Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Induction of Compact Neural Network Trees through Centroid Based Dimensionality Reduction

Hirotomo Hayashi and Qiangfu Zhao
Department of Computer and Information Systems
The University of Aizu
Aizuwakamatsu, Japan
{d8092103, qf-zhao}@u-aizu.ac.jp

***Abstract*—Neural network tree (NNTree) is a hybrid model for machine learning. Compared with single model fully connected neural networks, NNTrees are more suitable for structural learning, and faster for decision making. To increase the realizability of the NNTrees, we have tried to induce more compact NNTrees through dimensionality reduction. So far, we have used principal component analysis (PCA) and linear discriminant analysis (LDA) for dimensionality reduction, and confirmed that in most cases the LDA based approach can result in very compact NNTrees without degrading the performance. One drawback in using the LDA based approach is that the cost for finding the transformation matrix can be very high for large databases. To solve this problem, in this paper we investigate the efficiency and efficacy of two centroid based approaches for NNTree induction. One is to map each datum directly to the class centroids; and the other is to find the least square error approximation of each datum using the centroids. Experimental results show that both approaches, although simple, are comparable to the LDA based approach in most cases.**

***Index Terms*—Machine learning, pattern recognition, decision tree, neural network, multivariate decision tree.**

I. Introduction

Neural networks (NNs) are a class of learning models analogous to the human brain. They often get good answers for solving non-linear problems, and have been applied successfully to many fields, such as image recognition, speech recognition, data mining, robot control, and so on. One drawback in using NNs is that determination of a proper network structure is usually difficult. Although several approaches have been proposed in the literature [1]-[4], these methods are often time consuming because the network structure has to be revised little by little during learning.

In our research, we have proposed a hybrid learning model called neural network tree (NNTree) [5]. An NNTree is a special decision tree (DT) with a small NN embedded in each internal node (see Fig. 1). The small NNs are used for local decisions, and the tree controls the whole decision making process. Usually, an NNTree is induced recursively. For the current node (start from the root), a small NN is added if the data assigned to this node are not pure enough (measured by the information gain ratio in this study). The small NN divides all data assigned to the current node into several groups. For each group, we do the same thing as above recursively. Because there is no trial and error, and the number of small NNs needed is often proportional to the number of classes, the structure of the NNTree can be determined efficiently and automatically.

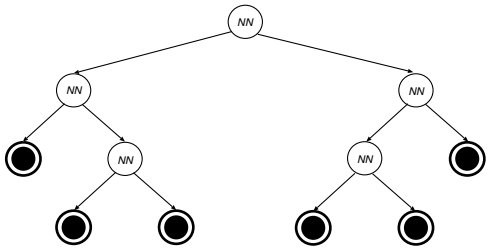

Fig. 1. An example of neural network trees

Another advantage in using the NNTrees is that an NNTree is usually faster than a single model fully connected NN (SMFC-NN). Suppose that the SMFC-NN under consideration is a 3-layer feedforward NN. If there are all together N_n neurons, and N_p processing elements (PEs) are available to implement the NN (for software implementation, we usually suppose $N_p = 1$), then the computing time for making a decision is proportional to N_n/N_p. For example, if $N_n = 100$ and $N_p = 4$, the computing time is about 25 time units, where one time unit is defined as the time used for calculating the output of one neuron. On the other hand, if we use NNTree of the same number of neurons, the computing time is proportional to $\lceil n_n/N_p \rceil \times P_l$, where n_n is the number of neurons used by the small NN embedded in each internal node, and P_l is the average path length for making a decision. It is worthwhile to note that P_l is proportional to $log_2[N_n/n_n]$. Again, for $N_n = 100$ and $N_p = 4$, if we suppose $n_n = 5$ (one output neuron plus 4 hidden neurons), then the tree contains 20 internal nodes, and P_l is less than 5. The average computing time for making a decision will be less than $2 \times 5 = 10$ time units in average. Thus, NNTree in general can make decisions faster than SMFC-NN if they are implemented using the same number of PEs.

One bottleneck in using the NNTrees is the possible high cost for induction. Although the recursive induction process can finish in a very limited number of steps, finding the best NN in each internal node can be very time consuming. In fact, finding the best multivariate test function in each internal node is an NP-complete problem [6]. To solve the problem

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

more efficiently, we have proposed several algorithms [7], [8]. Among them, the algorithm given in [8] is the most efficient one. This algorithm is based on a heuristic grouping strategy, and the NNTrees can be induced very quickly even for relatively large databases. In a certain sense, we have solved the hardest problem in using the NNTrees.

