Decision Tree Induction Using Genetic Programming

Gökhan Tür and Halil Altay Güvenir
Department of Computer Engineering and Information Science
Faculty of Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey
{tur,guvenir}@cs.bilkent.edu.tr

Abstract. A decision tree induction algorithm using genetic programming (GP) is presented. The best decision tree is defined as the one which achieves maximum accuracy with minimum number of internal nodes. In this approach every individual is a decision tree candidate. The results are satisfactory in the sense that it can find the optimum solution, i.e. the best decision tree, for a small sized dataset. For the dataset related to American Congress, this algorithm can reach an accuracy of 97.3%. Solutions to unknown or noise data are also proposed in this framework.

Keywords. Machine Learning, Genetic Programming, Decision Trees, Genetic Algorithms, Artificial Intelligence

1 Introduction

Genetic algorithms (GA), by combining the survival of the fittest among string structures, (called individuals) with a randomized genetic information exchange, try to form a search algorithm similar to the evolution process in the nature. Algorithm begins with a population of randomly generated individuals, and the "fitness" of each individual is measured. The fittest individuals in the first generation have a higher number of offsprings in the second generation. GA is covered in detail in [2] and [3].

Genetic programming (GP) is a method of "Adaptive Automatic Program Induction", originally created by John Koza and James Rice of Stanford University [4]. GP is similar to GA, except that, individuals are no longer strings, but parse trees of the programs. The programs are composed of elements from a *function-set* and a *terminal-set*, which are typically fixed sets of symbols selected to be appropriate for the solution of problems in the domain of interest. Initial population is created randomly. The genetic information exchange is done by taking randomly selected subtrees in the individual programs and exchanging them. This corresponds to the *crossover* operation in GA. Because of the

closure property of the functions and terminals, this genetic crossover operation always produces syntactically legal parse trees. A random change in the individuals, *mutation* also takes role in this algorithm, but not so frequent.

As an example, consider a function set of $f = \{+, -, \times, /\}$ and terminal set of $t = \{a, b, c\}$. Consider a program that computes '$a + (b \times c)$'. The parse tree of such a program would be as shown in Figure 1. Note that this parse tree is considered to be one individual. Its internal nodes are predefined functions of the terminals in the leaves. The function set, the terminal set, and the fitness function is completely dependent on the specific GP application, and there is no limit in the number of arguments.

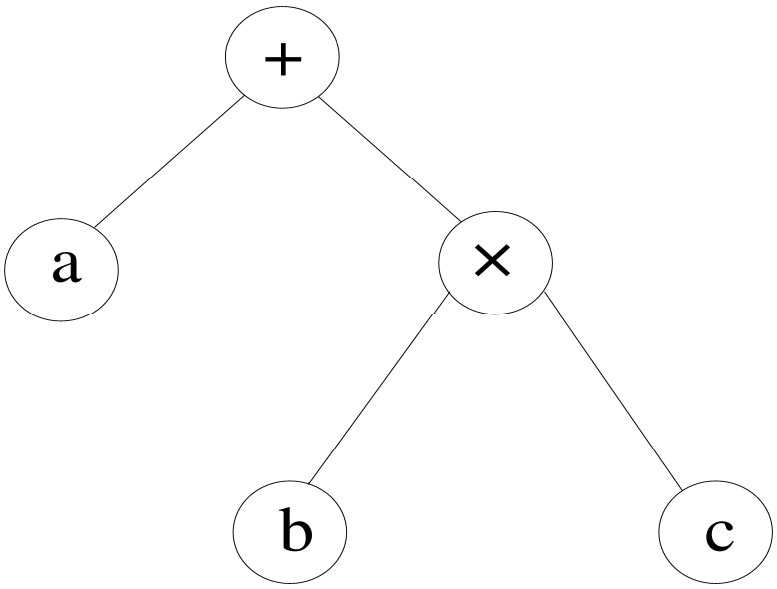

Figure 1: A parse tree example.

