Pattern Recognition 48 (2015) 976–983

Contents lists available at ScienceDirect

Pattern Recognition

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

Disjunctive normal random forests

Mojtaba Seyedhosseini [a,b,*], Tolga Tasdizen [a,b]

[a] Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT 84112, USA
[b] Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112, USA

ARTICLE INFO

Article history:
Received 17 January 2014
Received in revised form
1 July 2014
Accepted 23 August 2014
Available online 4 September 2014

Keywords:
Random forest
Decision tree
Classifier
Supervised learning
Disjunctive normal form

ABSTRACT

We develop a novel supervised learning/classification method, called *disjunctive normal random forest* (DNRF). A DNRF is an ensemble of randomly trained *disjunctive normal decision trees* (DNDT). To construct a DNDT, we formulate each decision tree in the random forest as a disjunction of rules, which are conjunctions of Boolean functions. We then approximate this disjunction of conjunctions with a differentiable function and approach the learning process as a risk minimization problem that incorporates the classification error into a single global objective function. The minimization problem is solved using gradient descent. DNRFs are able to learn complex decision boundaries and achieve low generalization error. We present experimental results demonstrating the improved performance of DNDTs and DNRFs over conventional decision trees and random forests. We also show the superior performance of DNRFs over state-of-the-art classification methods on benchmark datasets.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Random forests became popular with Breiman's seminal paper [1] in 2001 due to their ease of use and good classification accuracy. The main idea of random forest classification is to grow an ensemble of decision trees such that the correlation between the trees remains as low as possible. This is achieved by injecting randomness into the forest using a different set of training samples for each tree. These sets are obtained by sampling the original training set with replacement, *i.e.*, bagging. Another source of randomness in random forests is the subset of features randomly selected to consider at each node as the splitting function. This parameter can directly control the correlation between the trees and also affect the accuracy performance of each individual tree. At test time, each tree in the random forest casts a unique vote for the given input and the most popular class among the trees is selected as the predicted label for that input. Random forests have been shown to be effective in many applications like image segmentation/classification [2,3], object detection [4], and biomedical image analysis [5,6].

Random forests have certain advantages over other widely used classification algorithms. For instance, support vector machines (SVMs) [7] offer good generalization performance due to the fact that they guarantee maximum margin, but choosing the kernel function and the kernel parameters can be time consuming. Boosting [8] is another popular classification approach, which trains a single strong classifier by combining multiple weak classifiers. However, convergence of the learning algorithm can be slow for problems with complex decision boundaries. Artificial neural networks (ANNs) [9] are powerful but slow at training due to the computational cost of backpropagation [10]. In addition to all the aforementioned shortcomings of ANNs, SVMs, and boosting methods, these techniques do not naturally handle multi-class problems [11–13]. On the other hand, random forests are fast to train and handle multi-class problems intrinsically [14]. Moreover, they perform consistently well for high dimensional problems [15].

The weak learner used at each node of the decision trees plays an important role in the behavior and performance of random forests. The conventional random forest exploits axis-aligned decision stumps, which partition the feature space with orthogonal hyperplanes. While this type of partitioning can be suitable for certain types of datasets, it results in overfitting and produces "blocky artifacts" in general datasets [14]. It has been shown that using linear discriminants that can be at any arbitrary orientation to the axes improves the performance of random forests [16]. Nonlinear weak learners like conic sections have also been proved successful in increasing the accuracy and generalization performance of random forests [14].

A lot of work has been put into improving the random forest, through the use of more powerful node models and less correlated trees. Rodriguez et al. [17] used PCA to make a linear combination of features at each node. Bernard et al. [18] focused on the number of features randomly selected at each node of the tree. They showed that

* Corresponding author at: Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112, USA. Tel.: +1 801 585 1867; fax: +1 801 585 6513.
E-mail address: mseyed@sci.utah.edu (M. Seyedhosseini).

http://dx.doi.org/10.1016/j.patcog.2014.08.023
0031-3203/© 2014 Elsevier Ltd. All rights reserved.

using a random number of features, which can be different at each node, can improve the performance. Tripoliti et al. [19] improved the prediction peformance of random forests by modifying the node split function as well as the voting procedure in the forest.

In this paper, we propose a novel approach for learning linear discriminants of arbitrary orientation at each node of a decision tree. However, the main advantage of our approach over the above-mentioned methods such as [16,17] is that it learns all the weak learners of the decision tree in a unified framework. To be clear, unlike conventional decision trees and their variants that learn the splitting function at each node independently, our approach allows weak learners of different nodes to interact with each other during the training because it minimizes a single global objective function. To achieve this goal, we formulate each decision tree as a single disjunction of conjunctions [20] and approximate it with a differentiable function. Next, we use this approximation in a quadratic error cost to construct a single unified objective function. Finally, we minimize this objective function using the gradient descent rule to update the parameters of the discriminants in the decision tree. We call this type of decision tree a disjunctive normal decision tree (DNDT).