Since the final goal of this research is to use the NNTree as a core module in different portable systems (e.g. cell phone, digital camera, and IC-card), we must try to induce NNTrees that are compact enough to be realized in a VLSI chip. The problem is that, even if we can induce an NNTree efficiently, we may not be able to realize the NNTree easily in hardware because the size or the number of inputs of each neuron can be very large for many applications. An NNTree containing large neurons usually requires more memory space and longer time for making decisions.

To increase the realizability of the NNTrees, we have tried to induce more compact NNTrees through dimensionality reduction. So far, we have used principal component analysis (PCA) [9] and linear discriminant analysis (LDA) [10] for dimensionality reduction, and confirmed that in most cases the LDA based approach can result in compact NNTrees without degrading the performance of the tree [11]. However, the time complexity of the LDA based dimensionality reduction can be very high if the number of data and the number of features are large.

In this paper we study the possibility of using low-cost approaches for dimensionality reduction. Specifically, we investigate two centroid based approaches here. The first one is to map each datum directly to the centroids, and the second is to find the least square error (LSE) approximation of each datum using the centroids. In both approaches the data are mapped to a N_c dimensional space, where N_c is the number of classes of a given pattern recognition problem. Therefore, for any test datum, the time for dimensionality reduction is almost the same as that for LDA. The time complexity for finding the transformation matrix, however, is much smaller. Experimental results on several public databases show that both approaches, although simple, can get relatively good results.

The rest of paper is organized as follows. Section 2 gives a brief review of DTs and NNTrees. Section 3 explains how to apply dimensionality reduction approaches to inducing compact NNTrees. Section 4 provides experimental results, and discusses the performance of the proposed method compared with existing method of inducing NNTrees. Section 5 is the conclusion.

II. Preliminaries

A. Definition of Decision Tree

Roughly speaking, a decision tree (DT) is a directed graph with no cycles. We usually draw a DT with the root at the top (see Fig. 1). 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 does not have any child. A node is called an internal node 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 $\boldsymbol{x}$ is as follows:

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

B. Induction of Decision Tree

To induce a DT, it is necessary to have a training set composing feature vectors and their class labels. The DT is induced by partitioning the feature space recursively. The induction process include three steps: splitting the nodes, finding terminal nodes, and assigning class labels to terminal nodes. The step of splitting nodes is the most significant and time consuming. To get a good DT, we should try to find a good test function for each internal node. Many criteria have been proposed for estimating the "goodness" of a test function [12],[13]. It is known that the efficiency of DTs is not affected greatly over a wide range of criteria [12]. In our study, we just adopt the information gain ratio (IGR), which is used in the well-known DT induction program C4.5 [13].

A test function $F(\boldsymbol{x})$ which maximizes the IGR decreases the entropy most for recognizing an unknown datum. Suppose S ($|S|$ is the size of S) is the set of data assigned to the current node, and n_i is the number of data belonging to the i-th class ($i = 1, 2, \ldots, N_c$, and N_c is the number of classes), the entropy for recognizing an unknown datum is given by

$$info(S) = -\sum_{i=1}^{N_c} \frac{n_i}{|S|} \times log_2(\frac{n_i}{|S|}) \tag{1}$$

Suppose S is divided into N subsets $S_1, S_2, \ldots, S_N$ by the test function F, the information gain is given as follows:

$$gain(F) = info(S) - info_F(S) \tag{2}$$

where

$$info_F(S) = \sum_{i=1}^{N} \frac{|S_i|}{|S|} \times info(S_i) \tag{3}$$

The IGR is defined by

$$gain\ ratio(F) = gain(F) \div split\ info(F) \tag{4}$$

where

$$split\ info(F) = -\sum_{i=1}^{N} \frac{|S_i|}{|S|} \times log_2(\frac{|S_i|}{|S|}) \quad (5)$$

C. Definition of NNTrees

As mentioned previously, an NNTree is a kind of multivariate decision tree, and each internal node contains a small NN. As the small NN, we use a small multilayer perceptron (MLP) here, although any kind of NNs can be adopted. The number of inputs and outputs of the NN correspond, respectively, to the dimensionality N_d of the feature space and the number N of child nodes. A major point in this research is to solve complex problems by embedding small NNs to the DT. Therefore, a small number (2, 4, or 6) is used as the number of hidden neurons as N_h in this paper.

