Incorporating Linear Discriminant Analysis in Neural Tree for Multidimensional Splitting

Asha Rani[1], Sanjeev Kumar[1], Christian Micheloni[2], Gian Luca Foresti[2]

[1] *Department of Mathematics, Indian Institute of Technology, Roorkee, Roorkee-247667, Uttrakhand, India*
[2] *Department of Mathematics and Computer Science, University of Udine, Via Della Scienze-206, Udine-33100, Italy*

Abstract

In this paper, a new hybrid classifier is proposed by combining neural network and direct fractional-linear discriminant analysis (DF-LDA). The proposed hybrid classifier, neural tree with linear discriminant analysis called NTLD, adopts a tree structure containing either a simple perceptron or a linear discriminant at each node. The weakly performing perceptron nodes are replaced with DF-LDA in an automatic way. Taking the advantage of this node substitution, the tree building process converges faster and avoids the over-fitting of complex training sets in training process resulting a shallower tree together with better classification performance. The proposed NTLD algorithm is tested on various synthetic and real datasets. The experimental results show that the proposed NTLD leads to very satisfactory results in terms of tree depth reduction as well as classification accuracy.

Key words: Decision tree, Linear discriminant analysis, Neural network, Neural tree, Pattern classification.

1 Introduction

The classification is a machine learning procedure in which individual items are placed into groups based on the characteristics inherent in the items and a

* **Corresponding Author**: Sanjeev Kumar
Email: *malikfma@iitr.ernet.in*
Dept. of Mathematics, Indian Institute of Technology, Roorkee, Roorkee-247667, Uttrakhand, India,
Phone: (+91)01332 285824

training set of previously labeled items. In general, decision trees (DT) (Quinlan, 1986a) and neural networks (NN) (Duda et al., 2001) are two powerful tools for pattern recognition and attracted a lot of interest in last decades. These techniques learn to classify instances by mapping points in the feature space onto classes without explicitly characterizing data in terms of parameterized distributions. Decision trees are easy to understand, but sensitive to noise in feature measurements. In particular, the threshold values used in tests at internal nodes are critical and consequently the sequential hard splitting classification process of a DT can lead to performance degradation. Several neural network models have been successfully applied to various problems in pattern classification, such as multilayer perceptrons (MLPs) using equalized error backpropagation (Martens and Weymaere, 2002), radial basis functions (RBFs) (Krzyzak and Linder, 1998), self-organizing maps (SOMs) (Hsu, 2006) and a number of their variants. However, they suffer of difficult problems to solve, such as the structure and the size of the network (number of neurons and connections involved, number of hidden layers, etc.), computational complexity and convergence analysis.

Neural trees were introduced to combine neural networks and decision trees in order to take advantages of both and to overcome some limitations. According to the existing literature, neural trees can be classified into two categories. The first category uses decision trees to form the structure of the neural network. The main idea is to construct a decision tree and then convert the tree into a neural network. In (Seth, 1990), a three-layered neural network has been proposed by extracting its hidden nodes from a decision tree. Some more details like complexity analysis and practical refinements about this idea have been given in (Brent, 1991). An extension to this idea has been proposed in (Park, 1994), where linear decision trees are used as the building blocks of a network. In (Cios, 1992), a modification of the ID3 algorithm (Quinlan, 1986a) called continuous ID3 algorithm has been proposed. A continuous ID3 algorithm converts decision trees into hidden layers. In the learning process, new hidden layers are added to the network until a learning task becomes linearly separable at the output layer. Thus the algorithm allows self-generation of a feedforward neural network architecture. Cauchy training is used to train the resulting hybrid structure.

The second category uses neural networks as building blocks in decision trees. In (Utgoff, 1989), a hybrid form has been proposed containing neural networks at the leaf nodes and univariate decision nodes as the non-leaf nodes in the tree. In (Sankar and Mammone, 1991), univariate decision nodes have been replaced by single layer perceptrons, i.e., linear multivariate decision nodes at the internal nodes. A new tree pruning algorithm has been proposed for this model based on a Lagrangian cost function (Sankar and Mammone, 1993). The nonlinear multivariate decision tree with MLPs at the internal nodes has been proposed in (Guo and Gelfand, 1992). In (Foresti and Micheloni,

2002), a generalized neural tree (GNT) model has been proposed, where the activation values of each node are normalized so that these can be interpreted as a probability distribution. The main novelty of the GNT consists in the definition of a new training rule that performs an overall optimization of the tree. Each time the tree is increased by a new level, the whole tree is re-evaluated. An adaptive high-order neural tree (AHNT) has been proposed in (Foresti and Dolso, 2004) by composing high order perceptrons (HOP) instead of simple perceptrons in a neural tree model. First-order nodes divide the input space with hyperplanes, while HOPs divide the input space arbitrarily, but at the expense of higher computational cost.

In (Song and Lee, 1998), a structural adaptive intelligent tree, called 'SAINT' has been proposed where the input feature space is hierarchically partitioned by using a tree-structured network that preserves a lattice topology at each subnetwork. Experimental results reveal that 'SAINT' is very effective for the classification of large sets of real words, hand-written characters with high variations, as well as multi-lingual, multi-font, and multi-size large-set characters. In (Yildiz and Alpaydin, 2001), it has been observed that the complexity of the nodes effect the size of tree. A tree with complex nodes may be quite small, while a tree with simple nodes may grow very large. They proposed a new class of DTs where decision nodes can be univariate, linear or nonlinear depending on the outcome of comparative statistical tests. The drawbacks of this method are the long training time and the need of appropriately selecting the parameters for the MLPs.

Recently, a pre-pruning strategy for MLP based tree has been proposed in (Maji, 2008) by defining a uniformity index for modeling the degree of correctness at each node. Due to the uniformity index, it has been shown that a significant reduction in the depth of the tree is achieved with a good classification accuracy. However, no rule has been given to define the values of the parameters like the networks architecture (number of hidden layers and number of nodes for each layer) and the uniformity index for different context as well as each training set. In (Amasyal and Ersoy, 2008), a new algorithm family, called 'Cline' has been proposed by incorporating a number of different methods for building multivariate decision tree. Several algorithms are used at each node and the best one is selected at the respective node. In this way, it searches for the best fitting algorithm at each node/current subspace.

Use of a single layer perceptron is a good choice to avoid the problem of selecting an optimal architecture for multilayer perceptron and the parameters for exhaustive search in heuristic methods. However, a single layer perceptron can get stuck into a local minima or it may keep on stalling in a flat region since it does not have a strong nonlinear function approximation property as in the case of MLPs. This fact generates a large depth tree or a non-converging training process. A solution to this problem has been provided in (Foresti and

Pieroni, 1998) by introducing split nodes. In case of the above situation, the split node simply divides the local training set into two parts by generating a hyperplane passing through the barycenter of the local training set. Such kind of split nodes ensure the convergence of the tree-building process but they are not able to provide any solution in case of large depth trees.