Many researchers have proposed converting decision trees into a differentiable form and performing some global parameter tuning to make a smooth decision boundary with high generalization performance. For example [21–24] propose to convert decision trees into artificial neural networks (ANN) and use back-propagation to fine tune the weights and improve the performance. These methods speed up the training of ANNs by using decision trees to initialize the weights of ANNs. However, it would be hard to generalize these methods to random forest framework due to the slowness of back-propagation. Our approach is different from these methods in the sense that unlike the neural networks that have at least two layers of adaptive weights, our disjunctive normal form has only one adaptive layer and thus is faster than back-propagation. Moreover, we will show that DNDTs outperform ANNs.

Fuzzy/soft decision trees are another technique that have been developed to improve the performance of decision trees. Olaru and Wehenkel [25] build a decision tree by introducing a third state at each node. The samples which fall in the third state go to both children nodes. Using this strategy, a sample might contribute to the final decision through multiple paths. Irsoy et al. [26] also propose a soft decision tree that uses a gate function to redirect each sample to all the children with a certain probability. This strategy results in more accurate and simpler trees. The fundamental difference of our approach with soft decision trees is that we propose a global objective function and learn all the splits simultaneously. We will show that DNDTs outperform soft decision trees.

We follow the idea of random forests and use DNDTs as building blocks of a new random forest, called a disjunctive normal random forest (DNRF). While DNRFs have all the advantages of conventional random forests, they outperform them due to their stronger building blocks, *i.e.*, DNDTs. Fig. 1 demonstrates the superior performance of DNRF over conventional random forest with artificial examples. We observe that conventional random forest results in box-like decision boundaries and overfits to the training data while DNRF produces a smooth boundary with lower generalization error. In the results section, we show that, similar to random forests, DNRFs are able to handle multi-class

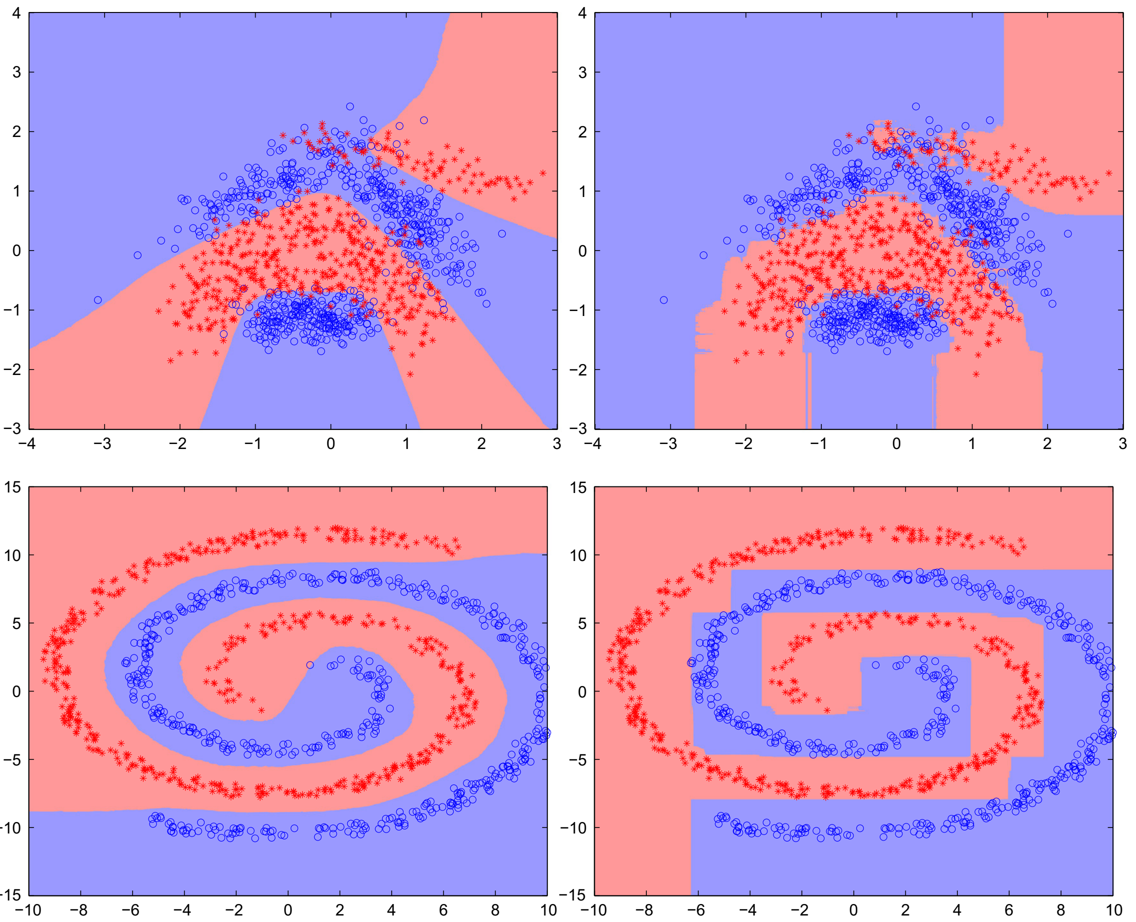

Fig. 1. Comparison of DNRF (left panel) with random forest (right panel) on the banana dataset [29] (upper panel) and two-spiral dataset (lower panel). DNRF results in a smoother decision boundary and, unlike random forests, does not overtrain.

classification problems, but with improved accuracy. We also show that DNRFs outperform state-the-art algorithms such as space partitioning method [27] and multiclass boosting [28].