Using an NNTree, any given example $\boldsymbol{x}$ is recognized as follows:

- Step 1: Start from the root. This is the current node.
- Step 2: If the current node is already a terminal node, assign its label to $\boldsymbol{x}$. Else find the next child node by using the following equation.

$$i = F(\boldsymbol{x}) = \arg \max_{1 \le k \le N} o_k \quad (6)$$

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

D. Induction of NNTrees

The overall process for inducing NNTrees is the same as that of C4.5. The difference is the method to generate the test function $F(\boldsymbol{x})$ in each internal node. For inducing NNTrees, it is not an efficient way to find the test function based on "generation and evaluation". To find the test function more efficiently, we can define the teacher signals first using some heuristics [14]. Based on the teacher signals, we can find the NN test function quickly using the well-known back-propagation (BP) algorithm. The algorithm for inducing an NNTrees can be explained as follows.

- Step 1: Check if the current node is a terminal node. If examples assigned to this node belong to several classes, this node is an internal node, goto Step 3; otherwise, this node is a terminal node, goto Step 2.
- Step 2: Assign class label of the biggest classes to this node, and finish.
- Step 3: Divide the examples assigned to the current node to N groups, and define the teacher signals for all data.
- Step 4: Train the NN using the BP algorithm with the teacher signals defined in Step 3.
- Step 5: Split the examples into N groups using NN obtained at Step 4, and create a new node for each group.
- Step 6: For each new node (child node), repeat the above process recursively.

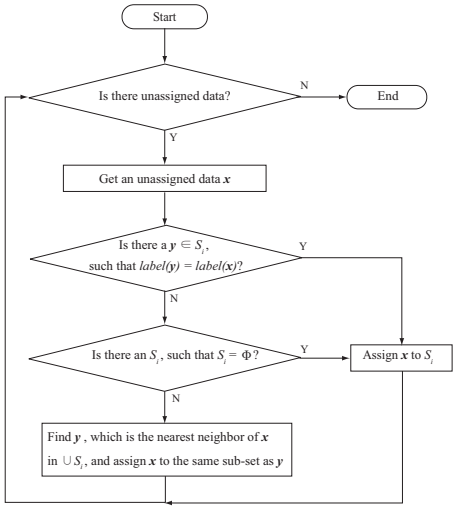

Fig. 2. Flowchart of the proposed method for assigning group labels

---

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 Φ. For any given example $\boldsymbol{x} \in S$, repeat following process:

- Step 1: Get an unassigned data $\boldsymbol{x}$ from S.
- Step 2: If there is a $\boldsymbol{y} \in S_i$, such that $label(\boldsymbol{y}) = label(\boldsymbol{x})$, assign $\boldsymbol{x}$ to S_i;
- Step 3: Else, if there is a S_i, such that $S_i = \Phi$, assign $\boldsymbol{x}$ to S_i;
- Step 4: Else, find $\boldsymbol{y}$, which is the nearest neighbor of $\boldsymbol{x}$ in $\cup S_i$, and assign $\boldsymbol{x}$ to the same sub-set as $\boldsymbol{y}$.

where $\cup$ represents the union of sets, and Φ is the empty set.

The flowchart of the above algorithm is given in Fig. 2. The teacher signal of a data $\boldsymbol{x}$ is the subset number to which it is assigned. That is, if $\boldsymbol{x}$ is assigned to S_i, its teacher signal is i.

E. Conceptual Model of NNTree for Hardware Implementation

As stated earlier, the goal of this research is to propose a method for inducing NNTrees that are compact enough for VLSI implementation. Since all NNs in an NNTree have the same structure, we can design a hardware module for realizing all NNs, and another module for controlling the decision process. We call these two modules the NN processor and the tree controller, respectively (see Fig. 3). For any given input pattern x, the decision process can be described roughly as follows:

- Step 1: Reset the pointer p of the tree controller.
- Step 2: Based on p, load the node label of the current node from a global memory to the local memory of the tree controller. If the node label is meaningful, output the label, and stop; otherwise, continue.

