ISSN:2231-0711
www.ijcset.net

Constructing Near Optimal Binary Decision Tree Classifier Using Genetic Algorithm

Ayman Khalafallah
Department of Computer and Systems Engineering
Faculty of Engineering Alexandria University
Alexandria, Egypt
ayman.khalafallah@gmail.com

***Abstract*—Decision Trees are extensively used in classification, pattern recognition and Data Mining. Classical decision tree building algorithms Iterative Dichotomiser 3 (id3) [3] uses one attribute to test at each internal node, resulting in the decision boundaries being parallel to the axis and builds the tree one node at a time. In this paper a new method is proposed where at each internal node a hyperplane is selected based on all attributes, this hyperplane partitions the training set into two disjoint sets. Our method also tries to build most of the tree in a single optimization problem. Genetic Algorithm is used as the optimization methods. The resulting tree using the proposed algorithm may be more compact and accurate in the classification problems.**

Keywords : Decison Tree, Genetic algorithm, classifier, Optimization

I. Introduction

A classification problem is a problem where a label is assigned to an object based on this object attributes. A classical example is character recognition when a character is scanned at the scanned image is used to classify the character. This attributes might be the size the orientation, the contour or the scanned black and white matrix of the character. Classification problems are widely used in biology for example to detect possible diseases based on the outcome of a gene test.

To solve the classification problem a classifier has to be built, the classifier is built using training examples. In the case of supervised training considered in this paper, the training examples consist of objects identified with their attributes and labeled with their classes. The training process of the classifier is called learning process.

Decision trees are an important tool in decision making, pattern recognition, classification and data mining. Because of their importance many algorithms exists to build decision trees [1][2][3][6]. The purpose of these algorithms is to build as compact as possible decision tree to classify the data from the training examples. It is believed that a shorter decision tree is the best possible tree according to Occam Razor principle[1][2][3]. It has to be noted though that constructing an optimal decision tree is an NP-hard problem [4][5] so one must accept a heuristic to build a near optimal search tree. The references [6][7][10] show many attempts to build near optimal decision tree.

In this paper a new method to construct near optimal binary decision tree is presented this method defers from the above methods in the following:

- At each node a hyperplane test is used to decide on which branch to follow not just a single attribute test.
- The algorithm tries to construct a complete binary tree to make sure that the tree is as compact and balanced as possible.
- The algorithm does not attempt to completely construct the decision tree in order to reduce the complexity of the chromosome representation in Genetic algorithm.

II. Genetic Algorithm

A Genetic algorithm (GA) is an optimization method that tries to mimic the natural selection and mutations of living organisms [9]. The genetic algorithm is an iterative optimization technique in which a pool of possible alterative solution to an optimization problem is maintained. This pool is called the population and then using the basic GA operations which are Cross Over and mutation a new generation of the population is produced[9]. Genetic algorithm is widely used in optimization because it combines to benefits of random and steepest decent like search and it is a powerful mix between exploration and exploitation.

To be able to use genetic algorithm possible solution of the problem has to be encoded as a chromosome. A chromosome in GA is nothing than an array of bits or digits. The basic operation of GA is cross over and can be described as in Figure 1 Cross Over in GA

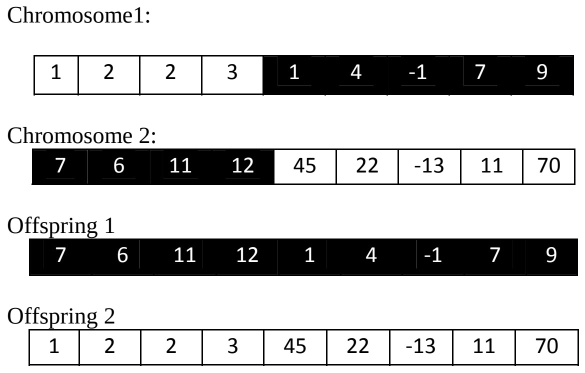

Figure 1 Cross Over in GA

Here a random point is chosen and the two chromosomes change content at this random point. Another GA operation is mutation where a single bit or digit changes value randomly, typically, with a very small probability. Crossover make sure that a new population is being generated from the old one while mutation ensure diversity and make sure that the algorithm is not trapped in a local minimum.

Important to the Genetic algorithm is what is called a fitness of the

chromosome. The fitness of chromosome is problem dependent and depends on the kind of the optimization problem and whether it is a minimization or the maximization. According to the survival of the fittest the higher the fitness value of the chromosome, the more the possibility of it or its offspring survival to the next population.
Many parameters have to be chosen for GA including crossover and mutation probabilities, population size, crossover type and number of generations. But the most important part is how to represent a possible solution as a chromosome and how to evaluate its fitness.