Decision trees are first used in Machine Learning by Quinlan [5]. The main usage of decision trees is *classification* (assigning things to categories or classes determined by their properties). A decision tree is a structure that is either a *leaf* indicating a class, or a *decision node* that specifies some test to be carried out on a single attribute value.

A decision tree can be used to classify a case by starting at the root of the tree and moving through it until a leaf is encountered. At each non-leaf decision node, the case's outcome for the test at the node is determined and attention shifts to the root of the subtree corresponding to this outcome. When this process finally (and inevitably) leads to a leaf, the class of the case is predicted to be that recorded at the leaf [5].

Most of the decision tree induction algorithms use greedy heuristics, which select at each step whatever test contributes most to accuracy. For example, C4.5 algorithm selects the attribute that maximizes the information gain ratio. Other algorithms, like EG2, CS-ID3, or IDX introduces costs of the tests besides information gain ratio. In fact, using greedy heuristics is a limitation of these algorithms, because they suffer from the *horizon effect*. This term is taken from the literature on chess playing games. Suppose that a

chess playing program has a fixed three-move lookahead depth and it finds that it will lose its queen in three moves if it follows a certain branch of the game tree, so it can choose another branch in which it will lose its pawn instead. But now, again lose of the queen can be three moves ahead. So the program foolishly sacrifices its pawn. This is the same situation in all of the hill-climbing algorithms [6].

These algorithms also need to prune the resulting decision tree. Pruning is removing those parts of the tree that do not contribute to classification accuracy on unseen cases, producing something less complex and thus more comprehensible [5]. As stated in the definition of the pruning operation, decision trees, created by the greedy algorithms are too complex that test cases can be misclassified, and these trees can not be easily handled by humans. Another reason is that, sacrifying some accuracy can result in a lot of benefit in the number of tests. So it is somehow necessary to balance these two issues: number of tests (in fact, number of internal nodes in the tree) and accuracy of the tree.

Although the idea of using genetic algorithms in the induction of decision trees is a very new concept, classification algorithms using genetic algorithms exist in the literature since 1986. In order to overcome the horizon effect, Turney uses genetic algorithm in the construction of a decision tree [6], but it uses EG2 algorithm. Genetic algorithm is only used in finding out the parameters existing in EG2, and it has nothing in the number of tests.

The following section intrduces an alternative approach to the induction of decision trees, then experimental evaluation of this approach is presented in Section 3. The last section concludes the overall presentation.

2 Algorithm

A more effective way of overcoming the limitations of standard greedy decision tree induction algorithms can be the usage of genetic programming. Each individual of the population in GP can be a decision tree. The functions to be used in the GP are the attributes of the decision tree and classes form the terminal set.

Consider the following example: There is a dataset of 10 senators, and their votes on five different areas of law. This vote can be either yes(y) or no(n). There are two classes of senators. Republicans and Democrats. Each voting is considered to be an attribute of a senator, so the resulting table is given in Table 1.

In fact, this is an artificial dataset and corresponds to the decision tree in Figure 2.

Since each attribute can hold just 2 values (yes and no), it is not necessary to define them, each left branch means no, each right branch means yes. So using an ordered tree is enough to hold that information. Terminal set has two elements: republican, democrat The interesting point is that, functions (in fact the attributes) to be used in the genetic programming does not return any value, because they are simply the decision points of the tree. If an attribute can have more than 2 values, nothing has changed.

CLASS	A0	A1	A2	A3	A4
democrat	n	y	n	y	y
democrat	y	n	y	y	y
democrat	n	y	y	y	y
democrat	y	n	n	y	y
democrat	n	y	n	y	y
republican	n	n	y	n	n
republican	n	n	y	y	n
republican	y	n	n	n	n
republican	n	n	y	n	y
republican	y	n	y	n	y

Table 1: Voting Dataset

The only problem left is the fitness function. As stated before, the fitness function must contain both the accuracy of the decision tree and the number of tests for each sample. Accuracy is the ratio of the number of correct classifications to the total number of samples tried on a given decision tree. The second component, i.e. the size of the decision tree can be found by calculating the internal nodes.

