Neural Networks 27 (2012) 81–90

Contents lists available at SciVerse ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

A balanced neural tree for pattern classification

Christian Micheloni [a], Asha Rani [a,*], Sanjeev Kumar [b], Gian Luca Foresti [a]

[a] AVIRES Lab, Department of Mathematics and Computer Science, University of Udine, Via Della Scienze-206, Udine, Italy

[b] Department of Mathematics, Indian Institute of Technology Roorkee, Roorkee-247667, India

ARTICLE INFO

Article history:
Received 2 November 2009
Received in revised form 13 October 2011
Accepted 17 October 2011

Keywords:
Decision trees (DTs)
Neural networks (NNs)
Neural trees (NTs)
Pattern classification
Perceptron
Performance evaluation

ABSTRACT

This paper proposes a new neural tree (NT) architecture, balanced neural tree (BNT), to reduce tree size and improve classification with respect to classical NTs. To achieve this result, two main innovations have been introduced: (a) perceptron substitution and (b) pattern removal. The first innovation aims to balance the structure of the tree. If the last-trained perceptron largely misclassifies the given training set into a reduced number of classes, then this perceptron is substituted with a new perceptron. The second novelty consists of the introduction of a new criterion for the removal of tough training patterns that generate the problem of over-fitting. Finally, a new error function based on the depth of the tree is introduced to reduce perceptron training time. The proposed BNT has been tested on various synthetic and real datasets. The experimental results show that the proposed BNT leads to satisfactory results in terms of both tree depth reduction and classification accuracy.

1. Introduction

Decision trees (DTs) (Rasoul & Landgrebe, 1991) and neural networks (NNs) (Lau & Widrow, 1990) are powerful tools for pattern classification. Several studies have been conducted using these two classifiers. A study comparing these two approaches' effects on various real life applications (e.g., load forecasting, power security and vowel recognition) has been presented by Atlas et al. (1990). Both techniques have advantages and drawbacks. The most existing top-down DT design methods make use of single-feature splits at successive stages of the tree design. While computationally attractive, single-feature splits generally lead to large, deep trees and low performance. On the other hand, one cannot decide an ideal architecture of an NN (number of hidden layers and number of nodes in each hidden layer) for a given training dataset. For this reason, a hybridisation of these two methodologies called neural tree (NT) (Deffuant, 1990; Lippmann, 1987; Sankar & Mammone, 1992; Sethi & Yoo, 1997; Sirat & Nadal, 1990; Utgoff, 1989), has been investigated to combine the advantages of both DTs and NNs.

Some approaches to this problem were motivated by the lack of a reliable procedure for defining the appropriate size of feed-forward neural networks in practical applications. These approaches used DTs to help to determine the topology of NNs to ease training and/or improve generalisation by controlling the number of nodes and connections (Kubat, 1991; Li, Fang, & Jennings, 1992; Sanger, 1991). Some approaches were motivated by the lack of a powerful procedure for computing the appropriate splits or tests in DTs. To improve generalisation, these approaches use NNs to refine the splits or even directly embed NNs as splits nodes in DTs. Murthy, Kasif, and Salzberg (1994), Sankar and Mammone (1992), Sethi and Yoo (1997) and Utgoff and Brodley (1990).

Apart from these approaches, consistent efforts have been made to combine DT and NN into one structure. A new concept called split node has been introduced in simple perceptron based NT (Foresti & Pieroni, 1996) for splitting the training set into two parts when the current perceptron node repeats the same classification of the parent node. This strategy has been provided to guarantee convergence in any case of the tree building process and to reduce misclassification. In Zhaou and Chen (2002), a hybrid decision tree (HDT) has been proposed for the simulation of human reasoning using symbolic learning. Foresti and Micheloni (2002), describe a generalised neural tree (GNT) model. The activation values of each node are normalised to allow for a probability distribution interpretation. The main novelty of the GNT consists in the definition of a new training rule that performs an overall optimisation of the tree. Each time the tree is increased by a new level, the whole tree is re-evaluated. In Foresti and Dolso (2004), an adaptive high-order neural tree (AHNT) employing high-order perceptrons (HOPs) instead of simple perceptrons as nodes of the tree has been suggested. First-order nodes divide the input space with hyperplanes, while HOPs divide the input space arbitrarily. The drawback of this approach is increased complexity and, thus, higher computational cost.

Recently, a neural network tree (NNTree) classifier (Maji, 2008) using a multilayer perceptron (MLP) at each node was

* Corresponding author. Tel.: +39 0432 558423.
E-mail address: asha.rani@dimi.uniud.it (A. Rani).

0893-6080/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2011.10.007

proposed for designing tree-structured pattern classifiers. To limit the depth of the tree, a new uniformity index was introduced. Such an index accomplishes the intuitive goal of reducing the misclassification rate. The performance of the NNTree has been evaluated in different contexts, such as in letter recognition, satellite image classification and splice-junction and protein coding region identification. Experimental comparisons have been proposed with respect to other classifiers. Such comparisons show comparable classification accuracy with significantly smaller trees and, thus, faster classification times. The main drawback is the necessity of an ad hoc definition of some parameters for each context and training set, such as the network architecture (e.g., the number of hidden layers, number of nodes for each layer, etc.) and the uniformity index.

This paper proposes a new NT architecture called balanced NT (BNT) to reduce the size of the tree (both in depth and in the number of nodes), improve the classification with respect to a standard NT and skip the definition of the network architecture as in MLP (Atlas et al., 1990) or MLP-based NT (Maji, 2008). To achieve this result, two main improvements are proposed: (a) perceptron substitution and (b) pattern removal. The first novelty aims to balance the tree structure by substituting the last trained perceptron if some conditions hold. In particular, a new criterion is proposed to check whether the current training set is largely misclassified into a reduced number of classes. If this is the case, the perceptron is substituted with a new perceptron that equally distributes the patterns among the classes. If the perceptron node repeats the same classification of the parent node, splitting nodes as in Foresti and Pieroni (1996) are employed to divide the training set into two subsets.

The second novelty consists of the introduction of a new criterion for the removal of tough training patterns that cause an over-fitting problem. The trained perceptron associates an uncertainty value, measuring the perceptron's classification confidence for each pattern in each class. If uncertainty is high between the two most probable classes, the pattern is removed. Finally, a new error function based on the depth of the tree is introduced to reduce the training time.

The proposed novelties aim to define a new training strategy that does not require the definition of complex network topologies as for NNTree (Maji, 2008) and the setting of ad hoc parameters other than those required for the simple perceptrons (e.g., learning epochs, learning rate, etc.).

The paper is organised as follows. The NT architecture and its related training algorithm are briefly outlined in Section 2. In Section 3, the proposed balanced NT model is presented. Finally, experimental results of both artificial geometric pattern distribution (e.g., chess-board, double spiral, etc.) and real data are presented. A comparison among the proposed BNT, the classic neural tree model and other well-known classifiers is also given.