2. Disjunctive normal random forests

The disjunctive normal random forest (DNRF) is a forest of simpler structures called disjunctive normal decision trees (DNDTs). DNDT is a special form of decision tree in which different nodes interact with each other during training and are learned simultaneously.

In this section, we first describe DNDTs and then show how they can be used in constructing a DNRF. For the sake of simplicity, we consider only the binary classification problem in this section. In the next section, we show how DNRF can be generalized to multi-class problems.

Notation: Unless specified otherwise, we denote vectors with lower case bold symbols, *e.g.*, $\mathbf{w_k}$, elements of vectors with lower case symbols, *e.g.*, w_{kj}, and sets with script symbols, *e.g.*, $\mathcal{S}$.

2.1. Disjunctive normal decision tree

A decision tree is a set of “rules” organized hierarchically in a tree-like graph [14]. An example is shown in Fig. 2. The goal is to predict the label of an input data point based on these rules. During the training, a “split function/rule” is learned at each node of the decision tree. This split function is a binary function, which determines whether incoming data points are sent to the left or right child node. The split function of node k for a d dimensional data point $\mathbf{x}$ can be written as follows:

$$f_k(\mathbf{x}, \mathbf{w_k}) = \mathbb{I}\left(\Phi(\mathbf{x})^T \mathbf{w_k} > 0\right) : \mathbb{R}^{d+1} \times \mathbb{R}^{d+1} \rightarrow \{0, 1\}, \tag{1}$$

where $\mathbf{w_k}$ is an axis-aligned line (it only has two nonzero elements: the bias and a 1 for the chosen splitting axis), which is learned during the training. $\mathbb{I}(\cdot)$ is the binary indicator function and $\Phi(\mathbf{x})$ is $[\phi_1, \ldots, \phi_{d+1}]^T = [x_1, \ldots, x_d, 1]^T$. We drop $\mathbf{w_k}$ and use $f_k(\mathbf{x})$ instead of $f_k(\mathbf{x}, \mathbf{w_k})$ for notational simplicity. Each decision tree can be written as a disjunction of conjunctions which is also known as the disjunctive normal form [20]:

$$h(\mathbf{x}) = \bigvee_{i=1}^{n} \left(\bigwedge_{j \in R_i} f_j(\mathbf{x}) \bigwedge_{j \in L_i} \neg f_j(\mathbf{x}) \right), \tag{2}$$

where n is the number of positive leaf nodes, R_i denotes the set of nodes visited from the root to the ith positive leaf node for which the right child is taken on the path, and similarly L_i denotes the set of visited nodes for which the left child is taken on the same path. For example, for the tree given in Fig. 2, $n=3$, $R_1 = \{2\}$, $L_1 = \{1\}$, $R_2 = \{1\}$, $L_2 = \{3\}$, $R_3 = \{1, 3\}$, $L_3 = \{4\}$, and $h(\mathbf{x})$ can be written as

$$\begin{aligned} h(\mathbf{x}) &= (\neg f_1(\mathbf{x}) \wedge f_2(\mathbf{x})) \vee (f_1(\mathbf{x}) \wedge \neg f_3(\mathbf{x})) \\ &\vee (f_1(\mathbf{x}) \wedge f_3(\mathbf{x}) \wedge \neg f_4(\mathbf{x})). \end{aligned} \tag{3}$$

The data point $\mathbf{x}$ is classified in class “0” if $h(\mathbf{x}) = 0$ and is classified in class “1” if $h(\mathbf{x}) = 1$. To be precise, a sample will be classified in class “1” if and only if it ends up in one of positive leaf nodes. According to Fig. 2, there are only three paths from root to positive leaf nodes and each of these three paths is represented with one conjunction term in Eq. (3). If one of these terms becomes “1”, which means that path is active in the tree, then the whole expression is equal to “1” and if all of them become “0” then the expression is “0”.

2.1.1. The differentiable disjunctive normal form

Once the decision tree is initialized in the conventional manner, we would like to modify Eq. (2) to be able to fine tune it with gradient descent. The first step is to replace Eq. (2) with a differentiable approximation. First, any conjunction of binary variables $\bigwedge_i b_i$ can be replaced by their product $\prod_i b_i$. Also, using De Morgan's laws we can replace the disjunction of binary variables $\bigvee_i b_i$ with $\neg \bigwedge_i \neg b_i$ which in turn can be replaced by the expression $1 - \prod_i (1 - b_i)$. Note that, we use $(1 - b_i)$ to compute $\neg b_i$. Finally, we can approximate the split function with the logistic sigmoid function:

