Inducing Multivariate Decision Trees Quickly and Effectively

Qiangfu Zhao, Takaharu Kawatsure and Hirotomo Hayashi
The University of Aizu
Aizuwakamatsu, Fukushima, Japan 965-8580
Email: {qf-zhao,m5081132,m5081141}@u-aizu.ac.jp

***Abstract*— Decision tree (DT) is a popular model for machine learning. Compared with neural networks (NNs), DTs can make decisions with less computations, and they are more understandable. However, conventional axis-parallel DTs (APDTs) are not efficient for complex problems because their sizes can become very large. So far different multivariate DTs (MDTs) have been proposed to solve this problem. In our research, we have tried the neural network tree (NNTree) and the nearest neighbor classifier tree (NNC-Tree). The over all process for inducing an MDT is the same as that for inducing an APDT. The only difference is to find a multivariate test function (MTF) in each non-terminal node instead of finding a univariate one. This difference, however, makes it very hard to induce MDTs because finding the best MTF is in general NP-complete. To induce MDTs efficiently, this paper proposes a method for finding the MTF based on supervised learning. The efficiency and efficacy of the proposed method is validated through experiments with several public databases.**

I. INTRODUCTION

Decision tree (DT) is a popular model for machine learning. Compared with neural networks (NNs), DTs can make decisions with less computations. They are also more understandable because a reasoning process can be provided for each decision. In addition, DTs also provide a natural way to classify and analyze the patterns hierarchically, which is often important for many applications such as data mining and medical diagnosis.

Conventional DTs often perform a test at each non-terminal node based on one of the features. These kind of DTs are often called axis-parallel DTs (APDTs) because the decision boundary made by a univariate test is a hyperplane parallel to one of the axes. The univariate tests are simple and convenient for human users to understand. However, the partitioning ability of univariate tests is often not powerful enough, and the induced DTs can become very large. Many approaches have been proposed in the literature to improve the efficiency of DTs [1]- [10]. The most direct approach is to use a multivariate test function (MTF) in each non-terminal node.

An example of multivariate DT (MDT) is the oblique decision tree (ODT) [1], [2]. In ODTs, the MTF used in each non-terminal node is a general hyperplane which can be "oblique" instead of being parallel to some axis. We can extend the DT further by using a non-linear hypersurface instead of using a hyperplane. A natural way to realize a hypersurface is to use an NN [3]. In general, MDTs can make decisions more efficiently than APDTs because an MTF usually has more powerful partitioning ability.

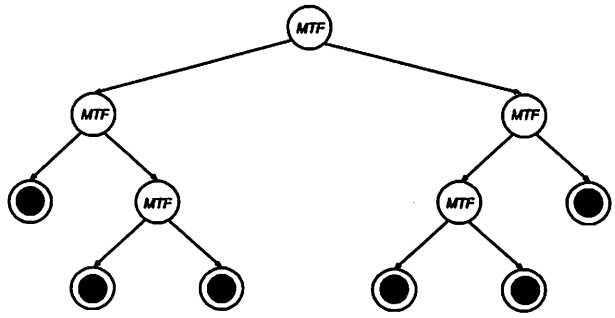

Fig. 1. Structure of a neural network tree

The over all process for inducing an MDT is the same as that for inducing an APDT. The only difference is to find an MTF in each non-terminal node instead of finding a univariate one. This difference, however, makes it very hard to induce MDTs because finding the best MTF is in general NP-complete [2]. To reduce the computational cost, some kind of *generate-and-test* algorithm is often used. For example, to find an oblique hyperplane, we can start from the best axis-parallel hyperplane, and then perform a local search. Each potential solution generated during local search is tested using some evaluation criterion (say, the information gain ratio), and the best one is used as the result. For more complex MTFs, this local search approach may not be efficient. For example, if the MTF is an NN, it is difficult to find the best solution through local search because a great number of generate-and-test operations must be performed.

So far we have studied two kinds of MDTs: neural network trees (NNTrees) [11] and nearest neighbor classifier trees (NNC-Trees) [12]. These two kinds of MDTs have the same structure (Fig. 1), but use different MTFs in the non-terminal nodes. As the MTFs, expert neural networks (ENNs) are used in NNTrees, and nearest neighbor classifiers (NNCs) are used in NNC-Trees. To find the MTFs efficiently, we have proposed to use the genetic algorithm (GA) [11]. However, experimental results tell us that the GA-based approach is too time-consuming to be practically useful.