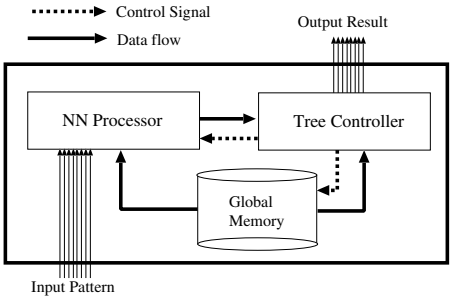

Fig. 3. A conceptual hardware structure of the NNTree

- Step 3: Based on the same p as above, load all other information about the current node from the global memory. The weights are sent to the local memory of the NN processor, and the pointers of the child nodes are sent to the local memory of the controller.
- Step 4: Find the output of the NN processor based on x.
- Step 5: Based on the output of the NN processor, update the pointer p of the controller, and go to Step 2.

So far, we have successfully proposed a method for inducing the NNTrees efficiently. However, the NNTrees so obtained are still not compact enough. In fact, when the number of inputs for each neuron is large, a great number of weights are required for each NN. This is absolutely not good for implementing an NNTree on a chip. The main target of this paper is to reduce the number of inputs of the neurons. After reduction, we can make decisions more quickly, using a much smaller NN processor (as well as a smaller global memory).

III. Dimensionality Reduction

A. General Considerations

Model reduction of NNTree consists of 2 steps. The first step is to reduce the number of features of each datum using dimensionality reduction, and the second is to induce the NNTree using the compressed data. These two steps are independent of each other, and therefore we can employ any dimensionality reduction methods in the first step.

So far, we have applied LDA for dimensionality reduction [11]. It is known that the time complexity for finding the transformation matrix of the LDA is $O(N_d^2 \times N_t)$, where N_d is the dimensionality of the feature space, and N_t is the number of data in the training set. Here, we assume $N_t > N_d$. When N_t and N_d are large, the cost for dimensionality reduction can be very high. To reduce the cost for dimensionality reduction, we study two centroid based approaches. The question is, can we keep the performance of the NNTrees if we use such kind of low-cost approaches? We will answer this question in the next section through experiments.

B. Direct Dimensionality Reduction Based on Centroids

Now, let us consider dimensionality reduction based on class centroids. Suppose there are N_c classes. The first step is to find the class centroids as follows:

$$\mu_i = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}, i = 1, 2, \cdots, N_c \tag{7}$$

where x_{ij} is the jth training example of the ith class, and n_i is the number of training examples in the ith class. The second step is to compress the datum $x \in R^{N_d}$ to $y \in R^{N_c}$ by finding the distance between x and all centroids. That is, for any training example x,

$$y = [distance(x, \mu_1), \cdots, distance(x, \mu_{N_c})]^T. \tag{8}$$

Theoretically, any distance measure can be used here. In this paper, we suppose that all examples are normalized so that the Euclidean norm of each vector is 1. In this case, inner product will be enough to measure the closeness between vectors, and for any training example x, its projection y can be found as follows:

$$y = [< \mu_1, x >, \cdots, < \mu_{N_c}, x >]^T = C^T x \tag{9}$$

where $C = [\mu_1, \mu_2, \cdots, \mu_{N_c}]$.

TABLE I
Parameters of the Databases

	Number of instances	Number of features	Number of classes
adult	48842	14	2
arrhythmia	452	279	13
crx	690	15	2
dermatology	366	34	6
diabetes	768	8	2
Internet advertisements	3279	1558	2
isolet	7797	617	26
soybean	307	35	19
thyroid	215	5	3
wine	178	13	3

C. Dimensionality Reduction through Centroid Based Approximation

Note that the projection y obtained in the above direct method is not optimal in any sense. Compared with PCA or LDA, some important information for reconstructing or discriminating the data must be lost after dimensionality reduction. To get a better solution, we can find the best approximation of each datum using the centroids. That is, for any x, we should find y, such that

$$||x - Cy|| = ||x - \sum_{k=1}^{N_c} y_k \mu_k|| \tag{10}$$

is minimized. We can find the LSE solution as follows:

$$y = [C^T C]^{-1} C^T x = C^+ x \tag{11}$$

where C^+ is the pseudo-inverse matrix of C. To obtain the pseudo-inverse matrix, we can use the singular value decomposition (SVD). The SVD of an arbitrary matrix A is defined by

$$A = U \Sigma V^T \tag{12}$$

The pseudo-inverse matrix can be obtained simply as follow:

$$A^+ = V\Sigma^+ U^T \tag{13}$$

where Σ^+ is a matrix with every nonzero entry replaced by its reciprocal. One advantage of using the SVD is that we can get the solution even if the matrix C^TC is singular. Of course, if the non-singularity of C^TC is guaranteed, we can also find the pseudo-inverse using $[C^TC]^{-1}C^T$. The computational complexity of this method will be roughly the same as that of the SVD.