III. DECISON TREES

A decision tree is a tree data structure with the additional properties that at each node there is a test of a set of attributes to decide which branch to follow and also each leaf is labeled with the name of a class. The root and each internal node are called decision nodes. The node attributes' check can be looked at as a partition of the search space. Classical decision tree building algorithms [3] uses a single attribute range tests at each internal node and this will result in each node parathion the search space parallel to the basic axis. A quick look at Figure 2 will show that this partition might be highly inefficient and generally a hyperplane test of the following equation might be more useful.

$$(a_0 + \sum_{i=1}^{n}(a_i x_i))$$

Using the above equation samples are classified to the left or right node branch depending on the sign of above term. In this equation x_i's are the set of attributes and a_i's are the set of weight the algorithm must find in order increase the efficiency of the hyperplane cut.
There are mainly three alternatives to build decision trees, the main objective of each method is to build as compact as possible decision tree in hoping that it will be the most useful tree according to the debated but highly accepted Occam's Razor[3] principle. These three are

1. Build the tree one node at a time using a selected attribute at each node to partition the search space.
2. Use hyperplanes to partition the search space.
3. Build the tree as a single optimization problem.

The Survey in [6] gives more concrete examples about those methods.

Any of these three approaches has its merits but suffers from some shortcomings. The first alternative is the simplest and tends to produce trees that rules can be extracted from them easily but looking at Figure 2, it will become clear that trees built using this method might be not optimal. Method1 though is the preferred method because the resulting tree and the rules associated with traversing the tree is easier to read and interpret.
Method2 although superior at each node may suffer from local anomalies as the cuts defined at upper levels may result in bad cuts later down the tree. Method3 is superior but suffer from the deficiency in Method1 though tends to be more robust. Also the optimization problem associated with method3 turns to be much more difficult and no guarantee of reaching the global optimization goal in a reasonable amount of time. Also allowing the tree to be of arbitrary structure tends to render the problem very difficult and time consuming and the gain from such method might not justified.

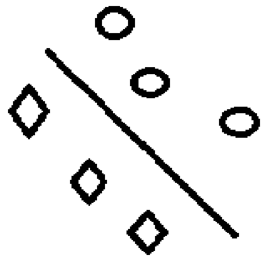

Figure 2

IV. PROPOSED ALGORITHM

The proposed method tries to overcome all the previous shortcomings. The method builds a binary search tree using hyperplane as the separation method at each node. Because of the complexity of optimizing both the tree structure and the hyperplane simultaneously a decision was made to look for a particular binary tree structure namely a complete binary tree. Complete binary tree as shown in Figure 3 is a binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible. The number of leafs is chosen to be the number classes. Which leads to the fact that the number of decision nodes to be equal to number of classes minus one.
Any binary decision tree will have a number of leafs greater than or equal to the number of classes and a complete binary tree has the shortest length and least average length of all binary trees of given size making it the most compact of all decision trees. Another advantage of complete binary tree is that it has a compact array representation the following array is a representation of the tree in Figure 3

A	B	C	D	E	F	G	H	I

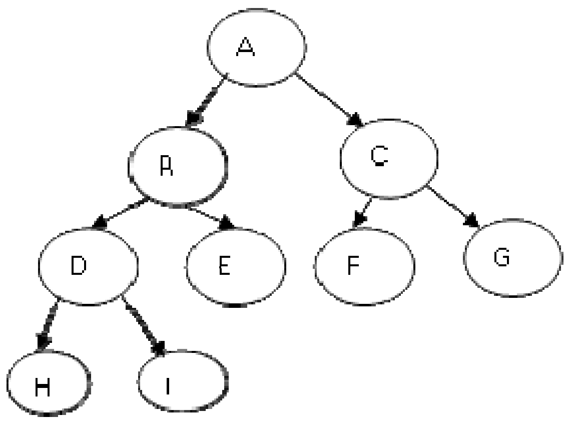

Figure 3

Basically the children of node at position i at the array if they exist are at position 2i and 2i+1. This achieves two goals

1. Traversing the tree is easy.
2. Encoding the tree as a chromosome in GA is simple.

The first step of the Algorithm uses GA to find a possible optimal search tree of this structure and as noted above the tree representation as a chromosome is simple.
Since there is no guarantee that neither such tree exist as a solution to the classification problem nor the GA will be able to find one if one actually does exist, another method has to be carried from the leafs of the tree found in step1 of the algorithm. Method2, namely finding a hyperplane cut at each leaf node of the tree in step 1, is chosen to complete the tree with GA as the optimization method. Figure 4 shows a possible tree resulting from step 1 identifying the places where the hyperplane cuts must be found by the number 1 to 7.