To solve the above problem, we propose in this paper a new method for finding MTFs. The basic idea is to partition the data first according to their class labels and their neighborhood relations. The partition is then realized by an MTF. To find

the MTF, we can use the well-known back-propagation (BP) algorithm for the ENNs, or the R^4-rule for the NNCs. The R^4-rule is an algorithm proposed by one of the authors for obtaining the smallest or nearly smallest NNC [13].

This paper is organized as follows. In the next section, we provide a brief review of the DTs. In Section III, the R^4-rule is revisited because it will be used for inducing the NNC-Trees. In Section IV, the key technique for inducing MDTs quickly and effectively is proposed. Section V provides the experimental results, and Section VI is the conclusion.

II. A Brief Review of Decision Trees

A. Definition of DTs

A decision tree (DT) is a directed graph with no cycles. There is one special node called root. We usually draw a DT with the root at the top. Each node (except the root) has exactly one node above it, which is called its *parent*. The nodes directly below a node are called its *children*. A node is called a *terminal node* if it dose not have any child. A node is *non-terminal* if it has at least one child. The node of a DT can be defined as a 5-tuple as follows:

$$node = \{I, F, Y, N, L\}$$

where I is a unique number assigned to each node, F is a test function that assigns a given input pattern to one of the children, Y is a set of pointers to the children, $N = |Y|$ is the number of children or the size of Y, and L is the class label of a terminal node (it is defined only for terminal node). For terminal nodes, F is not defined and Y is empty (N=0).

The process for recognizing an unknown pattern x is as follows:

- Step 1: Set the root as the current node.
- Step 2: If the current node is a terminal node, assign x with the class label of this node, and stop; otherwise, find $i = F(x)$.
- Step 3: Set the i-th child as the current node, and return to Step 2.

B. Induction of DT

To induce a DT, it is assumed that a training set is available. Usually, the DT is induced by partitioning the training set recursively. This procedure involves three steps: 1) splitting nodes, 2) determining which nodes are terminal nodes, and 3) assigning class labels to terminal nodes.

To see if a node is a terminal node or not, the simplest way is to check if all or most examples assigned to this node belong to the same class. If all or most examples are from the same class, the node is terminal, and its label is usually defined as the class label of the majority examples.

The purpose of splitting a node is to find a good test function F for that node, so that the training examples assigned to this node can be partitioned into N groups according to the test results. Here we need a measure to quantify the "goodness" of the test function. For example, the criterion used in the well known induction algorithm C4.5 is the information gain ratio (IGR) [16]. If a test function maximizes the IGR, the average information required to classify a given example can be minimized. Many other criteria have also been proposed in the literature. However, it is known that the performance of a DT does not appear to vary significantly over a wide range of criteria [15].

C. Definition of MDTs

As shown in Fig. 1, an MDT is a DT with each non-terminal node containing an MTF (multivariate test function). In this paper, we consider two kinds of MDTs: the NNTrees and the NNC-Trees. In an NNTree, each MTF is ENN, which is a multilayer perceptrons (MLPs) in our study. Using an NNTree, an example x can be recognized as follows:

- Step 1: Set the root as the current node.
- Step 2: If the current node is a terminal node, assign x with the class label of this node, and stop; otherwise, find

$$F(x) = i = \arg \max_{1 \le k \le N} o_k \tag{1}$$

  where o_k is the k-th output of the ENN.
- Step 3: Set the i-th child as the current node, and return to Step 2.

An NNC-Tree is an MDT with each MTF being an NNC. The process for recognizing an unknown pattern x is as follows:

- Step 1: Set the root as the current node.
- Step 2: If the current node is a terminal node, assign x with the class label of this node, and stop; otherwise, find the nearest neighbor of x from the prototypes of the NNC. Suppose that p^* is the nearest neighbor of x, the value of the test function is then given by $F(x) = i = g(p^*)$, where $g(p^*)$ is the group label of p^*.
- Step 3: Set the i-th child as the current node, and return to Step 2.

D. GA-based Induction of NNTrees

To induce an NNTree, we can follow the same procedure for inducing an APDT. The only difference is to find an ENN for each non-terminal node. From the above discussion we know that the ENN should be able to partition the training examples assigned to the current node into N groups. The problem is that we do not know which example should be assigned to which group in advance. This is why we have tried to use a GA first for inducing NNTrees [11].