The main novelty of the proposed work includes an implementation of the direct fractional-step discriminant analysis (DF-LDA) (J. Lu, 2003) into the neural tree for obtaining a multi-dimensional split in place of the above described two-dimensional split (Foresti and Pieroni, 1998). The proposed NTLD provides a good solution for convergence of the tree-building process as well as a reduction in the tree-depth. Moreover, there is no need to define a number of parameters such as number of hidden layers, number of nodes for each layer and the uniformity index as in (Maji, 2008), since we are using single layer perceptrons. A regularized linear discriminant analysis is used to handle the situation when the number of training vectors is smaller than the feature dimensionality (i.e., when the with-in-class scatter matrix of the samples is singular). The proposed hybrid classifier has been tested on various benchmark data-sets as well as on synthetically generated four-class chessboard data-set. A comparison study has been provided between the proposed NTLD, NT (Foresti and Pieroni, 1998), DF-LDA , MLP (Hall et al., 2009) and Naive Bayes classifier (Hall et al., 2009) for evaluating the performance over other existing algorithms.

2 Preliminaries

In the proposed work a multi-dimensional split using DF-LDA is introduced to overcome the disadvantages of existing Neural tree algorithms (Sankar and Mammone, 1991),(Foresti and Pieroni, 1998). In this section a brief overview of NT (Sankar and Mammone, 1991) algorithm, splitting nodes (Foresti and Pieroni, 1998) and DF-LDA are given.

2.1 Neural Tree

A neural tree (NT) is a decision tree where each intermediate/non-terminal node is a simple perceptron. Being supervised learning model, the NT accomplishes the task of classification in two phases: (a) training and (b) classification.

Training phase

The NT is constructed by linearly partitioning the training set, consisting of feature vectors and their corresponding class labels, for generating the tree in a recursive manner. Such a procedure involves three steps: computing internal nodes, determining and labeling leaf nodes. An intermediate node groups the input patterns into different classes. When a child node becomes homogeneous i.e., all the patterns in a group belong to the same class, the associated child node to this group is made a leaf node. The related class is used as label for the leaf node. Thus, NTs are class discriminators which recursively partition the training set such that each generated path ends with a leaf node. The training algorithm together with the computation of the tree structure calculates the connection's weights for each node. Connection's weights are adjusted by minimizing an objective function as mean square error or some other error function. At each node the weights corresponding to the optimal value of the objective function are used during the classification of test patterns. The NT training phase is summarized in Algorithm 1. The descriptions about the functions "Train Perceptron" and "Classify_NT" mentioned in the training algorithm are explained below:

Train Perceptron(v, T')

It trains a single layer perceptron at node v on the training set T'. The perceptron learns until the error is not reducing any more for a given number of iterations. A trained perceptron generates $(o_1, ...o_c)$ activation values.

Classify_NT(v, T')

It assigns the pattern to the class corresponding to the highest activation value. In other words, it divides T' into $\hat{T} = \{T_1, ..., T_c\}, \quad c \leq C$ subsets and generates next level of child nodes $\hat{v} = \{v_1, ...v_c\}, \quad c \leq C$ corresponding to $\hat{T}$. Returns $(\hat{v}, \hat{T})$.

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 way. At each node, activation values are computed on the basis of connection's weights. 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

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

Algorithm 1 : Training phase of NT

Set $S_v = \{v_0\}$ and $S_{T'} = \{T\}$
while (S_v) **do**
  $v = Pop\ (S_v)$ and $T' = Pop\ (S_{T'})$
  Train Perceptron(v, T')
  $(S_{\hat{v}}, S_{\hat{T}})$=Classify_NT (v, T')
  while $S_{\hat{T}}$ **do**
    if $\tilde{T} = Pop\ (S_{\hat{T}})$ is homogeneous **then**
      $\tilde{v} = Pop\ (S_{\hat{v}})$ is set to leaf
    else
      $Push\ (S_v, Pop\ (S_{\hat{v}}))$ and $Push\ (S_{T'}, \tilde{T})$
    end if
  end while
end while

where σ is the sigmoidal activation function, $\mathbf{w_i}$ is the vector of the weights for the connections from the inputs to the i^{th} output, and $\mathbf{x}$ is the input pattern. This rule determines the class associated with a leaf node as well as the next internal node to be considered on the path to reach a leaf node. The classification phase is summarized in Algorithm 2. The symbols used in the

Algorithm 2 : Classification Phase of NT

Set $v = v_0$
while $(v) \neq leafnode$ **do**
  Input $\mathbf{x}$ to v
  Set $v = v_i$ corresponding to $arg\ max\ i\{o_1, ..., o_C\}$
end while
Assign $\mathbf{x}$ to c_i corresponding to v

above Algorithms 1 and 2 are explained in the following table:

Table 1
Symbols used and their meaning.

symbols	stands for
v_o	root node of the tree
T	training set (TS) at the root node
v	an internal node which is being trained currently by the algorithm
T'	local training set (LTS)† at node v
S_v	stack holding the nodes to be trained
$S_{T'}$	stack holding the LTSs corresponding to nodes in S_v
$\hat{v}$	next level of child nodes resulted by the split of node v
$\hat{T}$	next level of LTSs corresponding to $\hat{v}$
$S_{\hat{v}}$	local stack holding $\hat{v}$
$S_{\hat{T}}$	local stack holding $\hat{T}$
C	number of classes (output)
$\{o_1, \ldots, o_c\}$	probabilities of a pattern belonging to each class
†LTS is a subset of training set obtained by splitting at a node.	

2.2 Split node

A single layer perceptron is simple to train but sometimes, when the TS is non linearly separable, the training process falls into a local minimum or stalls in a flat region. In such a case, the neural tree using single layer perceptron to learn and split the training set may not generate a splitting rule and keep on passing the same local training set to the successive child nodes hence leading to non convergent tree building process. To handle such a situation Foresti and Pieroni (Foresti and Pieroni, 1998) employed an univariate split node when perceptron fails to split the training set. This particular splitting rule at first computes the barycentres of two classes with higher cardinality and the component k which satisfy L_1 norm is identified. Then it computes the splitting hyperplane orthogonal to the k axis and passing through the median point between the two barycentres (see Fig. 1). This kind of split node guarantees the convergence of tree building process but the two-dimensional split may generate deeper tree resulting in overfitting and hence degraded performance. A tree building process using split nodes has been demonstrated in Fig. 2 for a five-class data-set. The split node each time chooses the two dominating classes and classifies the patterns of other classes also as belonging to these two classes. In this way the work done by perceptron is lost and the tree building process may generate a large number of nodes before reaching convergence.