IV. Experimental Results

To verify the efficiency and efficacy of the two centroid based approaches, we conducted experiments with databases taken from the machine learning repository of the University of California at Irvine. The databases used are adult, arrhythmia, crx, dermatology, Internet advertisements, isolet, soybean, thyroid, wine. Table I shows the parameters of the databases.

For each database, we conducted 5 trials of 10-fold cross validation (all together 50 runs). Each database was shuffled before each trial. The computer used in the experiments is Sun Java Workstation W1100z (the CPU is 1.8 GHz AMD Optron 144, and the main memory is 1,024 MB).

We compare four approaches. Method-I is the original induction without dimensionality reduction. In this approach, NNTrees are induced by using the original data. In Method-II, Method-III, and Method-IV, different dimensionality reduction methods are applied to induce NNTrees. LDA is used in Method-II, the direct centroid (DC) approaches is used in Method-III, and the centroid based approximation (CBA) approach is used in Method IV.

We set experimental parameters of NNTrees as follows. First, for the BP algorithm, the learning rate is fixed to 0.5, and the maximum number of epochs for learning is 1,000. For the small NN in each internal node, the number of inputs of the NN is N_d for Method-I, $N_c - 1$ for Method-II, and N_c for Method-III and Method-IV; the number of hidden neurons is fixed to 4; and the number of output neuron is 1 because we consider only binary NNTrees here.

Table II shows the experimental results for all databases. In this table, the error rate for the test set, the number of nodes of the NNTree (including both terminal nodes and internal nodes), the total number of weights of all NNs, the total training time, the time used for dimensionality reduction, and the test time are provided. For each result, we have the average over 50 runs and the 95% confidence interval.

A. Discussion about the Error Rate

Method-I wins 3 times, Method-II 3 times, Method-III 3 times, and Method-IV 1 time. We have expected that the performance of Method-I should be the best because there is no information loss caused by dimensionality reduction. However, for 7 out of 10 cases dimensionality reduction produces better results. That is, for most cases, if we select a proper dimensionality reduction method, the recognition rate can be improved.

If we compare Method-II with Method-III, Method-II win 6 times, and Method-III win 4 times. It seems that Method-III, although simple, is quite good. The same thing can be said for Method-IV. If we compare Method-III and Method-IV, Method-III win 6 times, and Method-IV win 4 times. Although Method-III is much simpler, it was better than Method-IV.

B. Discussion about the total number of nodes

Method-I wins 2 times, Method-II wins 5 times, Method-III wins 4 times. Except for isolet, we can obtain NNTrees with less nodes through dimensionality reduction. Especially, in 3 cases (adult, crx, and diabetes), the NNTrees become much smaller after dimensionality reduction. It seems that in most cases dimensionality reduction can actually extract better features for making local decisions in each internal node, and thus less nodes are needed for the whole tree.

C. Discussion about the total number of weights

Method-II wins 8 times, and Method-III wins 2 times. That is, in all cases, the NNTrees obtained after dimensionality reduction are more compact. Especially, if we use LDA for dimensionality reduction, the total number of weights is often orderly smaller. Thus, the NNTrees obtained after LDA based dimensionality reduction are much easier for hardware implementation. There is no orderly difference between Method-II and Method-III or Method-IV.

D. Discussion about the computing time for training

As expected, Method-III is the fastest method for dimensionality reduction. However, when we look at the total training time which includes the time for induction of the NNTree and the time for dimensionality reduction, Method-II wins 6 times out of 10. The most possible reason is that LDA can provide the best projection for recognition after dimensionality reduction. Thus, even if Method-III and Method-IV can provide the transformation matrix faster, the extracted feature vectors are more difficult to recognize, and as the whole, Method-II is more efficient and more effective.

However, considering that the centroids are much simpler to be updated when new data are available, the DC and CBA methods might be more effective for on-line learning. This is one of the topics for future study.