The selection of the more important one is critical: the accuracy or the size of the tree. So the following fitness function is used:

$$fitness = 1 - (w \times \frac{1}{numberofinternalnodes+1} + (1-w) \times accuracy)$$

The weight (w) varies between 0 and 1. The algorithm tries to make the fitness function minimum, i.e. 0. It can be seen that the value of the fitness function varies also between 0 and 1. It both tries to maximize the accuracy and minimize the number of internal nodes. The effect of w, in the formula is discussed in the next section, as well as other issues.

Upto this point, we did not care about any noise in the dataset. This problem is solved as follows: If we have any attribute that is wrong indeed, the algorithm does not suffer from it, because it only wants to increase the accuracy, so if such values are not too much, this is not a big problem. But if the value of an attribute is not known, then this is a problem, because the algorithm does not know, from which branch it must continue, when it reaches that attribute in the decision tree. Such attributes must be *ignored*. This is done by traversing all of the subtrees of that internal node. If any branch leads to a leave, which is the same as in the training dataset, then it is accepted, but if neither of them reaches such a solution, then it is considered to be a wrong sub-decision tree for that instance. Of course, this could lead us to an exponentially growing search space, but if the ratio of such instances are very few, it is acceptable. (in fact, only 4.1% of the data is missing in the dataset used in the experiments.)

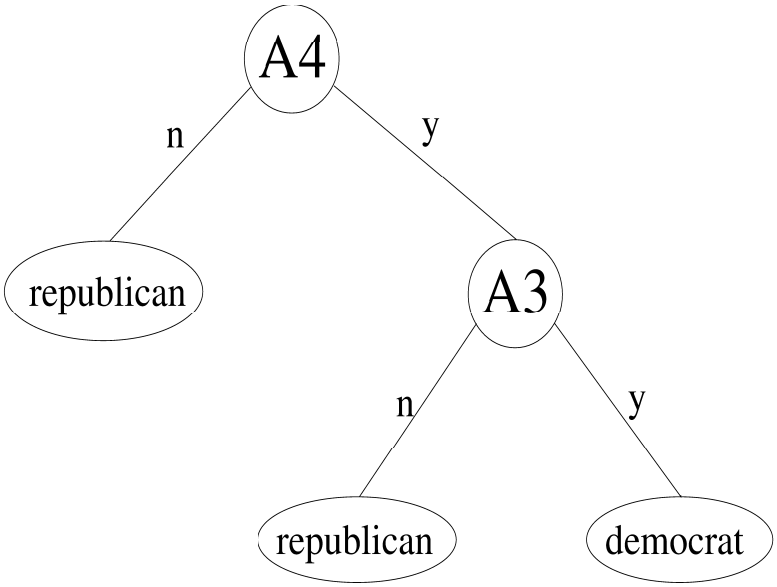

Figure 2: A decision tree.

3 Empirical Evaluation

The algorithm is implemented using the SGPC (Simple Genetic Programming in C), which is developed by Walter Alden Tackett and Aviram Carmi in 1993. In fact, it does the same thing as the application developed by Koza and Rice in LISP and provided to the public, but this version is faster due to the usage of C in the implementation. It requires two datasets, one for the evaluating the fitness of the population, and the other is for validating the result. So, first dataset corresponds to a training set, and the other to a test dataset. According to the application of the fitness function to the data in the training dataset, the best decision tree in that generation is found, and then applied to the testing dataset in order to *validate* the result. In the implementation, the population is consisted of 100 individuals.