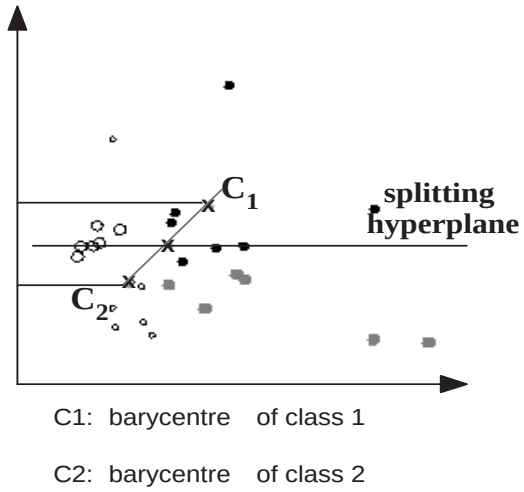

Fig. 1. Splitting hyperplane generated by split node (Foresti and Pieroni, 1998)

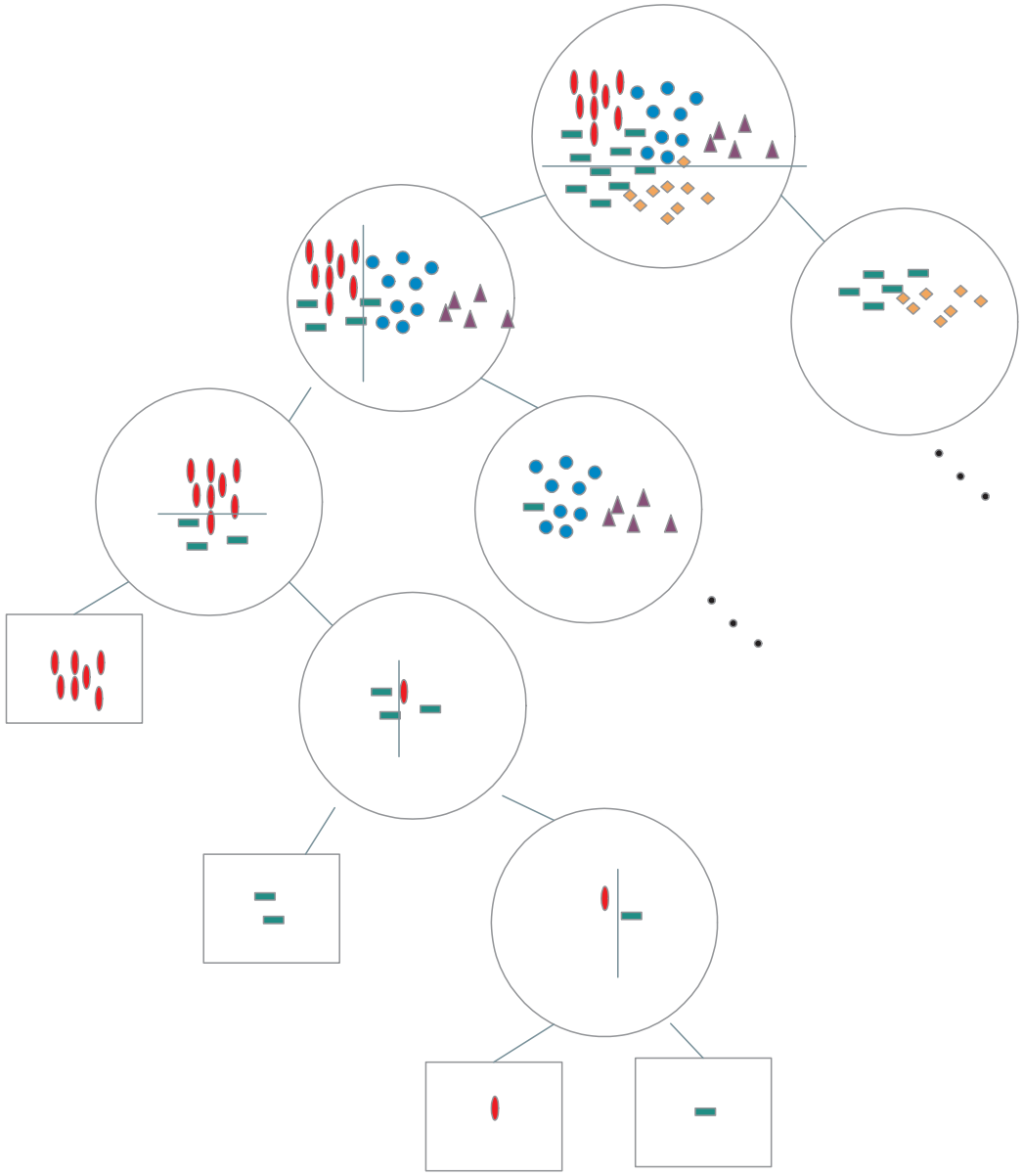

Fig. 2. A tree building process using split nodes for a five-class data-set. A large number of nodes are required in order to make the data linearly separable.

2.3 Linear Discriminant Analysis

A LDA node searches for those vectors in the underlying space that best discriminate among classes (rather than those that best describe the data). Then the objective of LDA is to seek the direction Ψ not only maximizing the between-class scatter of the projected samples, but also minimizing the with-in-class scatter. These two objectives can be achieved simultaneously by

maximizing the following function:

$$J(\Psi) = \frac{\Psi^T \mathbf{S}_B \Psi}{\Psi^T \mathbf{S}_W \Psi} \tag{2}$$

where the between-class scatter matrix $\mathbf{S}_B$ is defined as

$$\mathbf{S}_B = \sum_{i=1}^{C} n_i (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T \tag{3}$$

in which $\mathbf{m}$ is the k-dimensional sample mean for the whole set, while $\mathbf{m}_i$ is the sample mean for i^{th} class. The with-in-class scatter matrix $\mathbf{S}_W$ is defined by

$$\mathbf{S}_W = \sum_{i=1}^{C} \mathbf{S}_i \tag{4}$$

where the scatter matrix $\mathbf{S}_i$ corresponding to i^{th} class is defined by

$$\mathbf{S}_i = \sum_{\mathbf{x} \in D_i} (\mathbf{x} - \mathbf{m_i})(\mathbf{x} - \mathbf{m_i})^T, \quad i = 1, ...C. \tag{5}$$

However, the traditional LDA suffers two problems 1) the degenerated scatter matrices caused by the so-called small sample size (SSS) problem, 2) the classification performance is often degraded by the fact that their separability criteria are not directly related to their classification accuracy in the output space (Lotlikar and Kothar, 2000). To avoid such problems a regularized version of LDA i. e. "direct fractional-step linear discriminant analysis" (DF-LDA) is used. The method DF-LDA uses variants of "direct-linear discriminant analysis" (D-LDA) (Chen et al., 2000) and fractional step- discriminant analysis (F-LDA) (Lotlikar and Kothar, 2000). The traditional solution to the SSS problem requires the incorporation of a principal component analysis (PCA) step into the LDA framework, which may cause loss of significant discriminatory information. In the D-LDA framework, data are processed directly in the original high-dimensional input space avoiding the loss of significant discriminatory information due to the PCA pre-processing step. A solution to the second problem is introducing weight functions in LDA. Object classes that are closer together in the output space, and thus can potentially result in misclassification, should be more heavily weighted in the input space. This idea has been framed in (Lotlikar and Kothar, 2000) with the introduction of the fractional-step linear discriminant analysis algorithm (F-LDA), where the dimensionality reduction is implemented in a few small fractional steps allowing for the relevant distances to be more accurately weighted.