2. Standard NT algorithm

A neural tree (NT) is a decision tree with a simple perceptron for each intermediate/non-terminal node. Two main phases can be distinguished: (a) training and (b) classification.

2.1. Training phase

In the training phase, the NT is constructed by partitioning a training set consisting of feature vectors and their corresponding class labels to generate the tree in a recursive manner. Such a procedure involves three steps: computing internal nodes, then determining and labelling leaf nodes. An intermediate node groups the input patterns into different classes. In contrast, a leaf node classifies input patterns into a single class which is used as the

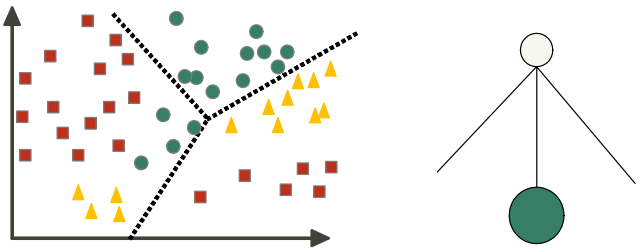

Fig. 1. Partition of the training set by the root node.

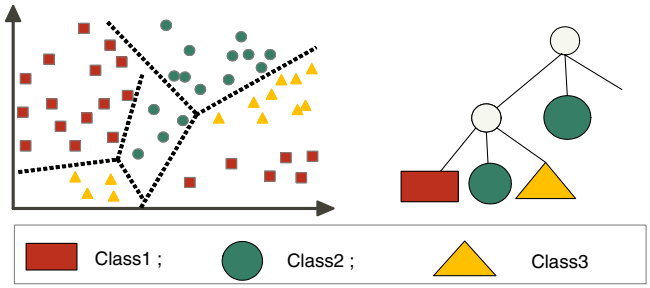

Fig. 2. Partition by the internal node.

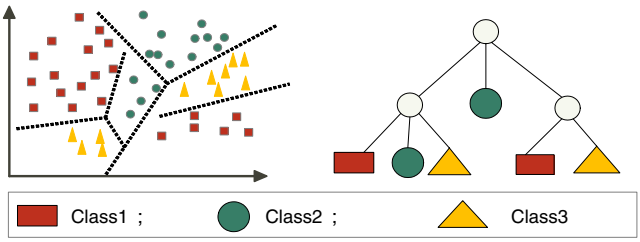

Fig. 3. Final NT.

label for the leaf node. Therefore, NTs are class discriminators that recursively partition the training set such that each generated path ends with a leaf node. The training algorithm, along with the computation of the tree structure, calculates the connection's weights for each node. Each connection's weight is adjusted by minimising a cost function as mean square error or another error function. These weights are used during the classification phase to classify unseen patterns. The NT training algorithm can be summarised as follows.

(1) The patterns of the training set are presented to the root node, and the node is trained to divide the training set into subsets. The process stops when a particular condition is reached.
(2) If a subset is homogeneous, a leaf node is associated and labelled with the corresponding class (See Fig. 1).
(3) If one or more subsets are not homogeneous, a new node is added to the NT at the next level to learn these subsets (See Fig. 2). Go to step 1.
(4) Training stops when all nodes become the leaf node (See Fig. 3).

2.2. Classification phase

For the classification task, unknown patterns are presented to the root node. The class is obtained by traversing the tree in a top-down manner. At each node, activation values are computed on the basis of each connection's weight. Starting from the root, the activation values of the current node determine the next node to consider until a leaf node is reached. Each node applies the winner-takes-all rule so that

$$\mathbf{x} \in \text{class } i \Leftrightarrow \sigma(w_j \dot{x}) \leq \sigma(w_i \dot{x}) \quad \text{for all } j \neq i \tag{1}$$

where σ is the sigmoidal activation function, w_i is the vector of the weights for the connections from the inputs to the ith output and $\mathbf{x}$ is the input pattern. This rule determines the classes associated with the leaf nodes as well as the paths of the internal nodes.

3. The proposed BNT algorithm

3.1. Training phase

Let TS $= \{(p_j, c_i)|j = 1, \ldots, N \wedge i \in [1, M]\}$ be the training set containing N number of d dimensional p_j patterns. Each of them belongs to one of the M classes. The training phase of the proposed BNT algorithm is borrowed from the NT of Foresti and Pieroni (1996) with the introduction of new aspects. A flowchart of the algorithm is shown in Fig. 4. The overall algorithm is described in the following steps.

(1) Create a perceptron (single-layer neural network without hidden layers) and initialise its weight matrix in such a way that the hyperplane generated by these weights passes through the centre of mass of the TS. This is done by fixing the bias as the centroid of the TS. The neural network used is as shown in Fig. 5. The shaded nodes on the left are input layers. The input layer neurons are only used to pass and distribute the inputs; they perform no computation. Thus, the only true layer of neurons is the one on the right. Each of the inputs $x_1, x_2, x_3, \ldots, x_d$ is connected to every neuron in the output layer through its connection weight. Because every value of the outputs $y_1, y_2, y_3, \ldots, y_M$ is calculated from the same set of input values, each output varies based on the connection weights.
(2) Input the training set (TS)/local training set (LTS)[1] to the root node or internal node.
(3) Start training the perceptron by updating the weight matrix to minimise the classification error by optimising a cost function. Note that the perceptron is forced to consider only the classes, which are present at that node, instead of all of the classes.
(4) The pattern is assigned to the class with the highest activation value. If this value is similar to the other activation values, and the classification error of the perceptron is low, the considered pattern is removed from the LTS. The removal criterion will be explained further in Section 3.1.2.
(5) A perceptron divides TS/LTS into $m \leq M$ non-empty groups $(S_1, \ldots, S_m)$.
(6) If $m > 1$, a new level of m child nodes is created. Each i-th child node is assigned to the corresponding LTS S_i.
(7) If $m = 1$, LTS is divided into two groups, S_l and S_r, using the splitting rule (Foresti & Pieroni, 1996) and producing two child nodes. Go to step 9.
(8) If the classification performed at the current node is inaccurate and unbalanced (i.e., one class receives the large majority of the patterns), the perceptron weights are replaced by the initial weights, which divide the training set with a hyperplane passing through the centre of the mass. This solution allows the balanced distribution of patterns among all classes. This step will be explained in more detail in Section 3.1.1.
(9) If all the patterns present at a child node belong to the same class, the node is made a leaf node and labelled with the class of such patterns. If all child nodes become leaf nodes, the current perceptron ends with M leaf nodes, one for each class; otherwise, go to step 2.
(10) If all current nodes are leaf nodes, the algorithm ends. Otherwise, go to step 2 to train the remaining perceptrons.