This approach is first tried on an artificial dataset, in fact the dataset shown in Table 1. The corresponding subtree is found after 40 generations, and running time is 12.13 seconds. It is the best decision tree, and its fitness is 0 if w is 0, and $1 - \frac{1}{3}$, a relatively high value if w is 1, i.e. if accuracy has no importance.

Then this approach is tried for the actual dataset of the voting in the congress. It is the 1984 United States Congressional Voting Records Database, taken from the Congressional Quarterly Almanac, 98th Congress, 2^{nd} session 1984, Volume XL: Congressional Quarterly Inc. Washington, D.C., 1985, by Jeff Schlimmer. This dataset includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The CQA lists 9 different types of votes: voted for, pired for, and announced for (these three simplified to *yes*), voted against, paired against, and announced against (these three simplified to *no*, voted present, voted present to avoid conflict of interest, and

did not vote or otherwise make a position known (these three simplified to an *unknown* disposition). The number of instances is 435 (267 democrats, 168 republicans), and the number of attributes is $16 + classname = 17$ (all boolean valued). The actual names of the attributes are as follows:

1. Class Name: 2 (democrat, republican)
2. Handicapped-infants: 2 (y,n)
3. Water-projest-cost-sharing: 2 (y,n)
4. Adoption-of-the-budget-resolution: 2 (y,n)
5. Physician-fee-freeze: 2 (y,n)
6. El-salvador-aid: 2 (y,n)
7. Religious-groups-in-schools: 2 (y,n)
8. Anti-satellite-test-ban: 2 (y,n)
9. Aid-to-nicaraguan-contras: 2 (y,n)
10. Mx-missile: 2 (y,n)
11. Immigration: 2 (y,n)
12. Synfuels-corporation-cutback: 2 (y,n)
13. Education-spending: 2 (y,n)
14. Superfund-right-to-sue: 2 (y,n)
15. Crime: 2 (y,n)
16. Duty-free-exports: 2 (y,n)
17. Export-administration-act-south-africa: 2 (y,n)

This dataset is divided into 2 parts, 300 instances are used in training and 135 in the testing. The accuracy of the decision tree is found using the testing dataset.

The results are dependent on the parameter w, weight in the fitness function. For weights 0.00, and 0.25, i.e. where accuracy is more important, the algorithm finds out a solution with an accuracy of 97.3% with only one internal node. This result is obtained by examining $100 \times 5 = 500$ decision tree candidates for a weight of 0.25, and $100 \times 9 = 900$ candidates for a weight of 0.00. Each examination is done on the 300 instances. The whole process is done in less than 4 seconds. Table 2 shows the best results and the running time of the algorithm to obtain these results. Note that an accuracy of 62.3% for a weight of 0.75 is nothing but choosing everyone as a *democrat* (There are 267 democrats among 435 senators in Congress). A more explanatory chart is given in Figure 3.

Weight	Best Accuracy (%)	Running Time (secs)
$w = 0.00$	97.3	3.69
$w = 0.25$	97.3	1.91
$w = 0.50$	85.0	0.19
$w = 0.75$	62.3	0.32

Table 2: The best results for each weight (w) parameter

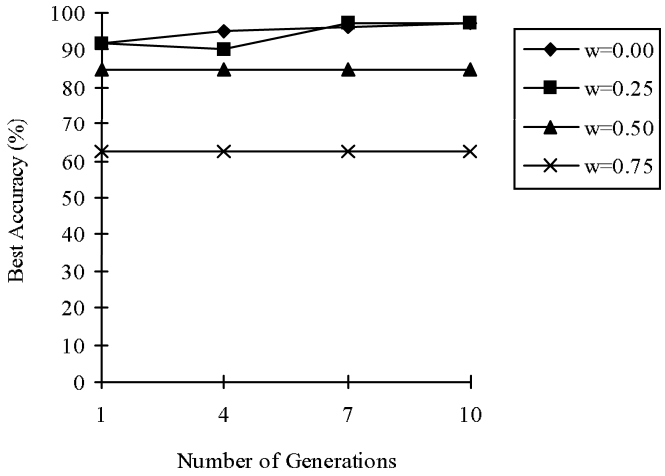

Figure 3: The results of the GP application.