To use a GA, we need two definitions and three operators. The two definitions include: definition of the genotype and that of the fitness function. The three operators are: selection, crossover and mutation. In [11], we adopted a simple GA with three operators: truncation selection, one point crossover, and bit by bit mutation. By truncation selection we mean that in each generation, $r_s \times 100$ percent of the worst individuals are replaced with the off-spring of the best ones, where r_s is called the selection rate. The genotype of an ENN is defined as the concatenation of all weight vectors (including the threshold values) represented in binary numbers. The fitness of an individual ENN is defined as the information gain ratio.

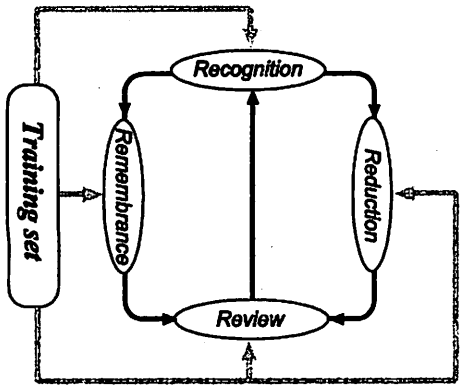

Fig. 2. The R^4-rule

Actually, GAs are also "generate-and-test" search algorithms. Experimental results have shown that GAs are usually better than conventional local search algorithms in the sense that they can find the global optimal solution with higher probability. However, the computational costs of GAs are often very large, and cannot be used easily.

III. A Review of the R^4-rule

In [13], one of the authors proposed a non-genetic evolutionary learning algorithm called the R^4-rule for designing the smallest or nearly smallest NN-MLP (nearest neighbor based multilayer perceptron). In fact, NN-MLP is just a neural network realization of the NNC. Therefore, the R^4-rule can be used to design the smallest or nearly smallest NNC directly. The R^4-rule consists of four basic operations: recognition, remembrance, reduction, and review. One learning cycle is defined as $recognition \wedge (remembrance \vee reduction) \wedge review$, where $\wedge$ and $\vee$ are logical AND and OR operations, respectively. The learning can be performed cycle after cycle until some criterion is satisfied (Fig. 2).

In the R^4-rule, each operation is performed by a process or a subroutine. Briefly speaking, *recognition* is a process to test the ability of the current NNC, and the fitness of each prototype. After *recognition*, we can know how the NNC performs and how important each prototype is. If there are too many recognition errors, some of the prototypes should be added in the process *remembrance*. On the other hand, if the recognition rate is already very high, some prototypes with very low fitness can be removed by *reduction* to make the NNC more compact and more efficient. The process *review* is necessary when some prototypes are removed or added. In *review*, the prototypes are readjusted so that the NNC can achieve better performance using recent prototypes.

Each learning cycle can be briefly described as follows. First, present all training examples to the subroutine *recognition*. If the recognition rate is higher than a desired value r_0, call subroutine *reduction* to remove an unimportant prototype; otherwise, call subroutine *remembrance* to add a new prototype. When some prototype is removed or added, call *review* to achieve higher performance. After *review*, another learning cycle starts.

The goodness or fitness of a prototype is defined as follows. At the beginning, the fitness values of all prototypes are given at random. The fitness values are updated in *recognition*. Specifically, if a prototype is a winner for a given example, its fitness value is increased by a factor δ; otherwise, if the prototype is a loser; its fitness value is decreased by δ. A prototype is the winner/loser if its current fitness value is the highest/lowest among all prototypes that can recognize the given example correctly. In the process *reduction*, we select at random one of the prototypes with fitness value less than a respecified value f_{min}, and remove it from the NNC.

To use the R^4-rule, we need to specify at least three parameters: r_0, δ and f_{min}. To select a set of good parameters, however, is not easy if we do not know the domain knowledge well. We can modify the R^4-rule can be modified as follows. First, the desired recognition rate r_0 can be obtained automatically as follows:

- Initialize the NNC with sufficiently many prototypes, which are given at random.
- Train the NNC using some existing learning vector quantization (LVQ) algorithm.
- Use the recognition rate of the NNC after training as r_0.

To delete the parameter δ, we can change the way to evaluate the fitness of the prototypes. In each learning cycle, the fitness values of the prototypes are first reset to zeros. For an given example x, if it can be classified correctly only by the prototype p, the fitness of p is increased by 1. On the other hand, if x is mis-classified only if p exists, the fitness of p is decreased by 1. The parameter f_{min} can also be deleted if we remove a prototype whenever its fitness is the lowest.