3 Neural Tree with Linear Discriminant Analysis (LDNT)

The proposed NTLD classifier is basically a decision tree whose nodes are either single layer perceptrons without hidden nodes or split nodes using DF-LDA. The adopted perceptron takes k inputs and generates C outputs, called activation values. Winner-takes-all rule is applied and the class with the highest associated activation value is taken to be the winner class. Each perceptron is characterized by a sigmoid activation function $\sigma(\mathbf{x}) = 1/(1 + e^{-\mathbf{x}})$. Thus the activation value o_i^q of the q^{th} pattern corresponding to the class c_i is given by:

$$o_i^q = 1/[1 + exp(-\sum_{j=1}^{k} w_{ij} x_j^q)] \tag{6}$$

where, w_{ij} are the elements of the weight matrix $\mathbf{W}$ of the perceptron. A DF-LDA node projects the patterns into a new space in order to bring the patterns of same class nearer and of different classes farther. The Euclidean distance of a pattern is computed from the centroid of each class in the projected space. A pattern is classified as belonging to the class whose centroid is closest to the projected pattern. The detailed descriptions of the adopted training and testing strategies are given in the following subsections.

3.1 Training Phase of NTLD Classifier

Let $T = \{(\mathbf{x_j}, c_i)|j = 1, \ldots, n \wedge c_i \in [1, C]\}$ be the training set containing n number of k-dimensional $\mathbf{x_j} = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3} \ldots, \mathbf{x_k}\}$ patterns, each of them belonging to one of the C classes. The training phase of the NTLD is described in the Algorithm 3.The notations used in Algorithm 3 have already been given in Section 2. The algorithm starts with a simple perceptron at root node (v_0) and whole training set (T) as input to the root node. The stacks S_v and $S_{T'}$ initially holds only v_0 and T respectively. The last elements of S_v and $S_{T'}$ are popped into v and T' and processed i.e. a perceptron is trained at the node v with input as T'. The trained perceptron at node v splits the LTS T' into further LTSs $\hat{T} = \{T_1, ..., T_c\}, \quad c \leq C$ generating next level of child nodes $\hat{v} = \{v_1, ...v_c\}, \quad c \leq C$ which are held in the local stacks $S_{\hat{T}}$ and $S_{\hat{v}}$ respectively. If the local stack $|S_{\hat{T}}| = 1$, i.e., the perceptron is unable to split the LTS and passes it to the child node as it is, such perceptron is replaced with DF-LDA at this node. A DF-LDA is trained at node v for LTS T' to split it into further LTSs $\hat{T} = \{T_1, ..., T_c\}, \quad c \leq C$ and corresponding child nodes $\hat{v} = \{v_1, ...v_c\}, \quad c \leq C$. Each child node is popped into $\hat{v}$ and corresponding LTS into $\hat{T}$ from the local stacks $S_{\hat{v}}$ and $S_{\hat{T}}$ and checked if $\hat{T}$ is homogeneous i. e. all the patterns of $\hat{T}$ belongs to the same class. If yes then this node is

marked as leaf node and labeled with the class of related patterns if not the node $\hat{v}$ is pushed into the stack S_v and the corresponding LTS $\hat{T}$ is pushed into the stack $S_{S'}$. The algorithm runs until the stack S_v counts to 0, i.e., all the nodes becomes leaf nodes. Descriptions about the functions "Train LDA" and "Classify_LD" mentioned in Algorithm 3 are explained below:

Algorithm 3 : Training phase of NTLD classifier

```
Set S_v = {v_0} and S_T' = {T}
while (S_v) do
  v = Pop (S_v) and T' = Pop (S_T')
  Train Perceptron(v, T')
  (S_v̂, S_T̂)=Classify_NT (v, T')
  if |S_T̂| = 1 then
    Train LDA(v, T')
    (S_v̂, S_T̂)=Classify_LD (v, T')
  end if
  while S_T̂ do
    if T̃ = Pop (S_T̂) is homogeneous then
      ṽ = Pop (S_v̂) is set to leaf
    else
      Push (S_v, Pop (S_v̂) and Push (S_T', T̃)
    end if
  end while
end while
```

Train LDA(v, T')

It employs DF-LDA to project the patterns into a new space such that the ratio $J(\Psi)$ is maximized. After finding such a projection matrix, T' is projected into the new space and the centroid of the classes are computed.

Classify_LD (v, T')

Each pattern $\mathbf{x_j} \in T'$ is projected in the new space. The Euclidean distances $(d_1, \ldots, d_c)$ of the projected pattern with respect to classes centroid are computed. The pattern $\mathbf{x_j}$ is assigned to the class corresponding to the minimum distance. Thus it divides T' into $\hat{T} = \{T_1, ..., T_c\}, \quad c \leq C$ subsets and generates next level of child nodes $\hat{v} = \{v_1, ...v_c\}, \quad c \leq C$ corresponding to $\hat{T}$. Returns $(\hat{v}, \hat{T})$.

3.2 Classification Phase of NTLD Classifier

The classification phase of NTLD is inherited from NT. The same top-down traversal technique is adopted. The pattern moves through the tree starting from the root node and adopting the path suggested by the classification given by each node (maximum activation value in case of perceptron/minimum Euclidean distance in case of DF-LDA) (see Fig. 3). When a leaf node is reached, the pattern is classified corresponding to the label of this node. The steps involved in the classification of a pattern $\mathbf{x} = (x^1,\ x^2,\ \ldots,\ x^k)$ at the current node v are given in the Algorithm 4.

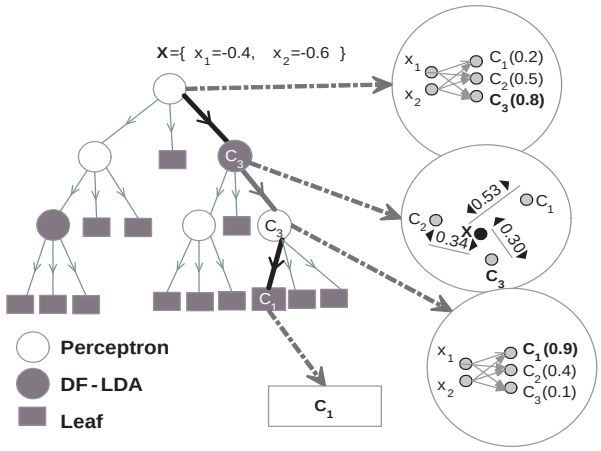

Fig. 3. Classification example for a three class problem, the pattern to be classified follows the path suggested by current node (perceptron/DF-LDA) to reach a leaf node.