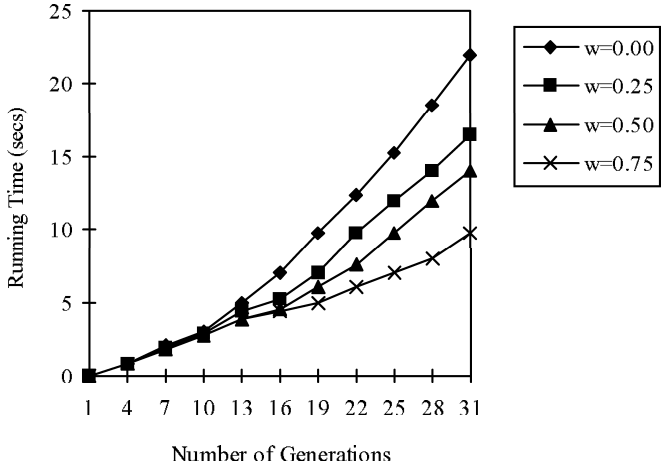

Figure 4: Running time with respect to number of generations.

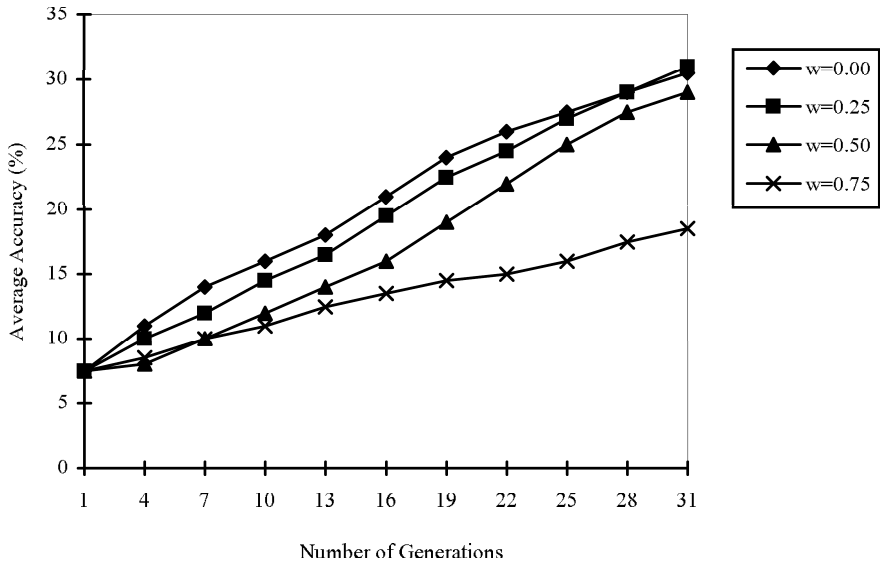

Figure 5: Number of internal nodes with respect to number of generations.