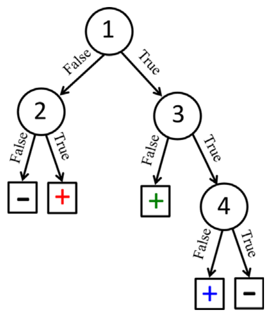

Fig. 2. An example of a decision tree. Non-leaf nodes are denoted with circles and leaf nodes are denoted with squares. A split function is learned at each non-leaf node. Each leaf node represents a class label, “+” for class “1” and “−” for class “0”. The first, second, and third positive leaf nodes are colored in red, green, and blue respectively. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

$$\tilde{f}_k(\mathbf{x}, \mathbf{w_k}) = \frac{1}{1 + e^{-\sum_{j=1}^{d+1} w_{kj} \phi_j}}. \tag{4}$$

This gives in the differentiable disjunctive normal form approximation of h:

$$\begin{aligned} \tilde{h}(\mathbf{x}) &= 1 - \prod_{i=1}^{n} \left[\underbrace{1 - \prod_{j \in R_i} (\tilde{f}_j(\mathbf{x}, \mathbf{w}_j) \prod_{j \in L_i} (1 - \tilde{f}_j(\mathbf{x}, \mathbf{w}_j))}_{g_i(\mathbf{x})} \right] \\ &= 1 - \prod_{i=1}^{n} (1 - g_i(\mathbf{x})). \end{aligned} \tag{5}$$

For the example in Fig. 2 the approximation of h can be written as

$$\begin{aligned} \tilde{h}(\mathbf{x}) = 1 &- \left(1 - \tilde{f}_2(\mathbf{x})\left(1 - \tilde{f}_1(\mathbf{x})\right)\right) \\ &\times \left(1 - \tilde{f}_1(\mathbf{x})\left(1 - \tilde{f}_3(\mathbf{x})\right)\right) \\ &\times \left(1 - \tilde{f}_1(\mathbf{x})\tilde{f}_3(\mathbf{x})\left(1 - \tilde{f}_4(\mathbf{x})\right)\right). \end{aligned} \tag{6}$$

The next step is to update the weights $\mathbf{w_k}$ to improve the performance of the classifier. Unlike decision trees for which weights at each node are learned separately, the disjunctive normal form allows us to update all weights simultaneously; therefore, the obtained decision boundary will be smoother and the generalization performance will be higher compared to decision tree.

Given a set of training samples $\mathcal{S} = \{(\mathbf{x_m}, y_m); m = 1, \ldots, M\}$ where $y_m \in \{0, 1\}$ denotes the desired binary class corresponding to $\mathbf{x_m}$, M denotes the number of training samples and a disjunctive normal classifier $\tilde{h}(\mathbf{x})$, the quadratic error over the training set is

$$E(\tilde{h}, \mathcal{S}) = \sum_{m=1}^{M} \left(y_m - \tilde{h}(\mathbf{x_m})\right)^2. \tag{7}$$

This error function can be minimized using gradient descent. The gradient of the error function with respect to the parameter w_{kj} in the disjunctive normal form, evaluated for the training pair $(\mathbf{x}, y)$, is

$$\begin{aligned} \frac{\partial E}{\partial w_{kj}} = -2x_j(y - \tilde{h}(\mathbf{x})) \times &\left(\sum_{\substack{l \\ k \in R_l}} \left(\prod_{r \neq l} (1 - g_r(\mathbf{x})) g_l(\mathbf{x}) (1 - \tilde{f}_k(\mathbf{x})) \right) \right. \\ &\left. - \sum_{\substack{l \\ k \in L_l}} \left(\prod_{r \neq l} (1 - g_r(\mathbf{x})) g_l(\mathbf{x}) \tilde{f}_k(\mathbf{x}) \right) \right). \end{aligned} \tag{8}$$

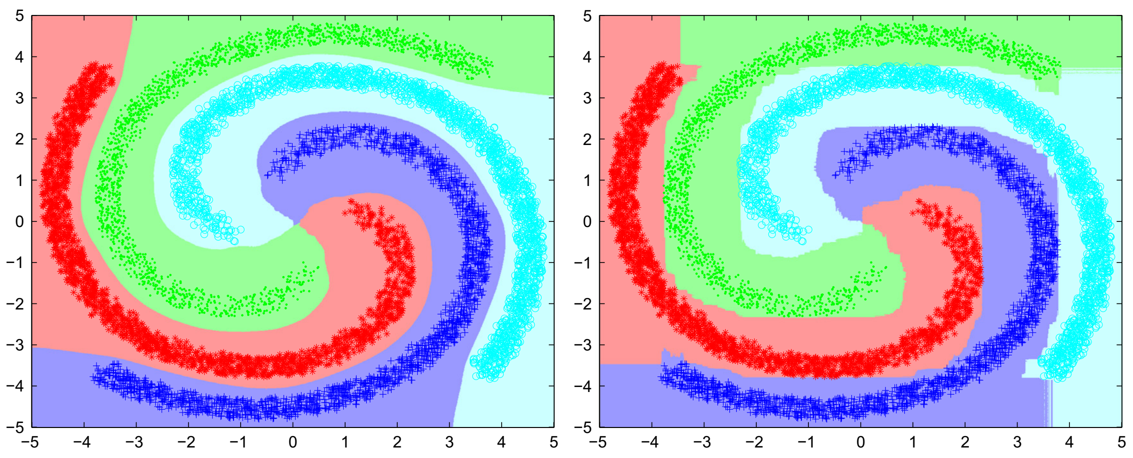

Fig. 3. Comparison of DNRF (left panel) with conventional random forest (right panel) on the four-spiral dataset. DNRF is robust against overfitting and results in better testing performance.