Algorithm 4 : Classification phase of NTLD classifier

```
Set v = v_0
while (v) ≠ leafnode do
  Input x to v
  if v=Perceptron node then
    Set v = v_i corresponding to arg max_{i=1,...,C} o_i
  end if
  if v=LDA node then
    Set v = v_i corresponding to arg min_{i=1,...,C} d_i
  end if
end while
Assign x to c_i corresponding to v
```

Using the above described steps, all patterns from a testing set can be classified into their corresponding classes.

4 Results and Discussions

The performance of the proposed NTLD classifier has been evaluated in the classification of synthetic as well as various real benchmark datasets. The achieved classification accuracy, the size of the tree and the time taken to classify test set have been used as the measures for performance evaluation. The classification accuracy has been measured in terms of the percentage of correctly classified patterns, and the size of the tree has been measured in terms of its depth and number of nodes. The selection of learning rate has been done by validating the training data set on a number of runs. The weights have been initialized randomly for a perceptron at each node in the tree. A comparison study has been done of the proposed NTLD classifier with some well known existing classification techniques in terms of correct classification with statistical analysis.

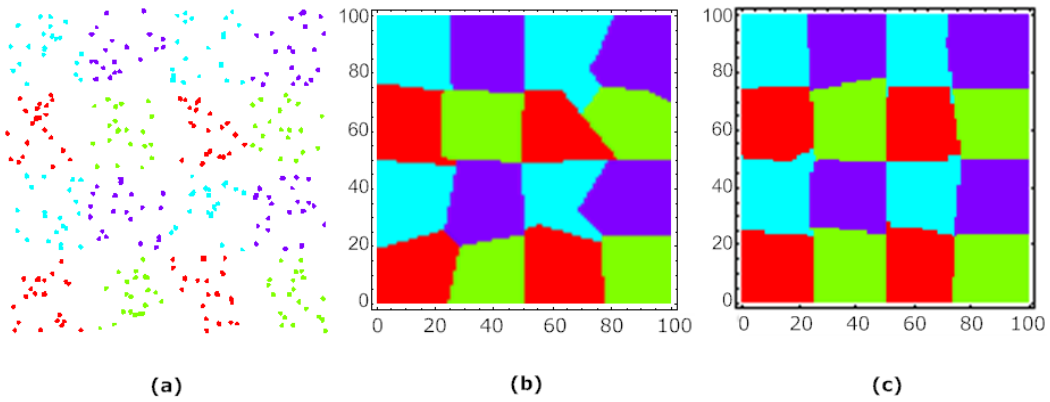

Fig. 4. (a) 336 random points distributed equally among 4 classes. (b) classification done by NT (Foresti and Pieroni, 1998) (c) classification done by NTLD.

4.1 Description of datasets used in classification

The synthetic dataset named "four-class chessboard" has been used for the classification. It consists with a geometric distribution of planar points among four classes inside a square, where 336 randomly drawn points distributed equally as shown in the Fig. 4(a). We have taken twelve different real datasets from UCI machine learning repository (Frank and Asuncion , 2010) for evaluating the performance of the proposed classifier. All these datasets contain real valued attributes and multiple output classes (varying from two to twenty six) summarized in Table 2. A more detailed information of these real datasets are available at UCI machine learning repository.

Table 2
Description of datasets taken from (Frank and Asuncion , 2010).

Dataset	No. of patterns	Attributes per pattern	No. of classes	Type of attributes
Satellite	6435	36	7	numeric
Letter	20000	16	26	numeric
DNA	3186	60	3	nominal
Segment	2310	19	7	numeric
Waveform	5000	40	3	numeric
Breast-W	799	9	2	numeric
Diabetes	768	8	2	numeric
E.coli	314	7	8	numeric
Ionosphere	350	34	2	numeric
Dermatology	336	35	7	numeric/nominal
Heberman	306	3	2	numeric
Iris	150	4	3	numeric

Table 3
Mean Classification accuracy of different algorithms for ten fold cross validation for datasets taken from (Frank and Asuncion , 2010). The mean classification accuracy is simple classification accuracy for Satellite and Letter datasets.

Data set	NTLD	NT	DF-LDA	Naive Bayes	MLP (1-h)	MLP (2-h)	MLP (3-h)
Satellite*	87.95	82.01	82.60	79.58	87.4	89.45[1]	88.75
Letter*	88.50[1]	79.03	68.82	62.3	80.97	83.68	83.13
DNA	94.31	91.16	90.52	95.29[1]	91.4	91.88	93.82
Segment	94.76	90.47	80.51	80.21	96.06	96.75[1]	96.66
Waveform	82.79	79.11	81.4	80	83.56[1]	83.74	83.34
Breast-W	95.56[1]	94.86	94.23	95.49	95.27	95.42	95.42
Diabetes	68.92	67.20	63.14	76.3	75.39[1]	75.13	73.82
E.coli	82.4	82.42	74.33	85.41	85.71[1]	84.22	83.33
Ionosphere	91.16	91.16	62	82.62	91.16	91.73	91.79[1]
Dermatology	94.44	87.02	84.94	97.06[1]	96.72	95.9	95.62
Heberman	62.58	60.61	68.9	76.14[1]	69	73	70.91
Iris	97.99[1]	96.07	92.5	96	97.33	96.66	96

4.2 Classification accuracy

The qualitative results obtained with NTLD for the synthetically generated four-classes chessboard dataset are shown in Fig. 4(c). A comparison has been made with the result achieved with neural tree (NT) (Foresti and Pieroni, 1998) in Fig. 4(b), where a binary split along with single layer perceptron have been used. For a quantitative evaluation, the patterns from the four-classes chessboard dataset are generated five times randomly, and the tree building and classification process is performed every time. The average classification accuracies obtained with NT and NTLD was 92.12 and 95.29, respectively.

Table 4
Standard Deviation of the classification results obtained in ten-fold cross validation.

Datasets	NTLD	SD NT	MLP (two hidden layers)
DNA	0.57	0.95	0.64
Segment	1.03	1.78	1.01
Waveform	0.81	1.95	0.75
Breast-W	0.73	2.04	0.75
Diabetes	0.96	1.07	0.89
E.coli	1.30	1.28	1.30
Ionosphere	1.09	1.09	1.01
Dermatology	0.85	1.23	0.83
Heberman	2.10	2.21	1.96
Iris	0.76	1.09	0.89

The average size of the tree obtained with NT algorithm was 51 nodes with depth 9, whereas it was 29 nodes with depth 6 with the proposed NTLD algorithm. It is worth to notice that the proposed classifier performs better in terms of classification accuracy as well as in size of the tree.