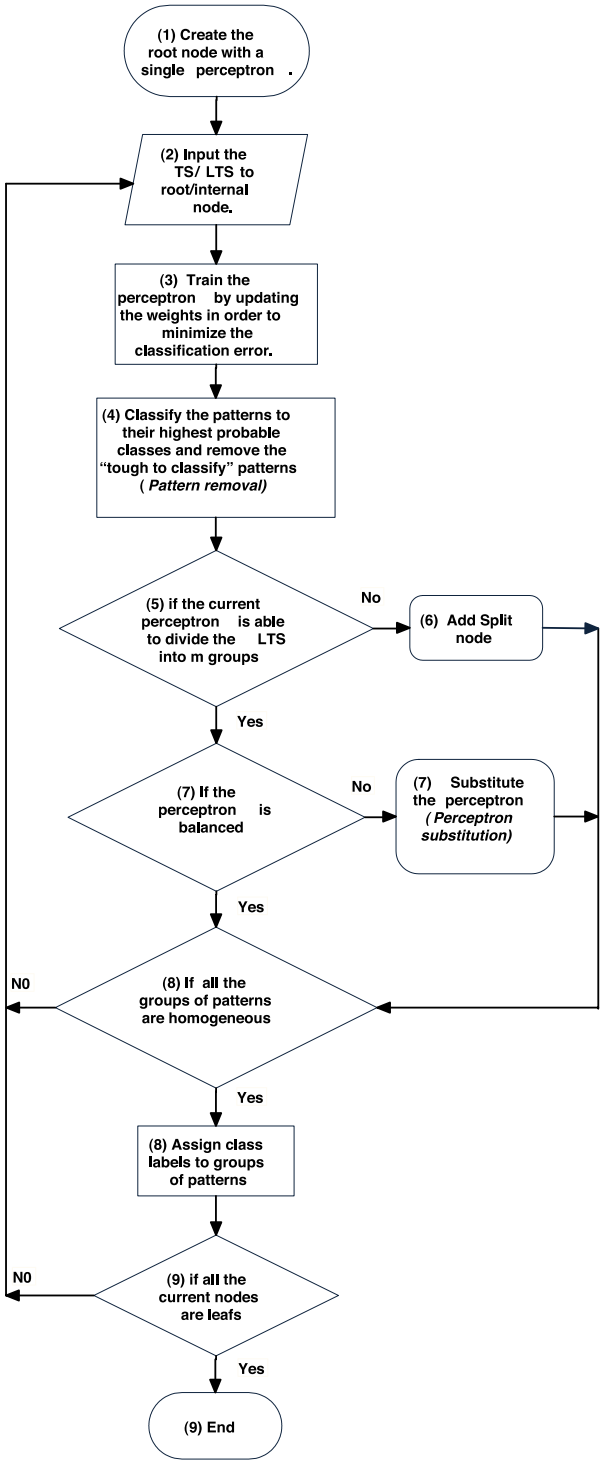

Fig. 4. Flowchart explaining the proposed BNT algorithm.

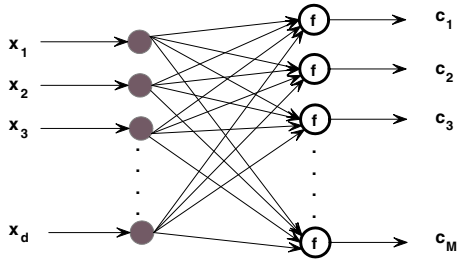

Fig. 5. The single layer neural network used at each node.

Let $\bar{\delta}$ be the error defined as the mean error computed on all output neurons and patterns,

$$\bar{\delta} = \frac{1}{KM} \sum_{k=1}^{K} \sum_{i=1}^{M} \delta_i^k \tag{2}$$

where K is the total number of patterns at the current node and the error δ_i^k is the difference of output o_i^k and the target's class t_i^k, computed as follows:

$$\delta_i^k = \left| t_i^k - o_i^k \right|, \quad i = 1, \ldots, M \tag{3}$$

where $o_i^k \in (0, 1)$, and

$$t_i^k = \begin{cases} 1, & \text{if pattern } k \text{ belongs to class } i \\ 0, & \text{otherwise.} \end{cases}$$

As the tree grows during the training phase, the training set continues splitting into smaller subsets. It can be observed that in smaller subsets with a reduced number of classes, the boundaries become more defined. As a result, the error reduces faster. Therefore a more precise perceptron is required as the depth of the tree grows. The precision depends on the number of classes as well. A criterion considering the depth of the tree at the node being processed and the number of classes present has therefore been defined to compute the *stop rule* of the training process.

Each perceptron is trained either until the error $\bar{\delta}$ becomes less than $1/\bar{m}^l$, where $\bar{m}$ is the number of classes actually present in the current LTS and l is the depth of the node or until there is no further reduction in the error for a given number of *Wait* epochs. The criterion for the perceptron to stop the training can be summarised by the following equation:

$$\begin{aligned} &\bar{\delta} < 1/\bar{m}^l \quad \text{or} \\ &(\bar{\delta}^{(n)} - \bar{\delta}^{(n-1)}) < \text{ toler for Wait number of epochs} \end{aligned} \tag{4}$$

where $\bar{\delta}^n$ and $\bar{\delta}^{n-1}$ are the mean errors in consecutive iterations, and *toler* is a small number of the order 10^{-6}. This criterion for stopping the training process is formulated in an effort to reduce the training time.

Three different modalities for splitting the TS have been included in the proposed algorithm. The first is performed by a trained perceptron, the second by an untrained perceptron and the third by a split node.

The studied strategy bases its decision on which splitting criteria must be adopted on the basis of the classification errors. In particular, at each node the global classification error is computed as follows:

$$E_t = 1 - \frac{K_c}{K_t} \tag{5}$$

where K_c is the number of correctly classified patterns and K_t is the total number of patterns presented at the current node.

In addition, to better localise the error among the classes, for each class a misclassification error is computed as

$$E_i = 1 - \frac{K_{c_i}}{K_{t_i}} \tag{6}$$

where K_{c_i} is the number of correctly classified patterns of class i and K_{t_i} is the total number of patterns classified as class i.

[1] LTS is a subset of the TS. It is the input to a child node that resulted from the classification/split performed by its parent node.

3.1.1. Perceptron substitution

Studying the error trends for different datasets reveals, that if the total error is relevant and the difference between the lowest and highest errors among the classes is considerable, the degree of correctness in the classification will be low. As a consequence, the tree will be unbalanced, making a good trade-off between errors and tree depth impossible. For these reasons, the trained perceptron is not considered optimal.

The first novelty of the proposed solution is to substitute the badly trained perceptron with an initial untrained one. This is generated by fixing the bias such that the hyperplane passes through the centroid of the local training set. This procedure ensures a uniform distribution of the patterns included in the current LTS among all the classes. As a result of this substitution, the tree is more balanced and its depth reduces.

Considering that the training phase of a perceptron may stop before reaching the stop criterion described in Eq. (4), new criteria have been studied to decide whether a perceptron has been badly trained. In particular, two main properties have been identified.