The derivation of Eq. (8) is given in Appendix A. At test time, the weights found by gradient descent are used in Eq. (5) followed by thresholding to predict the label for a new data point.

2.2. Decision tree to random forest

Random forests are an ensemble of randomly trained decision trees [1]. The randomness comes from the fact that each tree is trained using a random subset of training samples. Moreover, at each node of tree a random subset of input features is used to learn the split function. The main idea is to make the decision trees as independent as possible. This improves the robustness and generalization of the ensemble.

Using the same idea, we can use an ensemble of DNDT to generate a DNRF. DNRF takes advantage of more powerful DNDTs compared to the conventional random forest and thus results in better performance. The overall training algorithm for the DNRF is given in Algorithm 1. Note that, in the first step a conventional random forest is trained, which allows DNRFs to take advantage of the randomness existing in the random forest.

Algorithm 1. Training algorithm for the DNRF.

```
Input: Training data, $\mathcal{S}=\{(\mathbf{x_m},y_m); m=1,\dots,M\}$, number of trees, $N$, ratio of training samples per tree, $r$, and number of features per
    node, $F$.
Output: A set of weights, $\{\mathcal{W}_t, t=1,\dots,N\}$.
  • Train a conventional random forest with parameters $N, r, F$.
for $t=1$ to $N$ do
  • $p \leftarrow$ Number of nodes which are visited to reach positive leaf nodes in tree $t$.
  • Convert tree $t$ to disjunctive normal form using Eq. (5).
  • Compute updated weights, $\mathbf{w_1},\dots,\mathbf{w_p}$, using gradient descent (Eq. (8)).
  • $\mathcal{W}_t \leftarrow \{\mathbf{w_1},\dots,\mathbf{w_p}\}$.
end for
```

At test time, the predicted label for a given data point $\mathbf{x}$ can be computed as follows:

$$\tilde{y}=\mathbb{I}\left(\sum_{t=1}^{N}\mathbb{I}\left(\tilde{h}_t(\mathbf{x})>0.5\right)>\frac{N}{2}\right), \tag{9}$$

where N denotes the number of trees in DNRF and $\tilde{h}_t(\mathbf{x})$ is computed using the weights $\mathcal{W}_t$ obtained from the training in Eq. (5).

It must be emphasized that our method can be applied to other variants of random forests that use a feature selection strategy to pick the informative features instead of random sampling [30–32]. These classifiers can be converted to disjunctive normal form and the optimization in Algorithm 1 remains the same.

3. Multi-class DNRF

The concept of DNDT can be extended to multi-class problems. In this case, given a single decision tree, instead of binary leaf nodes, *i.e.*, "+" and "−" leaf nodes, there are leaf nodes with labels $1,\dots,C$, where C is the number of classes. Each tree can be represented with C disjunctive normal functions of type Eq. (2):

$$h^c(\mathbf{x})=\bigvee_{i=1}^{n_c}\left(\bigwedge_{j\in R_i^c} f_j(\mathbf{x})\bigwedge_{j\in L_i^c}\neg f_j(\mathbf{x})\right),\quad c=1,\dots,C, \tag{10}$$

where n_c denotes the number of leaf nodes with label c and R_i^c, L_i^c are similar to the binary case for the leaf nodes with label c. Each of these $h^c(\mathbf{x})$ then can be converted to the differentiable form of Eq. (5). Finally, the weights of each of these functions can be updated using Eq. (8). Note that, each $\tilde{h}^c(\mathbf{x})$ is updated independently and thus the update process can be done in parallel. At test time, the label of an input data point $\mathbf{x}$ can be predicted as follows:

$$\tilde{y}=\arg\max_c \tilde{h}^c(\mathbf{x}), \tag{11}$$

$$\tilde{h}^c(\mathbf{x})=1-\prod_{i=1}^{n_c}\left[1-\prod_{j\in R_i^c}(\tilde{f}_j(\mathbf{x},\mathbf{w_j^c}))\times\prod_{j\in L_i^c}(1-\tilde{f}_j(\mathbf{x},\mathbf{w_j^c}))\right]. \tag{12}$$

Note that, in the above equation the updated weights from the training are used to compute $\tilde{h}^c(\mathbf{x})$. It must be emphasized that although different classes share the same initial weights, the final updated weights, *i.e.*, $\mathbf{w_j^c}$, can be different since the gradient descent is run for different classes separately.

Similar to the binary case, the multi-class DNDT can be used in a forest structure. The training algorithm for multi-class DNRF is described in Algorithm 2.