In classification of real datasets, the results obtained with NTLD have been compared with other four different existing classification methods. These methods include a binary split based NT (Foresti and Pieroni, 1998), DF-LDA (J. Lu, 2003), multilayer perceptron (MLP) (Hall et al., 2009) and Naive Bayes (John and Langley, 1995). Here, Weka version-3.6 (Hall et al., 2009) software has been used for obtaining results with MLP and Naive Bayes classifiers. Three different network architectures of MLP have been used having one, two and three hidden layers, respectively. In each hidden layer, the number of nodes have been decided as the average of input and output's dimensions. Learning rate has been chosen 0.30 and the maximum number of epochs has been fixed at 500 for all the experiments. A ten-fold cross validation has been adopted, i.e, the data set is divided into ten equal parts and each time nine out of ten have been used for training while the left out is used for testing. This ten-fold cross validation has been done for all datasets except satellite and letter as the training and testing data for these two datasets are available separately at the UCI repository. The obtained results (mean classification accuracy) on these datasets with the proposed NTLD, NT, DF-LDA, Naive Bayes and MLP have been shown in Table 3. The standard deviation of the classification accuracies obtained in ten runs have been listed in Table 4. Here, results of standard deviation has been given with NTLD, NT and MLP classifiers only as these three belong to same category. i.e., perceptron based classifier. A graphical representation of standard deviation results has been shown in Fig. 5 in terms of box-plot. In box-plot, horizontal axis represents the datasets (in the same order as in Table 4). The bottom and top of each box represent the 25^{th} and 75^{th} percentiles (the lower and upper quartiles, respectively), and the band near the middle of the box is the 50^{th} percentile (the median).

4.3 Statistical analysis

A detailed statistical analysis has been performed to analyze whether the results obtained with NTLD classifier are significantly better than other algorithms. To analyze this, two tailed F-test and two tailed t-test have been performed. Here these test have been performed for two different pairs of algorithms. The NTLD and NT have been considered as first pair, while combination of NTLD and MLP have been taken as the another pair.

Two tailed F-test has been performed at 5% level of significance for testing the equality of variances of the results (in ten-fold cross validation) with these two pairs of algorithms. The calculated value of $F - statistics$ have been listed in the Table 5. It has been observed that all the calculated $F - statistics$ values are in the range of two tailed $F - critical$ values except the two datasets (Waveform and Breast-W). Hence, null hypothesis H_0 of population variances, i.e., of equal variances may be accepted in all cases apart from these two datasets. In case of NTLD and MLP pair, all the calculated values of $F - statistics$ have been found in the range of critical F-values. Hence, in this case the hypothesis of equal variance are accepted for all the datasets.

Table 5
Statistical analysis (F-test) for pair of algorithms

Data set	Comparison of NTLD and NT			Comparison of NTLD and MLP		
	H	$F - stat$	DOF	H	$F - stat$	DOF
DNA	0	0.3600	(9,9)	0	0.7932	(9,9)
Segment	0	0.3348	(9,9)	0	1.0400	(9,9)
Waveform	1	0.1725	(9,9)	0	1.1604	(9,9)
Breast-W	1	0.1281	(9,9)	0	0.9474	(9,9)
Diabetes	0	0.8050	(9,9)	0	1.1635	(9,9)
E.coli	0	1.0315	(9,9)	0	1.0000	(9,9)
Ionosphere	0	1.0000	(9,9)	0	1.1647	(9,9)
Dermatology	0	0.4776	(9,9)	0	1.0488	(9,9)
Heberman	0	0.9029	(9,9)	0	1.1480	(9,9)
Iris	0	0.4862	(9,9)	0	0.7292	(9,9)
$F - critical$=(0.2483, 4.02599)						

Now a two tailed t-test with equal variances has been performed at 5% level of significance. These results have been shown in Table 6. It can be observed that absolute value of t-stat is much greater than the t-critical value in comparison of NTLD and NT for six datasets (DNA, Segment, Waveform, Diabetes, Dermatology, Iris). Thus it is highly significant and null hypothesis, i.e., mean of two algorithm are identical is rejected. Hence the two types of means differ significantly. Further, since the mean classification accuracy of NTLD is higher than NT, we conclude that NTLD is definitely better than NT and this difference is statistically significant. Where, the null hypothesis is accepted at 5% level of significance, the mean classification accuracy is more, in general,

Table 6
Statistical analysis (t-test) for pair of algorithms

Data set	Comparison of NTLD and NT				Comparison of NTLD and MLP			
	H	$\|t-stat\|$	DOF	s_p^2	H	$\|t-stat\|$	DOF	s_p^2
DNA	1	8.9912	18	0.6137	1	9.2245	18	0.3673
Segment	1	6.5966	18	2.1146	1	4.3623	18	1.0405
Waveform	1	5.5112	12	2.2293	1	2.7214	18	0.6093
Breast-W	0	1.0217	12	2.3473	0	0.4230	18	0.5477
Diabetes	1	3.7836	18	1.0333	1	15.0011	18	0.8569
E.coli	0	0.0173	18	1.6642	0	3.0789	18	1.6900
Ionosphere	0	0.0000	18	1.1881	1	1.2130	18	1.1041
Dermatology	1	15.6937	18	1.1177	0	3.8862	18	0.7057
Heberman	0	2.0434	18	4.6471	1	11.4709	18	4.1258
Iris	1	4.5692	18	0.8829	1	3.5937	18	0.6849
DOF: degree of freedom; S_p^2: pooled variance; t-critical=2.1009								

we can conclude that the NTLD is better than NT algorithm.

In case of NTLD and MLP comparison, it can be seen from the table that NTLD algorithm is better that MLP in classification of DNA and Iris datasets at 5% level of significance. Whereas, The null hypothesis of equal mean can be accepted in classification of three datasets (Breast-W, E.coli and Dermatology). In other cases, MLP is better than NTLD statistically. However, the NTLD is having an advantage over MLP that there is no need to select network architecture for getting optimal performance.