V. Conclusion

In this paper, we have investigated the efficiency and efficacy of two centroid based dimensionality reduction approaches for inducing compact NNTrees. Experimental results show that these two approaches, although very simple, can get comparable results as long as the recognition rates are considered. However, if we consider the total time for induction, the LDA based approach is still the best.

For future study, we would like to increase the number of representatives in each class, instead of using only one centroid. We would like to investigate if such kind of simple piece-wise linear approach can get better results than the LDA. Also, we would like to investigate the efficiency and efficacy of the centroid/representative based approach for on-line learning.

TABLE II
THE EXPERIMENTAL RESULTS

	Method	Test error	Total node sizes	Total number of weights	Training time		Test time
					Total	Dimensionality reduction	
adult	Original	**14.88±1.54**	108.0±13.2	3210.0±395.1	325.34±29.69	-	0.014797±0.005085
	LDA	17.21±1.65	**4.0±0.4**	**12.2±1.6**	**82.58±4.17**	0.045203±0.008597	**0.005364±0.002916**
	DC	18.83±1.73	12.6±1.6	69.6±9.5	187.14±23.35	**0.023605±0.006140**	0.013423±0.004903
	CBA	19.20±1.74	4.2±0.4	19.0±2.5	91.19±4.65	0.024099±0.006201	0.006677±0.003254
arrhythmia	Original	36.80±3.32	30.2±3.3	16329.6±1844.8	11.00±1.15	-	0.000987±0.001249
	LDA	51.38±3.65	21.5±0.8	532.5±19.2	**1.50±0.09**	0.969424±0.040072	**0.000612±0.000980**
	DC	36.44±2.98	**17.8±1.9**	**469.3±53.5**	2.19±0.21	**0.005243±0.002868**	0.000668±0.001024
	CBA	**36.40±2.89**	30.2±1.8	816.5±48.7	2.95±0.18	0.105821±0.012965	0.000795±0.001116
crx	Original	16.26±2.05	13.5±0.9	399.4±28.6	3.34±0.13	-	0.000145±0.000476
	LDA	14.46±1.84	**3.6±0.4**	**10.6±1.7**	**0.93±0.06**	0.001278±0.001418	**0.000077±0.000348**
	DC	**14.00±1.97**	6.0±0.9	30.0±5.1	1.27±0.09	**0.000327±0.000716**	0.000110±0.000416
	CBA	14.12±1.97	4.7±0.6	22.3±3.6	1.14±0.08	0.000450±0.000843	0.000102±0.000401
dermatology	Original	**3.83±1.18**	14.4±0.6	935.2±38.0	**0.08±0.02**	-	0.000107±0.000409
	LDA	4.22±1.29	**12.6±0.5**	**139.7±5.8**	0.24±0.03	0.004646±0.002700	**0.000104±0.000404**
	DC	5.44±1.38	13.4±0.5	173.0±6.6	0.26±0.02	**0.000391±0.000783**	0.000112±0.000419
	CBA	4.06±1.26	13.4±0.5	173.0±7.0	0.31±0.04	0.000950±0.001282	0.000115±0.000424
diabetes	Original	25.95±2.41	23.9±2.6	411.8±47.2	3.07±0.18	-	0.000162±0.000504
	LDA	**23.16±2.39**	**4.8±0.7**	**15.4±2.7**	**1.17±0.09**	0.000657±0.001017	**0.000091±0.000377**
	DC	27.21±2.50	6.4±1.1	32.6±6.4	1.46±0.12	**0.000282±0.000665**	0.000116±0.000426
	CBA	27.63±2.60	6.0±0.8	30.0±4.6	1.42±0.11	0.000383±0.000778	0.000115±0.000424
Internet advertisements	Original	**2.65±0.70**	6.1±0.4	15964.2±1166.6	29.01±5.00	-	0.008288±0.003689
	LDA	3.07±0.72	3.9±0.4	**11.7±1.5**	171.07±0.73	167.727251±0.706014	**0.005761±0.003006**
	DC	4.38±0.87	**3.0±0.1**	12.0±0.1	**4.67±0.09**	**0.088666±0.011856**	0.010211±0.004001
	CBA	4.10±0.85	3.7±0.3	16.1±1.7	29.41±0.34	24.358377±0.244255	0.019719±0.005675
isolet	Original	11.40±3.02	**73.9±3.1**	90129.1±3864.3	304.10±23.06	-	0.035187±0.007528
	LDA	**8.59±1.20**	95.4±2.8	4910.9±141.9	**32.07±1.05**	17.214797±0.200151	**0.040282±0.008060**
	DC	11.05±1.37	86.0±3.2	**4590.0±170.0**	53.38±1.55	**0.323901±0.022553**	0.040952±0.008019
	CBA	12.81±1.49	141.1±4.5	7566.5±242.5	54.57±1.24	1.936167±0.059805	0.050272±0.008879
soybean	Original	16.53±2.36	48.8±1.8	3444.5±129.0	**0.56±0.05**	-	0.000166±0.000510
	LDA	16.07±2.35	**43.7±1.4**	**1621.8±52.3**	0.59±0.05	0.004675±0.002709	0.000204±0.000565
	DC	**14.73±2.56**	44.6±1.3	1742.4±49.2	0.88±0.06	**0.000655±0.001013**	**0.000201±0.000561**
	CBA	18.87±2.62	49.8±1.7	1953.6±65.3	0.84±0.06	0.001624±0.001600	0.000233±0.000604
thyroid	Original	2.76±1.17	**5.0±0.1**	48.0±0.3	**0.11±0.02**	-	**0.000022±0.000184**
	LDA	3.90±1.44	5.3±0.4	**25.9±2.1**	0.23±0.02	0.000270±0.000655	0.000032±0.000225
	DC	**1.71±0.94**	**5.0±0.1**	32.0±0.2	0.18±0.03	**0.000078±0.000350**	0.000041±0.000253
	CBA	6.86±2.04	5.3±0.2	34.2±1.8	0.37±0.03	0.000169±0.000519	0.000036±0.000237
wine	Original	3.29±1.39	8.0±0.4	197.1±10.5	0.10±0.02	-	**0.000026±0.000200**
	LDA	**2.47±1.22**	5.7±0.3	**28.3±1.8**	**0.05±0.01**	0.001250±0.001749	0.000026±0.000202
	DC	4.71±1.64	**5.3±0.2**	34.2±1.6	0.17±0.02	**0.000085±0.000364**	0.000030±0.000217
	CBA	3.76±1.69	6.4±0.4	43.2±3.4	0.16±0.02	0.000192±0.000553	0.000032±0.000222