Algorithm 2. Training algorithm for the multi-class DNRF.

```
Input: Training data, $\mathcal{S} = \{(\mathbf{x_m}, y_m); m = 1, \ldots, M\}$, number of trees, $N$, ratio of training samples per tree, $r$, and number of features per
   node, $F$.
Output: A set of weights, $\{\mathcal{W}_t^c,\ \ t = 1, \ldots, N,\ \ c = 1, \ldots, C\}$.
      •$C \leftarrow$ Number of classes.
      •Train a conventional random forest with parameters $N, r, F$.
for $t = 1$ to $N$ do
   for $c = 1$ to $C$ do
      •$p \leftarrow$ Number of nodes which are visited to reach leaf nodes with label $c$ in tree $t$.
      • Form $\tilde{h}_c^t(\mathbf{x})$ in eq. (12).
      •Compute updated weights, $\mathbf{w_1^c}, \ldots, \mathbf{w_p^c}$, by updating $\tilde{h}_c^t(\mathbf{x})$ using gradient descent (eq. (8)).
•$\mathcal{W}_t^c \leftarrow \{\mathbf{w_1^c}, \ldots, \mathbf{w_p^c}\}$.
   end for
end for
```

At test time, the label for a given data point is computed using voting among all trees:

$$\tilde{y} = \arg\max_{l} \sum_{t=1}^{N} \mathbb{I}\left(\arg\max_{c} \tilde{h}_t^c(\mathbf{x}) = l\right). \quad (13)$$

A comparison of DNRF against random forest on the four-spiral dataset is shown in Fig. 3. The superior performance of DNRF can be seen in the areas where the spirals end. Furthermore, the decision boundaries are more equidistant to the different classes.

4. Experimental results

We performed experimental studies to evaluate the performance of DNDTs and DNRFs in comparison to different classification techniques. The experiments were performed on both binary and multi-class classification problems. We used six datasets for the binary case and four datasets for the multi-class case from the UCI repository [33] and LIBSVM datasets [34]. Before training, the data was normalized by subtracting the mean of each dimension and dividing by the standard deviation of that dimension.

4.1. Binary classification

The eight datasets tested for binary classification are listed in Table 1. For each datasets, we used $\frac{2}{3}$ of the samples for training (10% of these samples were used for validation) and $\frac{1}{3}$ of the samples for testing.

Table 1
Description of the binary datasets used in the experiments.

Dataset	Training samples	Testing samples	No. of features	Categorical
Ionosphere	234	117	33	No
Wisconsin breast cancer	380	189	30	No
German credit	667	333	24	Yes
PIMA diabetes	513	255	8	No
Hearts	180	90	13	Yes
IJCNN	49,990	91,701	22	No
Australian credit	461	229	14	Yes
Sonar	139	69	60	No

We compared the performance of DNDT with decision trees, ANNs, and soft decision trees [26]. The test errors are reported in Table 2.

DNDT outperforms decision trees with a large margin. It also outperforms both ANNs and soft decision trees. These results assert that the superior performance of DNDT comes from both non-orthogonal splits in a tree structure, as opposed to the decision tree, and unified learning of all the learners, as opposed to the soft decision tree. For soft decision trees, we used the code publicly available by the authors of [26]. The training times of different methods for each dataset are given in Table 3. While DNDT is slower than decision trees, it is faster compared to soft decision trees and ANNs. It is worth mentioning that each epoch of update in DNDT is nearly 4 times faster than the back-propagation in ANNs due to the simpler structure of the disjunctive normal form compared to ANN. This simplicity comes from the fact that there is only one set of parameters, $\mathbf{w_k}$, in the differentiable disjunctive normal form (Eq. (5)) while in ANNs there are at least two sets of parameters, i.e., weights from the input layer to the hidden layer and weights from the hidden layer to the output layer. The time complexities of decision trees and DNDTs at test time are similar.

We also compared the performance of DNRF with SVMs, boosted trees, oblique random forests (ORF) [16], rotation forests [17], and random forests. For SVMs, we used RBF kernel and the parameters of kernel were found using the search code available in the LIBSVM library [34]. For boosted trees, we used the code publicly available by the authors of [35]. We also used the publicly available R package “obliqueRF” provided by the authors of [16] to run ORF on the binary datasets. Their code supports three different node models: ridge regression, SVM, and logistic regression. For rotation forest, we used the publicly available code provided as part of Weka by the authors of [17]. For all the datasets we used $F = \sqrt{d}$ as the number of features per node in the random forest and used 10% of the training set as the validation set to fine tune the number of trees, N. The same validation set was used to control the number of epochs and tune the step size in the gradient descent algorithm for the DNRF. We ran each classifier 50 times, except for SVMs which give deterministic results, for each dataset, and the average testing errors for different methods are reported in Table 4. The standard deviations are given in parentheses. DNRFs outperform SVMs, boosted trees, random forests, rotation forests, and ORFs. The standard deviation of DNRF is generally lower than other methods. We also ran the paired-sample t-test between DNRF and all the other methods. In all the cases, the null hypothesis, *i.e.*, both models have the same mean,

Table 2
Test errors of different methods for six binary datasets.