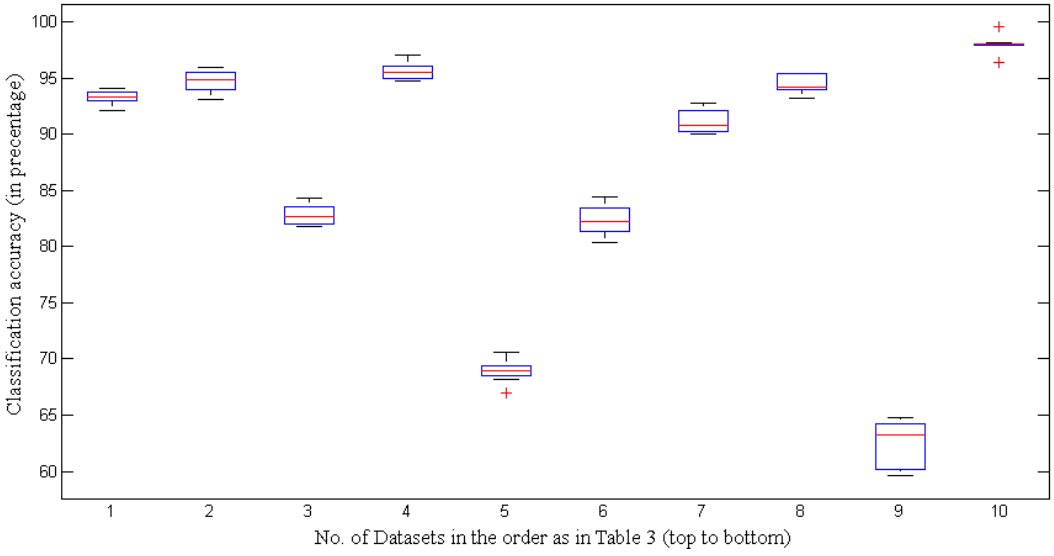

Fig. 5. Box plot for statistical analysis of the classification results obtained in ten–fold cross validation.

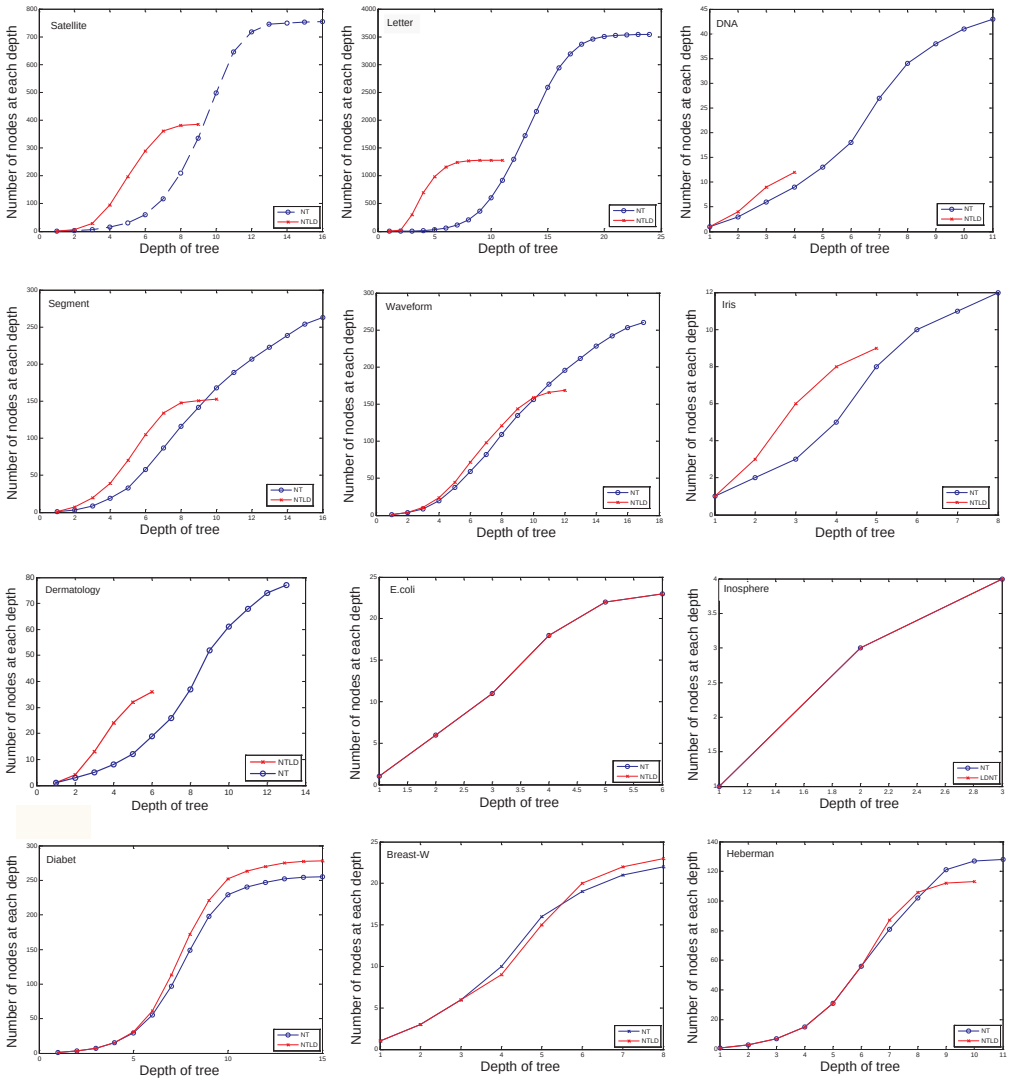

Fig. 6. Graphs drawn between depth of tree and number of nodes till respective depth

4.4 Complexity analysis

To discuss the complexity of NTLD and NT classification algorithms, a comparison study in terms of tree size has been done (See Table 7). The size, i.e, number of nodes and depth of tree is averaged for ten runs. It is observed that NTLD converges at shorter depths than NT with lower number of nodes. The graphs shown in Fig. 6 are drawn to highlight the fast converging behavior of NTLD over NT. These graphs are drawn between depth of tree and number of nodes generated till each depth. It is clear from the graphs that NTLD grows rapidly i.e., it grows with more number of nodes at each depth due to multidimensional split. Also it converges faster due to good discrimination properties of DF-LDA and Perceptron. On the other hand graph of NT is

Table 7
Comparison of depth, number of nodes and number of DF-LDA/Split nodes for NTLD and NT (Foresti and Pieroni, 1998).

Data set	NTLD	NT
Satellite	9,386,338	17,752,704
Letter	11,1281,1100	24,3435,3310
DNA	4.1,14.4,4	11,42,6
Segment	10,169.6,135.3	16,260,215.3
Waveform	12.6,177.5,50.8	17.1,220.4,59.9
Breast-W	8.4,22.1,8.9	8.9,23.8,9.2
Diabetes	15.1,246.2,66.7	15.4,252.6,71.8
E.coli	6,23,0	6,23,0
Ionosphere	3,4,0	3,4,0
Dermatology	6.3,32.3,23.5	12.3,68.2,56.7
Heberman	10,102.5,66.4	11,106.3,74
Iris	5.6,9.1,3.8	7.5,11.7,4.5

flatter as compared to NTLD i.e, it grows with less number of nodes at each depth and converges at more depth as the work done by perceptron is lost when it is replaced by a split node.