- The perceptron's classification error has to be equally distributed among the child nodes.
- The initial error of the perceptron should be reduced significantly after training.

The first property wants to avoid child nodes with high classification errors and child nodes with low classification errors. Therefore the following should hold:

$$E_i \approx \frac{E_t}{\bar{m}} \quad \forall i \in \{1, \ldots, \bar{m}\} \tag{7}$$

where E_i is the classification error for the i-th child node.

To satisfy the second property, it has been decided that the initial error E_0 is significantly reduced after training if such a reduction is quadratic with respect to the number of output classes $\bar{m}$. Therefore the following should hold:

$$E_i \leq \frac{E_0}{\bar{m}^2}. \tag{8}$$

Combining Eqs. (7) and (8), the following holds:

$$E_t \approx E_i \cdot \bar{m} \leq \frac{E_0}{\bar{m}^2} \cdot \bar{m} = \frac{E_0}{\bar{m}}. \tag{9}$$

A criterion for judging whether the classification performed by the trained perceptron is acceptable is as follows:

$$E_t \leq \frac{E_0}{\bar{m}} \quad \text{and} \quad (E_{\max} - E_{\min}) \leq E_t \tag{10}$$

where $E_{\max} = \max_i\{E_i\}$, $E_{\min} = \min_i\{E_i\}$. The difference between the maximum and minimum classification errors has been adopted to check for a good distribution of error among all the child nodes. Indeed, it could be maximum $\frac{E_0}{\bar{m}}$ that is the initial error equally distributed over the number of activated outputs.

As an initial qualitative justification of the proposed approach, the training results obtained with a standard NT and the proposed BNT on a simple two-class chess-board problem are shown in Fig. 6. At the root node, the trained perceptron is able to separate only a few patterns, which yield a high total error, $E_t = 0.3593$, and relevant difference in the errors among the two classes, i.e., $E_1 = 0$ and $E_2 = 0.4181$. Using the standard NT training procedure and, keeping such a perceptron yields a deeper NT. On the other hand, adopting the proposed strategy replaces the root perceptron with a perceptron whose weights define a hyperplane passing through the centre of the mass of the training set. Such a node divides the training set better, allowing it to define a more balanced NT: an NT with a lower depth.

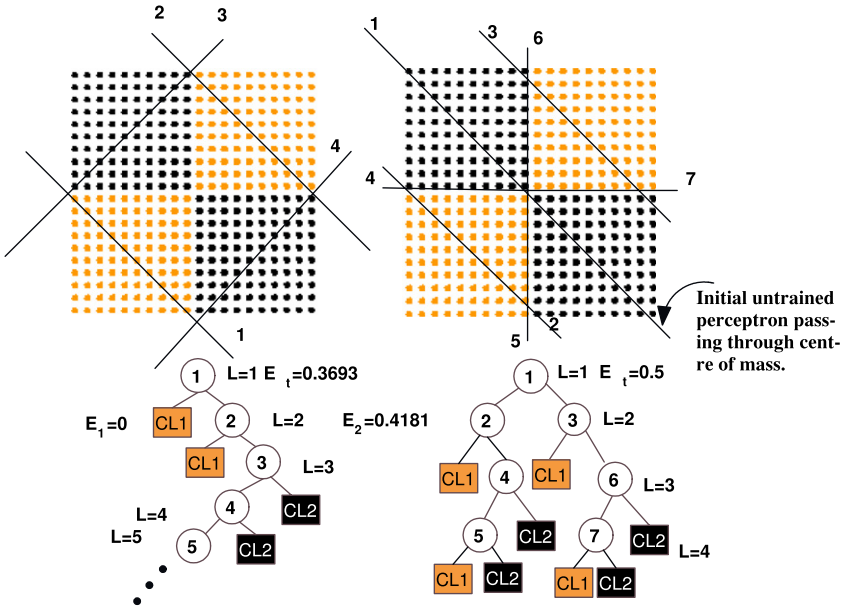

Fig. 6. Tree formation using NT (left) and BNT (right). The tree obtained with the proposed BNT algorithm is balanced and converges quickly whereas with the classic algorithm, it is unbalanced and deep.

In the case that a perceptron is unable to separate the training set into two or more classes, the third type of splitting is adopted. For this purpose, a splitting rule (Foresti & Pieroni, 1996) is considered to divide the data into two groups with nearly equal cardinality. The splitting rule consists of adding a new node, the splitting node, to the NT architecture. This node must satisfy some constraints: (a) the TS/LTS should be divided into at least two subsets, (b) the cardinality of these subsets should be similar and, (c) the splitting should be performed on the features with maximal variance, i.e., the most discriminating features. At first, the barycentres $\bar{x}^1$ and $\bar{x}^2$ of the two classes of the TS/LTS with higher cardinality are computed, and the component which maximises the L_1 norm is identified. Let k be the selected component. Then a splitting hyperplane is computed as the hyperplane orthogonal to the k axis and passing through the median point between $\bar{x}^1$ and $\bar{x}^2$.

3.1.2. Pattern removal

Concerning the third step of the proposed training algorithm, it is apparent that even a well-trained perceptron is unable to classify certain patterns. Such patterns require specialised classifications and force the algorithm to produce deeper trees. Moreover, such patterns produce over-fitting of the data, reducing the generalisation of the tree.

To avoid this situation, a pre-pruning strategy is applied. When a perceptron is shown to be a good classifier after training by the aforementioned strategy, the way it classifies some patterns allows to define a criterion for their removal from the child node's training sets. These patterns may be considered boundary line patterns (see Fig. 7 Left) or patterns surrounded by patterns of another class (see Fig. 7 Right). These patterns are always difficult to classify in the sense that the perceptron has a high uncertainty among different classes.

The decision to remove a pattern from the training set is made based upon the following aspects:

- probability of a pattern belonging to a class
- total classification error of the perceptron.

The first aspect considers classification uncertainty by checking whether two or more classes have a similar high probability of describing the pattern. The second aspect represents the reliability of the training process. Therefore, if a relevant uncertainty exists in the pattern classification and if the perceptron is reliable, the pattern under consideration is considered difficult to classify and therefore removed from the training set of the child.

The classification probability is modelled by normalising the activation values so that they form a distribution. Given two classes c_i and c_j and the respective probabilities $P(c_i|\mathbf{x})$ and $P(c_j|\mathbf{x})$ of describing a pattern $\mathbf{x}$, the uncertainty between the two classes is provided by

$$h_{ij} = \left|P(c_i|\mathbf{x}) - P(c_j|\mathbf{x})\right|. \tag{11}$$

Let $P(c_{\max 1}|\mathbf{x})$ and $P(c_{\max 2}|\mathbf{x})$ be the maximum and second-maximum classification probabilities, representing the two most probable classes to which $\mathbf{x}$ should belong. $h_{\max} = |P(c_{\max 1}|\mathbf{x}) - P(c_{\max 2}|\mathbf{x})|$ is defined to represent the uncertainty of the trained perceptron. The lower the difference between the maximum and second-maximum classification probabilities, the more unlikely the perceptron is to classify the pattern correctly. A probability less than 0.2 is considered sufficiently low, so a threshold Th $\in$ [0.05, 0.20] is considered to limit the value of $h_{\max}$.