IV. Inducing MDTs Quickly and Effectively

As stated earlier, to find the best MTF in each non-terminal node of an MDT is a very hard problem. So far researchers have tried different "generate-and-test" approaches. These approaches are usually very time consuming, and cannot be used easily. To solve this problem, we propose a different approach here. First, we define the teacher signals (group labels) for all examples assigned to the current node. Based on the teacher signals, we can obtain a good MTF through supervised learning.

In general, we do not know the best way to partition the training examples assigned to a non-terminal node. However, we can get a relatively good one as follows. Suppose that we want to partition S (which is the set of examples assigned to the current node by the tree) into N sub-sets $S_1, S_2, \cdots, S_N$, which are initially empty sets. For any given example $x \in S$,

1) if there is a $y \in S_i$, such that $label(y) = label(x)$, assign x to S_i;
2) else, if there is a S_i, such that $S_i = \Phi$, assign x to S_i;
3) else, find y, which is the nearest neighbor of x in $\cup S_i$, and assign x to the same sub-set as y.

where $\cup$ represents the union of sets, and Φ is the empty set. When x is assigned to S_i, the teacher signal of x is defined by

TABLE I

PARAMETERS OF THE DATABASES

Name	Number of examples	Number of features	Number of classes
car	1728	6	4
crx	690	15	2
dermatology	366	34	6
ecoli	336	7	8
housevotes84	435	17	2
ionosphere	351	34	2
iris	150	4	3
optdigits	5620	64	10
pendigits	10992	16	10
tic-tac-toe	958	9	2

$g(x) = i$. We call $g(x)$ the group label of x. Once the group labels are defined, we can obtain an MTF through supervised learning. If we realize the MTF by an ENN, we can use the BP algorithm. If we realize the MTF by an NNC, we can use the R^4-rule.

V. EXPERIMENTAL RESULTS

To verify the effectiveness of the method proposed here, we conducted experiments with databases taken from the machine learning repository of the University of California at Irvine. The databases used include: Car Evaluation Database (car), Credit Approval Database (crx), Dermatology Database (dermatology), Protein Localization Sites Database (ecoli), 1984 United States Congressional Voting Records Database (housevotes84), Johns Hopkins University Ionosphere Database (ionosphere), Iris Plants Database (iris), Optical Recognition of Handwritten Digits Database(optdigits), Pen-Based Recognition of Handwritten Digits Database (pendigits), and Tic-Tac-Toe Endgame Database (tic-tac-toe). Table I shows the parameters of the databases.

In all experiments, 2/3 of the data were used for training, and 1/3 of the data were used for testing. To increase the reliability, 30 runs were conducted for each databases. The database was shuffled before each run. The computer used in the experiments is Sun Blade 150 with 1 CPU, UltraSPARC-IIe 550MHz.

For comparison, the following results are used:

- APDTs obtained by C5 (a commercial version of C4.5).
- NNTrees obtained by GA-based approach.
- NNTrees obtained by BP-based approach.
- NNC-Trees obtained by R^4-rule.

Note that in the BP-based induction of NNTrees and the R^4-rule based induction of NNC-Trees, the technique proposed in the last section is used.

Parameters used in the GA-based NNTree induction are: 1) the number of generations is 1,000, 2) the population size is 200, 3) the number of bits per weight is 16, 4) the selection rate is 0.2 (truncation selection), 5) the crossover rate is 0.7 (one point crossover), and 6) the mutation rate is 0.01 (bit by bit mutation).

Parameters related to the ENN are: 1) the number of inputs is the number of features of the given problem, 2) the number of output neurons (or the number of branches for each non-terminal node) is 2 (e.g., we consider only binary trees here), and 3) the number of hidden neurons is 4.

Parameters used in BP are : 1) the learning rate is 0.5, and 2) the number of epochs is 2,000.

Parameters used in the R^4-rule based NNC-Tree induction are: 1) the method for review (and also for training the initial NNC) is DSM [22], 2) the learning rate for DSM is 0.2, 3) the maximum number of prototypes per NNC is 16, 4) the initial size of each NNC is 10, 5) the number of learning cycles is 32, 6) the number of epochs for review is 20, 7) the number of epochs for training the initial NNC is 200, and 8) the number of branches for each non-terminal node is 2.

