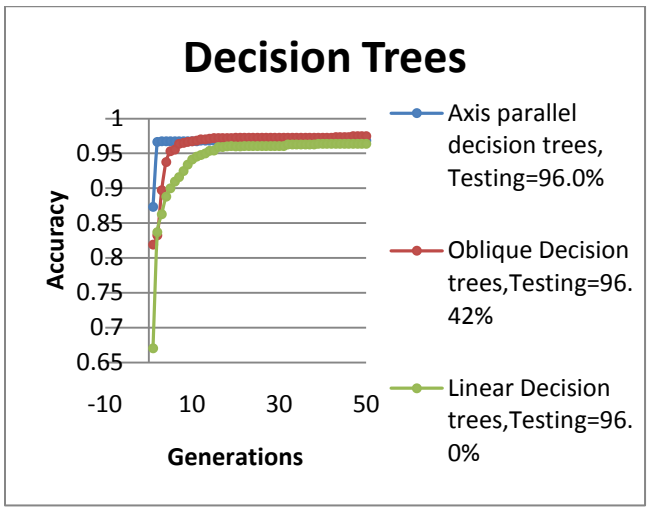

Figure 2. Comparing different comlexity decision trees.

Table 7. Decision Trees with penalty on nodes>40

Decision Tree	Accuracy on total training and testing data	Average number of error on total training and testing data
Axis Parallel	96.73%	4.9
oblique	97.06%	4.4
linear	96.4%	5.4

The next experiment compared axis-parallel, oblique, and linear decision trees as evolved with CGP. The results are presented in Figure 2 and Table 7. As seen, axis-parallel decision trees are the easiest to evolve, as it takes the fewest CGP generations to evolve high quality trees. However, oblique decision trees give better overall accuracy. Below is an example of a generated oblique decision tree, with only 3 errors.

(if (< (+ X1 2.00297) 3.79673)

(if (< (+ X2 0.13303) 2.45623)

(if (< (+ X2 X1) 3.10213) setosa setosa)

(if (< (+ X2 X1) 6.86323) versicolor virginica)) (if (< (+ X2 X1) 2.08085) versicolor virginica))

Conclusion

We have studied applying CGP to evolve different kinds of decision trees for the Iris dataset. As expected, CGP evolved trees outperform those generated by a standard GP. Among different kinds of trees evolved by CGP, the simplest axis-parallel trees were the easiest to evolve quickly, but the more complex oblique decision trees produced better accuracy when tested using 10-fold cross-validation. Even more complex linear trees did not outperform oblique trees, for this data set.

References

[1] Jose L. Alvarez, Jacinto Mata and Jose C. Riquelme (2001), "CGO3: An Oblique Classification System Using An Evolutionary Algorithm and C4.5".

[2] Martijn Bot (1999), "Application of Genetic Programming to Induction of Linear Classification Trees".

[3] Jeroen Eggermont, Joost N. Kok and Walter A. Kosters, "Genetic Programming for Data Classification: Partitioning the search space".

[4] Cezary Z. Janikow and Maciej Fajfer, "Fuzzy Partitioning with FID3.1".

[5] CGP lil-gp 2.1.1; 1.02 User's Manual, Version March 16, 2007, Cezary Z. Janikow, Scott DeWeese.

[6] Thomas Loveard and Victor Ciesielski, "Representing Classification problem in Genetic Programming".

[7] Kamel Faraoun and Aoued Boukelif (2005), "Genetic Programming Approach for Multi-Category Pattern Classification Applied to Network Intrusion Detection".

[8] Daniel Rivero, Jaun R. Rabunal Dorado, Alejandro pazos and Nieves Pedreira, "Extracting Knowledge from databases with Genetic Programming: Iris Flower Classification problem".

[9] J. R. Quinlan (2006), "Improved use of Continuous Attributes in C4.5"

[10] Maria-Luiza Antonie and Osmar R.Zaiane, "An Associative Classifier Based on Positive and Negative Rules".

[11] Satchidananda Dehuri, Sung-Bae Cho (2008), "Multi-objective Classification Rule Mining Using Gene Expression Programming".

[12] K. Hima Bindu, P.S.V.S Sai Prasad and C.Raghavendra Rao (2010), " Hybrid Decision Tree Based On Inferred Attribute".

[13] Anthony K.H. Tung, Xin Xu and Beng Chin Ooi (2005), "CURLER: Finding and Visualizing Nonlinear Correlation Clusters".