Concerning the reliability of the perceptron classification, studying the behaviours of the nodes with respect to their depth, suggested to define the reliability $R = 1/\bar{m}^l$. If the total error E_t of the node being processed is lower than the reliability factor R, the perceptron can be considered reliable.

To summarise the criteria for deciding whether a pattern must be removed from the training set, the decision rule is as follows:

if $h_{\max} <$ Th and $E_t < R$ and $\mathbf{x} \notin c_{\max 1}$ **then**
  Pattern $\mathbf{x}$ must be removed
else
  Pattern $\mathbf{x}$ is included in $LTS_{\max 1}$
end if

For the X-OR problem in Fig. 7, 400 patterns are used for training and 10,000 patterns are classified by the trained NT model with the standard algorithm and the proposed algorithm. The NT constructed with the standard algorithm reaches a depth of 8 levels with 27 nodes, whereas the BNT constructed with the proposed algorithm reaches a depth of only 5 levels with 15 nodes. The classification accuracies obtained are 98.40% and 98.43% with

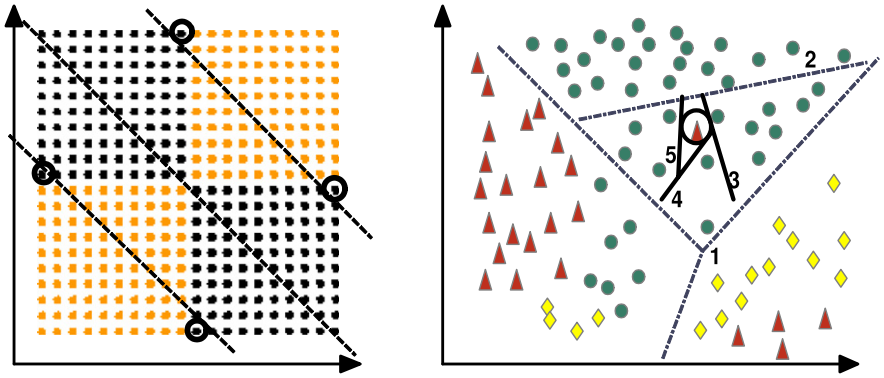

Fig. 7. Patterns removed in a two-class symmetric chessboard dataset during the training phase (left), patterns removed in a three-class random dataset during the training phase (right). If this particular pattern is not removed, the classic NT requires three more hyperplanes (thick black lines in (right)) to separate the pattern.

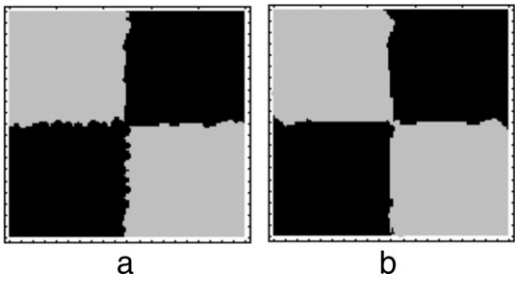

Fig. 8. Classification accuracy obtained on a two-class chessboard dataset using 400 patterns for training, (a) NT and (b) BNT.

NT and BNT, respectively. Therefore it is noted that the depth of the tree is reduced by 38% without diminishing the classification accuracy (see Fig. 8).

4. Experimental results

This section presents some experimental results on synthetic and real datasets. Synthetic datasets are used to compare the proposed tree model with the standard one. The classification accuracy is compared on various real datasets with existing algorithms, such as Cellular Automata (CA) (Maji, Shaw, Ganguly, Sikdar, & Chaudhuri, 2003), C4.5 (Eggermont, Kok, & Kosters, 2004; Qin & Lawry, 2005; Quinlan, 1993), Bayesian (Cheeseman & Stutz, 1996), MLP (Hertz, Krogh, & Palmer, 1991; Lippmann, 1987; Tanwani, Afridi, Shafiq, & Farooq, 2009), NNTree (Maji, 2008), Naive Bayes (Kotsiantis & Pintelas, 2005) and NT (Sankar & Mammone, 1992). For the training of BNT for all datasets only one parameter, learning rate η is selected variably.

4.1. Synthetic datasets

We have already discussed the performance of BNT over NT for a two-class chessboard dataset. Now let us consider a more complex problem, i.e., a symmetrically distributed four-class chessboard dataset (see Fig. 9). The dataset consists of 400 patterns equally distributed among four classes. NT and BNT are trained using 400 patterns and on the patterns remaining after randomly removing 10%, 20%, 30% and 40% of the patterns. Performance is evaluated in terms of the tree depth as well as the classification accuracy. A five-fold cross-validation is performed and averaged results are presented. The average depth of NT is greater than the average BNT depth (see Fig. 10). Considering the average classification accuracy shown in Fig. 11 for both algorithms, the gain of the proposed technique is considerable, as Fig. 9 also demonstrates.

It is also noticeable in these charts that the behaviour of the proposed BNT solution is more stable or linear compared to the standard NT with respect to the removal of patterns from the training set.

The second experiment consists of a more complex problem frequently used to check the performance of NTs. The training set is composed of 1110 patterns distributed uniformly across two crossed spirals. The results obtained with NT and BNT are shown in Fig. 12. The tree obtained with the NT algorithm is composed of 354 nodes, and the depth is 18. The tree obtained with the BNT algorithm converges at a depth of 14 with 279 nodes, and 7 patterns are removed during the training process. The classification accuracies obtained with the NT and the BNT are 86.64 and 89.37, respectively.

Again, evaluating both the depth of the trained tree and the classification accuracy, the proposed BNT algorithm achieves improved results with respect to the standard NT algorithm.

4.2. Real datasets

The results obtained on real datasets with the proposed algorithm are compared with various algorithms reported in the literature. The datasets were taken from the UCI Machine Learning Data Repository (Asuncion & Newman, 2007).

4.2.1. Satellite image classification

The original Landsat data for this database is generated from data purchased from NASA by the Australian Centre for Remote Sensing and used for research at the University of New South Wales. The database was generated taking a small section (82 rows and 100 columns) from the original data (Michie, Spiegelhalter, Taylor, & Campbell, 1995). The data were divided into training and test sets, with 4435 examples in the training set and 2000 in the test set. During BNT's training process, 10 patterns were removed. A comparison of the results for this dataset is presented in Table 1 using the well-known Bayesian, C4.5, MLP, CA, NNTree and NT algorithms and the classification accuracy as a parameter. The depth and the number of nodes involved in tree-based classifiers are also compared. Most of the time, NNTree is more compact with fewer nodes than BNT, but it is still comparable because each node of NNTree is a multilayer perceptron.