Method	Ionosphere (%)	Wisconsin breast cancer (%)	German credit (%)	PIMA diabetes (%)	Hearts (%)	IJCNN (%)	Australian credit (%)	Sonar (%)
Decision tree	12.48	6.08	32.73	31.59	27.62	4.79	19.72	29.57
ANN	12.10	2.28	26.96	22.11	20.26	2.02	15.65	30.43
Soft decision tree [26]	11.97	2.12	25.53	20.78	13.33	2.27	16.59	30.43
DNDT	7.15	1.89	24.44	20.56	13.11	1.94	15.30	22.55

Table 3
Training time (seconds) of different methods for eight binary datasets.

Method	Ionosphere	Wisconsin breast cancer	German credit	PIMA diabetes	Hearts	IJCNN	Australian Credit	Sonar
Decision tree	0.009	0.009	0.010	0.008	0.022	0.500	0.018	0.015
ANN	0.395	0.854	0.746	0.356	0.956	1860.81	0.486	0.204
Soft decision tree [26]	0.267	0.657	1.858	0.410	0.073	34,958.626	0.392	0.232
DNDT	0.036	0.033	0.228	0.095	0.031	535.36	0.077	0.045

Table 4
Test errors of different methods for eight binary datasets (average over 50 iterations). The standard deviations are given in parentheses.

Method	Ionosphere	Wisconsin breast cancer	German credit	PIMA diabetes	Hearts	IJCNN	Australian credit	Sonar
SVM	4.27% (–)	1.59% (–)	26.12% (–)	22.35% (–)	21.11% (–)	1.31% (–)	14.85% (–)	17.39% (–)
Boosted trees [35]	5.16% (0.9)	2.38% (0.42)	24.29% (0.96)	23.29% (0.99)	15.00% (1.68)	1.39% (0.05)	14.23% (0.91)	18.49% (1.64)
ORF-ridge [16]	5.69% (0.56)	1.63% (0.21)	25.10% (0.94)	23.25% (0.85)	15.07% (0.90)	1.68% (0.02)	16.14% (0.46)	24.12% (1.75)
ORF-svm [16]	4.32% (0.50)	1.58% (0.07)	24.23% (0.56)	23.25% (0.81)	13.18% (0.9)	1.87% (0.04)	15.50% (0.64)	27.86% (1.74)
ORF-log [16]	4.50% (0.62)	1.74% (0.26)	25.23% (0.82)	22.43% (1.26)	15.27% (0.99)	1.62% (0.05)	15.84% (0.59)	24.52% (1.92)
Rotation forest [17]	5.04% (1.35)	1.82% (0.76)	25.23% (1.22)	21.63% (1.42)	19.05% (2.56)	2.14% (0.24)	16.09% (0.79)	23.12(3.40)
Random forest [1]	6.99% (0.94)	2.15% (0.41)	24.71% (0.82)	23.92% (0.64)	15.11% (1.14)	2.00% (0.04)	15.81% (0.74)	25.48% (2.01)
DNRF	3.38% (0.57)	0.53% (≈ 0)	22.91% (0.37)	19.41% (0.42)	12.22% (≈ 0)	1.14% (0.01)	13.56% (0.44)	18.14% (1.50)

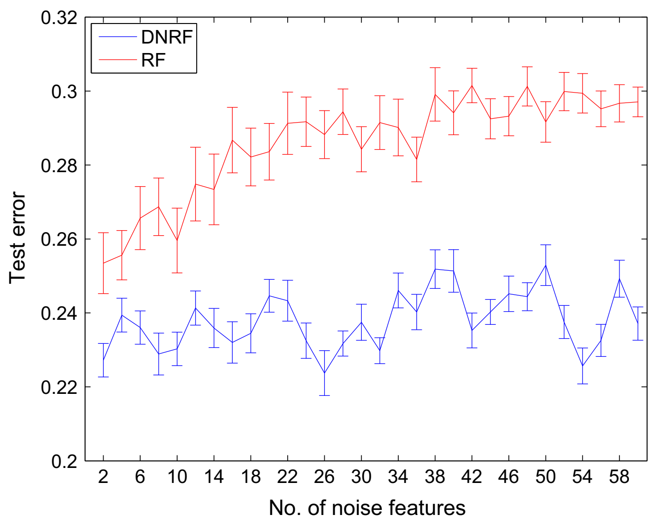

Fig. 4. The robustness of random forest and DNRF on problems with increasing number of noise/random variables. The dataset, *i.e.*, German credit dataset, has 24 features and an increasing number of noise variables were added. Each experiment was run 50 times and the error bars illustrate the standard deviations. For all the experiments, random forest used $F = \sqrt{d}$ as the number of features per node.