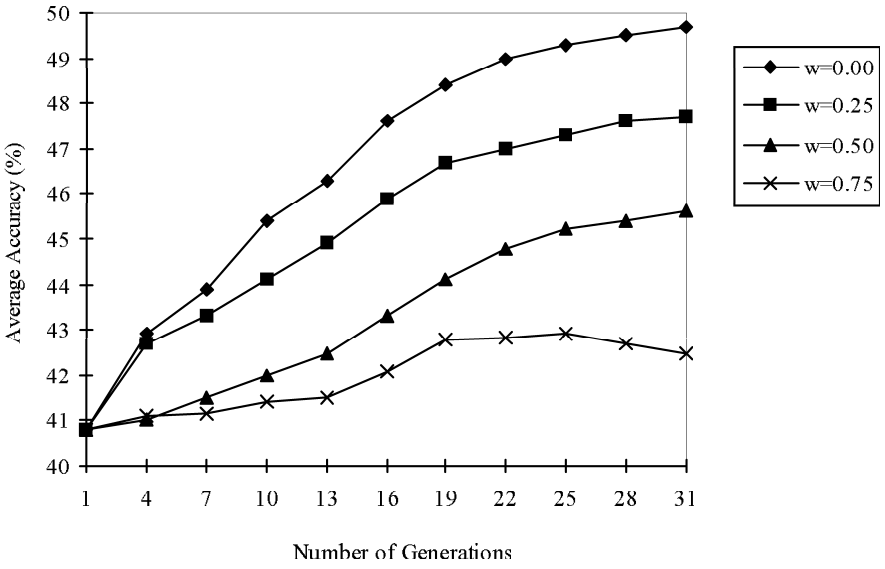

Figure 6: Accuracy in percentage with respect to number of generations.

The second experiment on this dataset is the calculation of average accuracy, average number of nodes and running time with respect to various w (weight in the fitness function) values. Note that, although a best result is found for a given weight, the average accuracy increases more slowly. This is a typical case in all genetic algorithms. The average number of nodes increases proportional to the number of generations. The reason is that, the decision trees created are becoming more complex after some generations, and that is also the main reason for the non-linearly increasing running time of the algorithm. Figures 4, 5,and 6 summarize these results.

4 Conclusion and Future Research

Decision trees can be created using genetic programming. The results are satisfactory, in the sense that it can find the optimum solution, i.e. the best decision tree, for a small sized dataset. For the dataset related to American Congress, the best accuracy is 97.3% with an internal node size of 1, the result of Quinlan for the same test dataset of 135 instances is 94.8% with an internal node size of 25 before pruning, and 97% with an internal node size of 7 after pruning. For both issues, the results of our approach is better. Of course, the same algorithm must be tested on other various datasets. The intuition is that, it must be better than other decision tree induction algorithms, using greedy searches. The reason behind is that, this approach does not suffer from the horizon effect and can find alternative solutions, without looking at the same optimum.

Also, this approach is not so slow. It finds out the best solution for each weight parameter in a couple of seconds.

Since, it is the first attempt of using a genetic programming in the construction of decision trees, it is only implemented to datasets whose all attributes are nominal. For example, in the dataset used in the experiments, the attributes can be yes (y), no (n), or missing (?). Handling of other kind of attributes is an open research area.

Behaviour of this approach to noise is also explained. Alternative solutions to this problem may be provided. In fact, this is the same behaviour that Turney handles delayed tests in a decision tree [6].

This apporach is not specific to decision tree induction, there can be many areas that are suitable for genetic programming. A recent example is 4-Operations [1].

One interesting point is that, in the very first generations, algorithm finds out the best result, and then only very small improvements are possible. This is contradictory to the typical genetic algorithm behavior.

Genetic Programming is known to have some restrictions in some applications. Fortunately, this is not the case for the induction of decision trees.

Including other features, such as costs of the tests or costs of misclassification errors, which are especially important in medical applications, into the decision tree may be interesting.

References

[1] T. Aytekin, E.E. Korkmaz, and H.A. Guvenir. An application of genetic programming to the 4-op problem using map trees. pages 28–40, Armidale, Australia, November 1994. Workshop on Evolutionary Computation in Association with 7^th Australian Joint Conference on Artificial Intelligence.

[2] D. E. Goldberg. *Genetic Algorithms in Search Optimization, and Machine Learning.* Addison Wesley, 1989.

[3] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[4] J. Koza and J. Rice. *Genetic Programming on the Programming by means of Natural Selection.* The MIT Press, Cambridge, 1992.

[5] J. R. Quinlan. *C4.5 Programs for Machine Learning.* Morgan Kaufmann Publishers, Inc., 1993.

[6] P. D. Turney. Cost sensitive classification: Emprical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, page 369 409, 2 1995.