The experiments were executed only for NT and BNT, whereas the results for other classifiers were taken from published works. Because the classification time can depend on various parameters (processor speed, memory size and efficiency of code written to implement the algorithm), to avoid all these factors, the classification time has been modelled in terms of the number of nodes a pattern has to traverse to be classified. In this way, the classification time is directly proportional to the tree depth. The deeper the tree is, the more time a pattern takes to be classified (reach leaf node). The path length (number of nodes to reach a

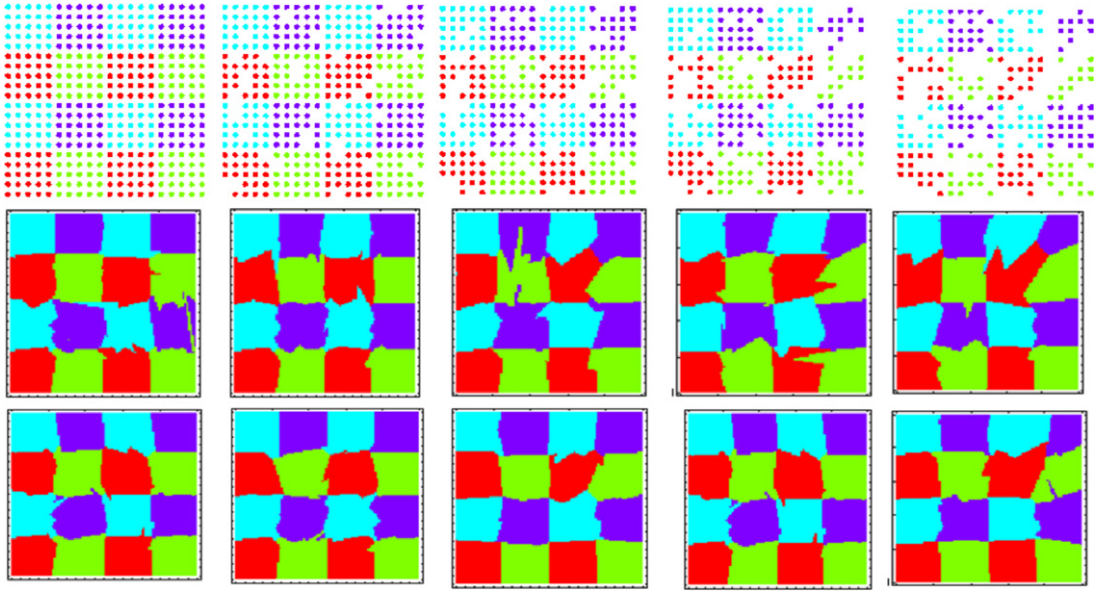

Fig. 9. First row—400 patterns distributed symmetrically among four classes and then after removing randomly 10%, 20%, 30% and 40% of the patterns. Second row—classification performed by NT. Third row—classification obtained by BNT.

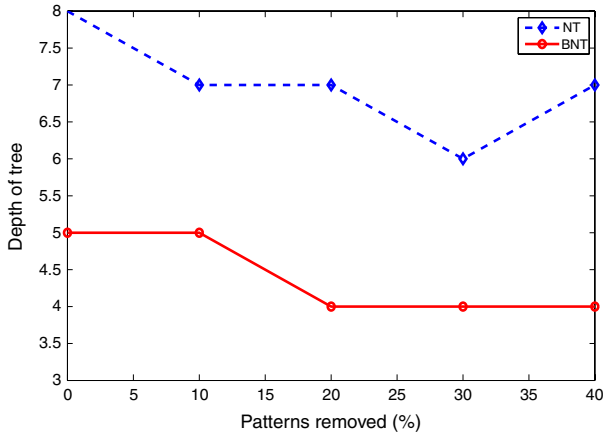

Fig. 10. Depth of NT and BNT obtained when trained with 400 patterns and the patterns remaining after randomly removing 10%, 20%, 30% and 40% of the 400 patterns of a four-class chessboard dataset.

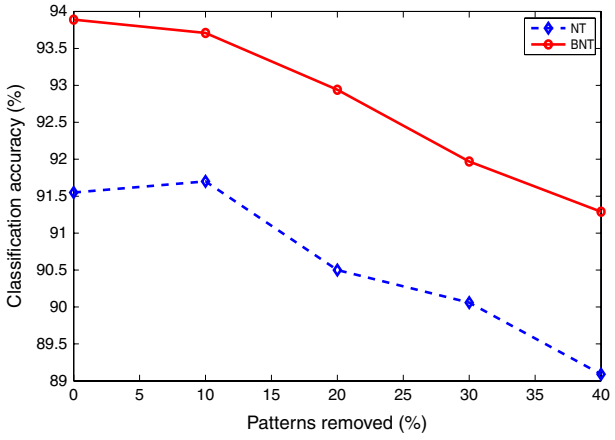

Fig. 11. Classification accuracy obtained with NT and BNT when trained with 400 patterns and the patterns remaining after randomly removing 10%, 20%, 30% and 40% of the 400 patterns of a four-class chessboard dataset.

Table 1
Classification accuracy of different algorithms for the satellite image database.

Algorithms	Accuracy (%)	Depth of tree	Total nodes	Avg. path length
Proposed BNT	85.25	14	592	8.4485
NT	NA	NA	NA	NA
NNTree	87.1	5	25	–
MLP	86.2	–	–	–
Bayesian	85.4	–	–	–
CA	77.5	–	–	–
C4.5	85.2	21	433	–

leaf node) has been computed for each pattern, and the average path length of all the patterns has been used as a parameter for the comparison between NT and BNT (column 5 of each table).

It is worth noting that the classification accuracy of the proposed solution is better than the other existing algorithms except MLP, NNTree and Bayesian. Even if it is lower, it is still comparable or equivalent. The BNT converges at depth 14, which is 33% less than the depth of the tree obtained with C4.5. The results used for comparison are taken from Maji (2008).

Once more, it is interesting to notice that the proposed solution reduces the depth of the trained tree with respect to other tree-based classifiers while keeping the classification accuracy and without requiring the definition of networks' topologies in terms of hidden layers and nodes.

4.2.2. Identification of splice junctions in DNA

This section presents the application of BNT to finding the splice junction in anonymous DNA sequences. In bioinformatics, one of the major tasks is the recognition of certain DNA subsequences that are important in the expression of genes. A DNA sequence is basically a string over the alphabet $D = \{A, C, G, T\}$. DNA contains the information with which a cell constructs protein molecules. The cellular expression of protein proceeds from the creation of messenger ribonucleic acid (mRNA) copies from the DNA template. This mRNA is then translated into a protein. One of the most unexpected findings in molecular biology is that large pieces of the mRNA are removed before it is translated further (Breathnach, Mandel, & Chambon, 1997). The utilised sequences are known as exons while the removed sequences are known as introns or intervening sequences. The points at which DNA

Fig. 12. Left to right: Training dataset, classification obtained with NT, classification obtained with BNT.