ACKNOWLEDGMENT

This research is supported in part by the Grants-in-Aid for Scientific Research of Japan Society for the Promotion of Science (JSPS), No. 19500128.

REFERENCES

[1] T. Ash, "Dynamic Node Creation in Back-propagation Networks," *Connection Science*, vol. 1, no. 4, pp. 365-375, 1989.

[2] M. Mozer, and P. Smolensky, "Using Relevance to Reduce Network Size Automatically," *Connection Science*, vol. 1, no. 1, pp. 3-16, 1989.

[3] S. Fahlman, C. Lebiere, and D. Touretzky, "The Cascade-Correlation Learning Architecture," *Advances in Neural Information Processing Systems*, vol. 2, pp. 524-532, 1990.

[4] Y. Le Cun, J. Denker, and S. Solla, "Optimal brain damage", *Advances in neural information processing systems*, vol. 2, pp. 598-605, 1990.

[5] 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, 2001.

[6] 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.

[7] H. Hayashi and Q. F. Zhao, "A Comparative Study on GA Based and BP Based Induction of Neural Network Trees", *IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 822-826, 2005.

[8] H. Hayashi and Q. F. Zhao, "A fast algorithm for inducing neural network trees," *IPSJ Journal*, vol. 49, no. 8, pp. 2878-2889, 2008 (in Japanese).

[9] I. T. Jolliffe, *Principal Component Analysis.* Springer, 2002

[10] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification.* Wiley-Interscience, 2 sub edition, 2000.

[11] H. Hayashi and Q. F. Zhao, "Model reduction of neural network trees based on dimensionality reduction," *Proc. of International Joint Conference on Neural Networks*, pp. 1171-1176, 2009.

[12] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stong, *Classification and Regression Trees.* Wadsworth Pub. Co., 1984.

[13] J. Quinlan, *C4.5: Programs for machine learning.* Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993.

[14] Q. F. Zhao, "A New Method for Efficient Design of Neural Network Trees", *Technical Report of IEICE*, vol. PRMU2004-115, pp. 59-64, 2004.