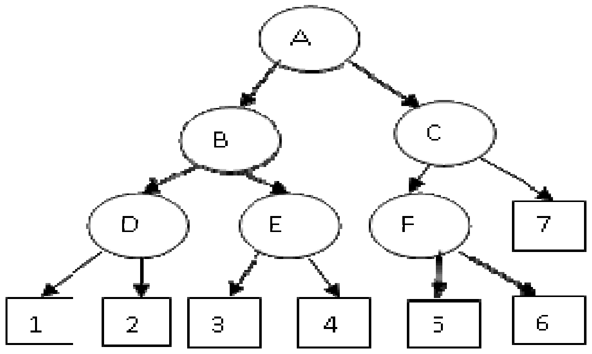

Figure 4

The last part is chosing the fitness function. For step 2 the usual entropy based[3] fitness is used. Step 1 fitness is a bit tricky though as near complete classification is preferred over minimum entropy so the fitness function is defined as follows: first the number of classes staying alive (having a probability of reaching this leaf more than 2%) at each leaf and this number is denoted by v, second the fitness of each leaf is ((2-v)/total number of classes). The fitness of the whole tree is the sum of the fitness of its leafs.
The logic behind that is that if a leaf node has only one class it will not need to be extended in step 2 which is best possible. If it has two classes classical methods in pattern recognition can separate them and the contribution of this node to fitness of the tree is zero. While if a leaf has more than two classes it will need more than on level in step 2 of the algorithm and this a bad outcome so it is assigned a negative value and it is subtractive to fitness of the tree. The more the classes that end alive in a leaf node the less its fitness value. Finally in Table I a simple pseudo code of step 1 of the algorithm is given as step 2 is more or less straight forward.

TABLE I. STEP 1TREE BUILDING PSEUDO CODE

```
function classify(chromosome Ch, Pattern t){
i = 1
While (i<Ch.length){
  If <t,Ch> >0
     i = 2i
  else
   i = 2i+1}
 Return i - ch.length-1}
Function Evaluatefitness(Chromosome Ch){
  For every pattern t
    classify(ch,t)
 return the fitness as described in the text above}
Procedure InitializePopulation(){
  Generate an array of arrays of random input;
  // Holding Array size is the population size
  // Internal  arrays size are chromosome size}
Procedure Crossover(Chromosome: CH1,CH1){
 // Do Crossover between CH1 and CH2}
Procedure Mutation(Chromosome CH){
 // Mutate Chromosome CH}
Procedure GA(){
  InitializePopulation()
  Repeat
     Repeat
        Pick two Chromosomes CH1,CH2
        Crossover(CH1,CH2)
        Evaluate fitness(OffSpring1)
        Evaluate fitness(OffSpring1)
    EndRepeat
      Mutate Some chromosomes
  End }
```

V. CONCLUSION AND FUTURE WORK

In this paper a novel method for constructing a near optimal binary search tree is given. Arguments are given to show why this method should be robust. Experimental results is needed to confirm the benefits of the proposed algorithm. Also experiments must be carried out to test different fitness function. The possibility of pruning the tree resulting from the first part of the algorithm before the second step has also to be explored.

REFERENCES

[1] Mitchell, T. M., ”Machine Learning”,McGraw-hill, 1997.

[2] Duda, R. O., Hart, P. E., and Stork, D. G., Pattern Classification, 2nd Ed., Wiley interscience, 2001.

[3] Quinlan, J. R., Induction of decision trees, Machine Learning, 1(1), 81-106.

[4] L. Hyafil and R. L. Rivest, Constructing optimal binary decision trees is NP-complete , Information Processing Letters, Vol. 5, No. 1, 15-17, 1976.

[5] Bodlaender, L.H. and Zantema H., Finding Small Equivalent Decision Trees is Hard, International Journal of Foundations of Computer Science, Vol. 11, No. 2 World Scientific Publishing, 2000, pp. 343-354.

[6] Safavian, S.R. and Landgrebe, D., A survey of decision tree classifier methodology,IEEE Transactions on Systems, Man and Cybernetics, Vol 21, No. 3, pp 660-674, 1991.

[7] Gehrke, J., Ganti, V., Ramakrishnan R., and Loh, W., BOAT-Optimistic Decision Tree Construction, in Proc. of the ACM SIGMOD Conference on Management of Data, 1999, p169-180.

[8] Zhao, Q. and Shirasaka, M., A Study on Evolutionary Design of Binary Decision Trees, in Proceedings of the Congress on Evolutionary Computation, Vol 3, IEEE, 1999, pp. 1988-1993.

[9] Goldberg D. L., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[10] S. Cha and C. C. Tappert, Constructing Binary Decision Trees using Genetic Algorithms, in Proceedings of International Conference on Genetic and Evolutionary Methods, July 14-17, 2008, Las Vegas, Nevada. Systems, Systems, IEEE Systems, Man, and Cybernetics, Vol. 21, No. 5, pp.1231-1238, 1991