Table II - V show the experimental results. In the tables, "Number of non-terminal nodes" is used to measure the size of a tree, "E(%)" is the error rate for the test set, and "Time in seconds" is the computing time used for one run. All results are the average of 30 runs. From these tables, we may draw the followings conclusions:

1) **About the accuracy**: In 10 cases, the NNTrees obtained using the BP-based approach win 4 times, the NNC-Trees obtained using the R^4-rule based approach win 3 times. For large databases such as optdigits and pendigits, the MDTs usually outperform the APDTs obtained by C5.
2) **About the size**: In all cases the NNC-Trees the NNTrees obtained using the proposed technique are much smaller than the APDTs. In most cases the NNTrees obtained by the GA-based approach are also smaller, but for some databases they are even larger. If we increase the number of generations and the population size, GA may also get better results, but this will increase the computing time further.
3) **About computing time**: Needless to say, C5 is the fastest method to obtain a DT. If we compare the other three methods, the BP-based approach can induce MDTs very quickly. As stated above, the BP-based approach can also produce smaller and more accurate NNTrees.
4) **About the comprehensibility**: Usually, the APDTs obtained by C5 are most comprehensible. However, for large databases, the APDTs are very large, and cannot be understood easily. The NNC-Trees are actually more comprehensible because they can be transformed into a set of *if-then* rules, and the decisions are made based on the similarity between the prototypes and the input patterns. For example, in the case of optdigits, we can often get the minimum MDT which can be transformed into a small set of comprehensible if-then rules.

VI. CONCLUSION

In this paper, we have proposed a new method for inducing MDTs (multivariate decision trees). Our purpose is of two-fold. One is to reduce the computation time (to induce the MDTs more quickly), and another is to reduce the tree size (to induce the tree more effectively). Experimental results with 10 public databases show that the BP-based approach is the best

in the sense that it can result in smaller NNTrees quickly. The generalization ability of the NNTrees so obtained (measured by their recognition rates for the test sets) is also better than or comparable with the DTs obtained by other approaches. If we want to induce comprehensible MDTs, the R^4-rule based approach is better because it can produce smaller and more understandable MDTs

ACKNOWLEDGMENTS

This research is supported in part by the Grants-in-Aid for Scientific Research of Japan Society for the Promotion of Science (JSPS), No. 17500148.

REFERENCES

[1] E. Cantú-Paz and C. Kamath, "Inducing oblique decision trees with evolutionary algorithms," IEEE Trans. on Evolutionary Computation, Vol. 7, No. 1, pp. 54-68, 2003.

[2] S. K. Murthy, S. Kasif and S. Salzberg, "A system for induction of oblique decision trees," Journal of Artificial Intelligence Research, Vol. 2, No. 1, pp. 1-32, 1994.

[3] H. Guo and S. B. Gelfand, "Classification trees with neural network feature extraction," IEEE Trans. on Neural Networks, Vol. 3, No. 6, pp. 923-933, Nov. 1992.

[4] C. Z. Janickow, "Fuzzy decision trees: issues and methods," IEEE Trans. Systems, Man, and Cybernetics B, Vol. 28, No. 1, pp. 1-14, 1998.

[5] M. Golea and M. Marchand, "A growth algorithm for neural network decision trees," Europhysics Letters, Vol.12, pp. 105-110, 1990.

[6] J. Basak, "Online adaptive decision trees," Neural Computation, Vol. 16, pp. 1959-1981, 2004.

[7] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and EM algorithm," Neural Computation, Vol. 6, pp. 181-214, 1994.

[8] R. G. Adams, K. Butchart and N. Davey, "Hierarchical classification with a competitive evolutionary neural tree," Neural Networks, Vol. 12, pp. 541-551, 1999.

[9] T. Li, L. Y. Fang and K. Q-Q. Li, "Hierarchical classification and vector quantization with neural tree," Neurocomputing, Vol. 5, pp. 119-139, 1993.

[10] J. Basak and R. Krishnapuram, "Interpretable hierarchical clustering by constructing an unsupervised decision tree," IEEE Trans. on Knowledge and Data Engineering, Vol. 7, No. 1, pp. 121-132, 2005.

[11] Q. F. Zhao, "Evolutionary design of neural network tree - integration of decision tree, neural network and GA," Proc. IEEE Congress on Evolutionary Computation, pp. 240-244, Seoul, 2001.

[12] T. Kawatsure and Q. F. Zhao, "Inducing multivariate decision trees with the R4-rule," Proc. IEEE International Conference on Systems, Man and Cybernetics (SMC'05), pp. 3593-3598, , Hawaii, 2005.