Table 2
Classification accuracy of different algorithms for the DNA database.

Algorithms	Accuracy (%)	Depth of tree	Total nodes	Avg. path length
Proposed BNT	91.67	5	19	3.4634
NT	91.16	11	42	4.6538
NNTree	94.2	3	5	–
MLP	91.4	–	–	–
Bayesian	90.3	–	–	–
C4.5	93.3	12	127	–
CA	87.9	–	–	–

is removed are known as splice junctions. The splice junction problem is to determine which of the following three categories a specified location in a DNA sequence falls into: (1) exon/intron borders, referred to as donors; (2) intron/exon borders, referred to as acceptors; and (3) neither.

The dataset used in this problem is a processed version of the Irvine Primate splice junction database (Michie et al., 1995). Each of the 3186 examples in the database consists of a window of 60 nucleotides, each represented by one of four symbolic values $\{A, G, C, T\}$. The classification of the middle point in the window is one of the intron boundary, exon boundary or neither. Processing involves the conversion of the original 60 symbolic attributes to 120 binary attributes and the conversion of symbolic class labels to numeric labels. The training set of 2000 is chosen randomly from the dataset and the remaining 1186 examples are used as the test set.

During the training process of the BNT, 6 patterns have been removed. The comparison of results for this dataset is presented in Table 2 with the well-known algorithms Bayesian, C4.5, MLP, CA, NNTree and NT considering the classification accuracy as a parameter. It is notable that the classification accuracy of the proposed solution is better than all other algorithms except C4.5 and NNTree, to which it is still comparable or considered equivalent. The BNT is composed of 19 nodes with depth 5, while NT is composed of 42 nodes with depth 11. The depth of BNT is 58% less than C4.5 and 55% less than NT. Again, it is established that the proposed solution reduces the depth of the trained tree with respect to other tree based classifiers without diminishing the classification accuracy.

4.2.3. Predicting the cellular localisation sites of proteins

Proteins from *E.coli* are classified into 8 classes: inner membrane lipoproteins (imL), outer protein membrane (omL), inner protein membrane with a cleavable signal sequence (imS), other outer membrane proteins (om), periplasmic proteins (pp), inner membrane proteins with an uncleavable signal sequence (imU), inner membrane protein without a signal sequence (im) and cytoplasmic protein (cp). Seven features were calculated from the amino acid for use in classification. For more details on this dataset, see Horton and Nakai (1996).

Table 3
Classification accuracy of different algorithms for the E.coli database.

Algorithms	Accuracy (%)	Depth of tree	Total nodes	Avg. path length
Proposed BNT	83.21	3	12	2.4159
NT	82.42	5	16	2.7876
NB boost	81.99	–	–	–
MLP	82.29	–	–	–
C4.5	80.36	–	–	–

During the training process of the BNT, 2 patterns were removed. The results of comparison of this dataset with well-known algorithms Naive Bayes, MLP, C4.5 and NT considering the classification accuracy as parameter are presented in Table 3. It is established that the classification accuracy of the proposed solution is higher than that of other algorithms. The BNT is composed of 12 nodes with depth 3, and NT is composed of 16 nodes with depth 5. The depth of BNT is 40% less than that of NT. It is interesting to note that the proposed solution reduces the depth of the trained tree with respect to NT while keeping the classification accuracy.

4.2.4. Iris plant classification

One of the most popular and best known databases of the neural network application is the iris plant dataset, created by R.A. Fisher in 1950. There are 150 instances, 50 each in three classes. The existing 4 numerical attributes for identification of the classes are sepal length, sepal width, petal length and petal width in centimetres. The classes are Iris setosa, Iris versicolor and Iris virginica. In this application, 33 data from each class are used for training, and 17 are separated for testing purposes.

During the training process of the BNT, 3 patterns were removed. The results' comparison for this dataset with well-known algorithms Bayesian, Naive Bayes, C4.5, Bagged C4.5, Boosted C4.5, and NT considering the classification accuracy as parameter are presented in Table 4. The experimental results establish that the classification accuracy of the proposed solution is better than that of other existing algorithms. The depth of the tree obtained with NT is 8 with 12 nodes, whereas with BNT the tree converges at depth 3 with 4 nodes, which is 63% less than that of NT.

It is interesting to notice that the proposed solution reduces the depth of the trained tree with respect to NT while keeping the classification accuracy.

4.2.5. Classification of radar returns from the ionosphere

The Johns Hopkins University Ionosphere database (Sigillito, Wing, Hutton, & Baker, 1989) contains 350 patterns, each with 34 numeric attributes. There are 2 classes, i.e., good and bad. This radar data were collected by a system in Goose Bay, Labrador. The system consists of a phased array of 16 high-frequency antennae with a total transmitted power of 6.4 kW. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

Table 4
Classification accuracy of different algorithms for the Iris database.

Algorithms	Accuracy (%)	Depth of tree	Total nodes	Avg. path length
BNT	96.07	3	4	2.33
NT	96.07	8	12	3.0588
Bayesian	93.20	–	–	–
Naive Bayes	95.53	–	–	–
C4.5	94.1	–	–	–
Bagged C4.5	95	–	–	–
Boosted C4.5	95	–	–	–

Table 5
Classification accuracy of different algorithms for the ionosphere database.

Algorithms	Accuracy (%)	Depth of tree	Total nodes	Avg. path length
Proposed BNT	94.6	2	2	1.8733
NT	94.6	2	2	1.8733
C4.5	91.1	–	–	–
Boosted C4.5	94.2	–	–	–
Bagged C4.5	93.8	–	–	–
Naive Bayes	92.7	–	–	–
MLP	96	–	–	–

During the training process of the BNT, 2 patterns were removed. The results compared for this dataset with well-known algorithms Naive Bayes, C4.5, Bagged C4.5, Boosted C4.5, MLP, and NT considering the classification accuracy as parameter are presented in Table 5. It is worth noting that the classification accuracy of the proposed solution is better than other existing algorithms with the exception of MLP, but even in the case of MLP, it is comparable. The tree obtained from the NT algorithm and the tree obtained from the BNT algorithm converge at a depth of 2 with 2 nodes. For this dataset, all the perceptrons have been considered efficient. Therefore, the BNT and the NT are identical, such that no perceptron substitution occurred. The domain is extremely simple; thus all algorithms behave well for this dataset.

4.2.6. Letter recognition

The dataset used here was constructed by David J. Slate of Odesta Corporation, Evanston, IL 60201. The objective is to classify each of a large number of black and white rectangular pixel displays as one of the 26 capital letters of the English alphabet. The character images produced are based on 20 different fonts and each letter within these fonts is randomly distorted to produce a file of 20,000 unique images. For each image, 16 numerical attributes have been calculated using edge counts and measures of statistical moments, which are scaled and discretised into a range of integer values between 0 and 15. As one of the fonts used, Gothic Roman, appears very different from the others, perfect classification performance is unlikely to be possible with this dataset (Michie et al., 1995).