The graphs of NTLD for multi-class data-sets like Satellite data-set, Letter data-set, DNA data-set, Segment data-set, Waveform data-set, Iris data-set and Dermatology data-set grows rapidly than NT, as a result of exploiting the multidimensional split done by DF-LDA and perceptron. Whereas graphs of NTLD for two-class data-sets like Diabetes data-set, Breast-w data-set and Heberman data-set are almost same like NT as the tree remains binary in both algorithms. Moreover for a two class problem DF-LDA can choose only one discriminating feature as the number of discriminating features chosen by DF-LDA is $C-1$, as like split rule (Foresti and Pieroni, 1998) does. The NTLD and NT grows only with perceptron nodes for E.coli data-set and Ionosphere data-set i.e., a perceptron is always able to separate the training set so a need for DF-LDA/split node does not arise. Hence the graphs of NTLD and NT for these two data-sets are similar.

A comparison of time taken by the trained NTLD and NT classifier to classify the test patterns is also made. Since the classification time may 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 modeled 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. More deep the tree is, more time a pattern takes to be classified (reach leaf node). The path length (number of nodes to reach a leaf node) has been computed for each pattern and then the average path length of all the patterns has been used as a parameter for the comparison between NTLD and NT (see Table 8). The average path length is defined as: Avg path length= sum of path lengths traversed by each pattern/total number of patterns. From Table 8 it is apparent that most of the time average path length of NTLD is

smaller than NT. It is more significant in case of complex and larger datasets such as letter, satellite, DNA, segment and waveform as it may reduce the classification time by good amount if each pattern traverse smaller number of nodes.

Table 8
Comparison of average path length (average number of nodes taken by each pattern to reach a leaf node) for NTLD and NT.

Data set	NTLD (Avg. path length)	NT (Avg. path length)
Satellite	4.49	8.77
Letter	4.16	13.10
DNA	3.11	4.65
Segment	3.92	7.73
Waveform	5.81	6.31
Breast-W	2.80	2.82
Diabetes	8.21	8.26
E.coli	2.78	2.78
Ionosphere	1.87	1.87
Dermatology	2.67	4.06
Heberman	5.05	5.15
Iris	2.90	3.06

5 Conclusions

We have presented a hybrid classifier composed by simple perceptrons and linear discriminant classifiers in a tree structure. The main novelty in the proposed classifier is the adoption of multi-dimensional split using DF-LDA when perceptron is not efficient in classifying patterns. We have tested the proposed classifier on various data-sets and derived the following remarks:

(1) The proposed NTLD classifier gives a good classification accuracy in case of multi-class problem (almost always the best except MLP in case of few data-sets). It performs almost similar to NT in case of two-class problems.
(2) NTLD generates shallower tree than NT in case of multi-class problem resulting in a faster classification.
(3) NTLD does not require the ad-hoc parameters like details about network architecture (number of hidden layers and nodes in each layer) as in case of MLP. Only one parameter i.e., learning rate is chosen using validation set.

The proposed classifier is easy to implement and adopts the good properties of neural network as well as linear discriminant classifiers. It is more accurate in case of complex classification problems where number of classes is large.

Acknowledgement

This work was partially supported by the Italian Ministry of University and Scientific Research (MIUR). The second author gratefully acknowledges the support of IIT Roorkee for carrying out this work.

References

Amasyal, M. F., Ersoy, O., Feb 2008. Cline: A new decision-tree family. IEEE Transactions on Neural Networks 19 (2), 356–363.

Frank, A. and Asuncion, 2010. UCI Machine Learning Repository *[http://archive.ics.uci.edu/ml]*. School of Information and Computer Science, University of California, Irvine, CA.

Brent, R. P., 1991. Fast training algorithms for multilayer neural nets. IEEE Transactions on Neural Networks 2, 346–354.

Chen, L. F., Liao, H. Y. M., Ko, M., Lin, J. C., Yu, G. J., 2000. A new lda-based face recognition system which can solve the small sample size problem. Pattern Recognition 33, 1713-1726.

Cios, K. J., 1992. A machine learning method for generation of a neural-network architecture: A continuous id3 algorithm. IEEE Transaction on Neural Networks 3, 280-291.

Duda, R. O., Hart, P. E., Stork, D. G., 2001. Pattern Classification. New York: John Wiley & Sons.

Foresti, G. L., Dolso, T., 2004. An adaptive high-order neural tree for pattern recognition. IEEE Transactions on Systemas, Man, Cybernatics- part B: Cybernatics 34(2), 988–996.

Foresti, G. L., Micheloni, C., Nov. 2002. Generalized neural trees for pattern classification. IEEE Transactions on Neural Networks 13, 1540–1547.

Foresti, G. L., Pieroni, G., 1998. Exploiting neural trees in range image understanding. Pattern Recognition Letters 19, 869–878.

Guo, H., Gelfand, S. B., 1992. Classification trees with neural-network feature extraction. IEEE Transactions on Neural Networks 3, 923–933.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., H, I., 2009. The WEKA Data Mining Software: An Update. Vol. 11 of 1. SIGKDD Explorations.

Hsu, C. C., 2006. Generalizing self-organizing map for categorical data. IEEE Transactions on Neural Networks 17 (2), 294–304.

Lu, J., Plataniotis, K. N., Venetsanopoulos, A. N., 2003. Face recognition using lda-based algorithms. IEEE Transactions on Neural Networks 14(1), 195–200.

John, G. H., Langley, P., 1995. Estimating continuous distributions in bayesian

classifiers. In: Eleventh Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann, San Mateo, pp. 338–345.

Krzyzak, A., Linder, T., 1998. Radial basis function networks and complexity regularization in function learning,. IEEE Transactions on Neural Networks 9, 247–256.

Lotlikar, R., Kothar, R., 2000. Fractional-step dimensionality reduction. IEEE Transactions on Pattern Analysis and Machine Intelligence 22, 623–627.

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

Martens, J. P., Weymaere, N., 2002. An equalized error backpropagation algorithm for the on-line training of multilayer perceptrons. IEEE Transactions on Neural Networks 13(3), 532–541.

Park, Y., 1994. A comparison of neural-net classifiers and linear tree classifiers: Their similarities and differences. Pattern Recognition 27, 14931503.

Quinlan, J. R., 1986. Induction of decision trees. In: Mach. Learn. Vol. 1. p. 81-106.

Sankar, A., Mammone, R. J., July 1991. Optimal pruning of neural tree networks for improved generalization. In: In Proc. Int. Joint Conf. Neural Networks. Seattle, WA, p. 219-224.

Sankar, A., Mammone, R. J., 1993. Growing and pruning neural tree networks. IEEE Transactions on Computers 42, 291–299.

Seth, I. K., 1990. Entropy nets: From decision trees to neural networks. In: IEEE. Vol. 78. p. 1605-1613.

Song, H. H., Lee, S., May 1998. A self-organizing neural tree for large set pattern classification. IEEE Transactions on Neural Networks 9, 369–380.

Utgoff, P. E., 1989. Perceptron trees: A case study in hybrid concept representations. Connection Sci. 1, 377–391.

Yildiz, O. T., Alpaydin, E., Nov. 2001. Omnivariate decision trees. IEEE Transactions on Neural Networks 12, 1539–1546.