[13] Q. F. Zhao and T. Higuchi, "Evolutionary learning of nearest neighbor MLP," IEEE Trans. on Neural Networks, Vol.7, No. 3, pp. 762-767, 1996.

[14] Q. F. Zhao, "Stable on-line evolutionary learning of NN-MLP," IEEE Trans. on Neural Networks, Vol. 8, No. 6, pp. 1371-1378, 1997.

[15] L. Brieman, J. H. Friedman, R. A. Olshen and C. J. Stong, *Classification and regression trees*, Belmont, CA: Wadsworth, 1984.

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

[17] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," IEEE Trans. on Information Theory, Vol. IT-13, No. 1, pp. 21-27, Jan. 1967.

[18] T. Kohonen, "Self-organized formation of topologically correct feature maps," Biolog. Cybern.,Vol. 43, pp. 59-69, 1982.

[19] T. Kohonen, "The self-organizing map," Proc. IEEE, Vol. 78, No. 9, pp. 1464-1480, Sept. 1990.

[20] G. A. Carpenter and S. Grossberg, "ART 2: self-organization of stable category recognition codes for analog input patterns," Applied Optics, Vol. 26, No. 23, pp. 4919-4930, Dec. 1987.

[21] G. A. Carpenter and S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network," IEEE Computer, Vol. 21, No. 3, pp. 77-88, Mar. 1988.

[22] S. Geva and J. Sitte, "Adaptive nearest neighbor pattern classification," IEEE Trans. on Neural Networks, Vol. 2, No.2, pp. 318-322, Mar. 1991.

TABLE II

RESULTS OF GA-BASED NNTREE INDUCTION

Database	Number of neurons per ENN	Number of non-terminal nodes	E(%)	Time in seconds
car	4.00	17.97	7.54	2575.98
crx	4.00	17.37	21.75	1398.36
dermatology	4.00	5.27	8.72	77.82
Ecoli	4.00	19.23	23.30	349.00
housevotes84	4.00	6.00	15.43	578.86
ionosphere	4.00	3.93	**9.62**	463.82
iris	4.00	2.63	5.07	16.72
optdigits	4.00	44.24	5.51	13982.68
pendigits	4.00	55.70	2.41	11399.34
tic-tac-toe	4.00	53.07	30.95	3171.70

TABLE III

RESULTS OF BP-BASED NNTREE INDUCTION

Database	Number of neurons per ENN	Number of non-terminal nodes	E(%)	Time in seconds
car	4.00	8.3	3.46	10.58
crx	4.00	5.7	18.22	25.33
dermatology	4.00	6.44	**3.20**	0.22
Ecoli	4.00	8.14	**17.29**	6.75
housevotes84	4.00	1.96	5.56	1.32
ionosphere	4.00	2.57	10.37	2.89
iris	4.00	2.24	**3.80**	0.88
optdigits	4.00	13.94	4.44	8.05
pendigits	4.00	16.47	2.87	66.39
tic-tac-toe	4.00	2.2	**1.83**	0.94

TABLE IV

RESULTS OF R^4-RULE BASED NNC-TREE INDUCTION

Database	Number of prototypes per NNC	Number of non-terminal nodes	E(%)	Time in seconds
car	4.30	4.07	**1.78**	15.06
crx	6.95	5.00	20.99	16.23
dermatology	2.01	5.00	4.56	7.27
Ecoli	4.09	8.93	23.42	5.02
housevotes84	5.44	1.07	15.08	4.18
ionosphere	4.59	1.07	11.51	4.68
iris	2.30	2.03	4.20	0.47
optdigits	3.13	9.00	**3.32**	452.70
pendigits	3.90	17.77	**1.81**	286.95
tic-tac-toe	6.68	4.70	17.17	31.40

TABLE V

RESULTS OF C5 (ALL PARAMETERS TAKE DEFAULT VALUES)

Database	Number of Non-terminal nodes	E(%)	Time in seconds
car	42.73	3.60	0.32
crx	16.93	**13.19**	0.04
dermatology	8.13	4.59	0.10
Ecoli	10.91	18.55	0.10
housevotes84	4.30	**4.32**	0.10
ionosphere	10.60	10.77	0.12
iris	3.75	7.07	0.10
optdigits	156.57	10.36	1.54
pendigits	156.00	3.95	1.22
tic-tac-toe	46.67	10.69	0.17