During the BNT training process, 10 patterns were removed. The results compared for this dataset with well-known algorithms Bayesian, C4.5, MLP, CA, NNTree and NT considering the classification accuracy as parameter are presented in Table 6. Because the dataset is enormous and complicated, NT is unable to converge, but BNT converges with the classification results comparably to the other algorithms.

5. Conclusions

This paper presented a new strategy of building a neural tree classifier based on simple perceptrons. The advantage of using simple perceptrons instead of complex MLP structures is possible and efficient because of the two main novelties we have introduced. In particular, balancing the tree and removing difficult patterns allow us to simplify the network structure of the nodes without introducing overfitting problems. With respect to standard perceptrons, no new parameters must be fixed during the design of the training strategy. Experimental results demonstrate that the proposed solution adapts well to different classification problems, showing a good generalisation property. In particular, the classification accuracy is always close to the best performance, while the depth of the generated tree is always less than the depth of trees generated with other available techniques. As a final but no less important consideration, the proposed perceptron-based neural tree always achieves convergence, while in some cases, the standard NT algorithm does not converge.

Table 6
Classification accuracy of different algorithms for the letter database.

Algorithms	Accuracy (%)	Depth of tree	Total nodes	Avg. path length
Proposed BNT	83.90	21	2921	10.84
NT	NA	NA	NA	NA
Bayesian	87.4	–	–	–
CA	84.4	–	–	–
C4.5	86.6	20	2127	–
NNTree	82.9	6	352	–
MLP	67.2	–	–	–

Acknowledgement

This work is partially supported by Ministry of Italian University and Scientific Research (MIUR).

References

Asuncion, A., & Newman, D. J. (2007). Uci machine learning repository. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Atlas, L., Cole, R., Muthusamy, Y., Lippman, A., Connor, J., Park, D., et al. (1990). A performance comparison of trained multilayer perceptrons and trained classification trees. *Proceedings of the IEEE*, *78*(10), 1614–1619.

Breathnach, R., Mandel, J., & Chambon, P. (1997). Ovalbumin gene is split in chicken dna. *Nature*, *270*(5635), 314–319.

Cheeseman, P., & Stutz, J. (1996). Bayesian classification (autoclass): theory and results. In *Advances in knowledge discovery and data mining* (pp. 153–180). CA, USA: American Association for Artificial Intelligence Menlo Park.

Deffuant, G. (1990). Neural units recruitment algorithm for generation of decision trees. In *Proceedings of the international joint conference on neural networks: Vol. 1* (pp. 637–642).

Eggermont, J., Kok, J. N., & Kosters, W. (2004). Genetic programming for data classification: partitioning the search space. In *SAC, 1001*.

Foresti, G. L., & Dolso, T. (2004). An adaptive high-order neural tree for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *34*(2), 988–996.

Foresti, G. L., & Micheloni, C. (2002). Generalized neural trees for pattern classification. *IEEE Transactions on Neural Networks*, *13*(6), 1540–1547.

Foresti, G., & Pieroni, G. (1996). Exploiting neural trees in range image understanding. *Pattern Recognition Letters*, *19*(9), 869–878.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the theory of neural computation. Vol. 1. Santa Fe Institute Studies in the Sciences of Complexity*.

Horton, P., & Nakai, K. (1996). A probablistic classification system for predicting the cellular localization sites of proteins. In *Proceeding of the fourth international conference on intelligent systems for molecular biology* (pp. 109–115).

Kotsiantis, S. B., & Pintelas, P. (2005). Logitboost of simple baysian classifier. *Informatica*, *29*, 53–59.

Kubat, M. (1991). Decision trees can initialize radial-basis function networks. *IEEE Transactions on Neural Networks*, *9*(5), 813–821.

Lau, C., & Widrow, B. (1990). Special issue on neural networks. *Proceedings of the IEEE*, *78*.

Li, T., Fang, L., & Jennings, A. (1992). Structurally adaptive self-organizing neural trees. In *Proceedings of the international joint conference on neural networks: Vol. 3* (pp. 329–334).

Lippmann, R. (1987). An introduction to computing with neural nets. *IEEE Acoustics, Speech, and Signal Processing Magazine*, *4*(2), 4–22.

Maji, P. (2008). Efficient design of neural network tree using a single spilitting criterion. *Nerocomputing*, *71*, 787–800.

Maji, P., Shaw, C., Ganguly, N., Sikdar, B. K., & Chaudhuri, P. P. (2003). Theory and application of cellular automata for pattern classification. *Fundamenta Informaticae*, *58*(34), 321–354.

Michie, D., Spiegelhalter, D. J., Taylor, C. C., & Campbell, J. (1995). *Machine learning, neural and statistical classification*. Upper Saddle River, NJ, USA: Ellis Horwood.

Murthy, S. K., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, *2*, 1–32.

Qin, Z., & Lawry, J. (2005). Decision tree learning with fuzzy labels. *Information Sciences*, *173*, 91–129.

Quinlan, J. (1993). C4.5: Programs for machine learning.

Rasoul, S., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, *21*(3), 660–674.

Sanger, T. D. (1991). A tree-structured adaptive network for function approximation in high-dimensional spaces. *IEEE Transactions on Neural Networks*, *2*(2), 285–293.

Sankar, A., & Mammone, R. (1992). Neural tree networks. In *Neural network: theory and application* (pp. 281–302). San Diego, CA, USA: Academic Press Professional, Inc., (Chapter).

Sethi, I. K., & Yoo, J. (1997). Structure-driven induction of decision tree classifiers through neural learning. *Pattern Recognition*, *30*(11), 1893–1904.

Sigillito, V. G., Wing, S. P., Hutton, L. V., & Baker, K. B. (1989). *Classification of radar returns from the ionosphere using neural networks*: *Vol. 10*. Johns Hopkins APL Technical Diges.

Sirat, J., & Nadal, J. (1990). Neural trees: a new tool for classification. *Neural Network*, *1*, 423–448.

Tanwani, A., Afridi, J., Shafiq, M., & Farooq, M. (2009). Guidelines to select machine learning scheme for classifcation of biomedical datasets. In *LNCS*: *Vol. 5483*. *EVOBIO* (pp. 128–139).

Utgoff, P. E. (1989). Perceptron tree: a case study in hybrid concept representation. *Connection Science*, *1*(4), 377–391.

Utgoff, P., & Brodley, C. (1990). An incremental method for finding multivariate splits for decision trees. In *Proceedings of the seventh international conference (1990) on Machine learning* (pp. 58–65). Austin, Texas, United States: Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Zhaou, Z., & Chen, Z. (2002). Hybrid decision tree. *Knowledge-Based Systems*, *15*(8), 515–528.