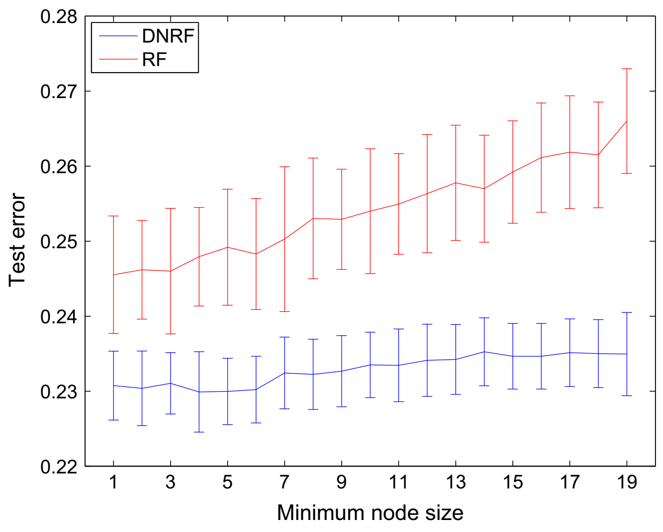

Fig. 5. The effect of tree size, *i.e.*, tree depth, on the performance of random forest and DNRF. The depth of tree was controlled by the minimum node size. Smaller node size results in deeper trees. Each model was run 50 times and the error bars represent standard deviations. The German credit dataset was used in this experiment.

was rejected at 5% significance level. These results again assert that the main advantage of DNRFs comes from both the non-orthogonal splits, as opposed to random forests, and the unified learning of all splits, as opposed to ORFs and rotation forests.

Similar to [36], we examined the effect of noise features and tree size on the performance of random forests and DNRFs. In the first experiment, we incrementally added noise/random features to the German credit dataset. Fig. 4 shows the test errors for different number of noise features. As the number of noise features increases the chance that relevant features are selected at each node decreases. While this degrades the performance of random forests, DNRFs remain stable due to the unified learning strategy. Optimizing all the nodes together decreases the effect of nodes that contain only noise features. In the second experiment, we incrementally decreased the tree size by increasing the minimum node size. We stop splitting the nodes that contain less than

Table 5
Description of the multi-class datasets used in the experiments.

Dataset	Training samples	Testing samples	No. of features	No. of classes	Categorical
Pendigit	7494	3498	16	10	No
Optdigit	3823	1797	62	10	No
Landsat	4435	2000	36	6	No
Letter	16,000	4000	16	26	No

Table 6
Test errors of different methods on four UCI datasets (multi-class classification).

Method	Pendigit (%)	Optdigit (%)	Landsat (%)	Letter (%)
GD-MCBoost [28]	7.06	7.68	13.35	40.35
Space partitioning [27]	4.32	4.23	13.95	13.08
Random forest [1]	3.87	3.22	10.49	4.70
DNRF	2.17	2.30	9.63	2.05

samples than the minimum node size. Fig. 5 shows the test errors on the German credit dataset for different values of minimum node size. DNRF is less sensitive to the size of tree compared to random forest. This can be seen as the effectiveness of the unified learning strategy which gives more degrees of freedom to DNRF.

4.2. *Multi-class classification*

The four datasets tested for multi-class classification are listed in Table 5. Similar to binary case, we used $F=\sqrt{d}$ as the number of features per node in the random forest and used 10% of the training set as the validation set to fine tune the number of trees, N, and the ratio of training samples per tree, r.

We ran the experiments 50 times for each dataset and the average testing errors are reported in Table 6. While random forests outperform the state-of-the-art algorithms [27,28], DNRF reduces the error rate even further. These results assert that DNRF idea works for multi-class datasets as well as the binary datasets. DNRF outperforms [27], which also takes advantage of space partitioning. This shows that our global optimization is more effective than [27].

5. Conclusion

We introduced a new learning scheme for random forests, called DNRF, based on the disjunctive normal form of decision trees. Unlike conventional random forests with orthogonal axis-aligned splits, DNRFs can learn arbitrary non-axis-aligned splits. Moreover, DNRFs allow different nodes of a decision tree interact with each other during training in a unified optimization framework. We showed that DNRFs outperform conventional random forests on binary and multi-class benchmark datasets. Our results are also better than oblique random forests [16] which learns non-orthogonal learners at each node.

It must be emphasized that optimizing all the individual trees together in DNRF would increase the correlation between the trees and increasing the correlation decreases the forest performance [1]. Hence, treating each tree individually, as mentioned in Algorithm 2, is crucial to the performance of DNRF. The initialization is also important to the performance of DNRF. If the DNRF is initialized with a random tree, it performs poorly. The reason is that the cost function is not convex and gradient descent might get stuck in a local minima.

DNRFs can handle categorical data similar to conventional random forests. After a conventional RF is trained, the same optimization approach can be applied to construct a DNRF. However, DNRFs do not handle missing values in the current format. One possible solution is to assign zero weight to the missing features in the paths containing samples with missing values, but this is a topic of future research.

DNRFs are slower to train compared to conventional random forests due to the gradient descent step. At test time, DNRFs computational time is similar to other multiplicative classifiers such as ANNs and SVMs.

Finally, even though we described DNRF for the cases that weak learners are linear, the DNRF formulation can be extended to any differentiable nonlinear weak learners theoretically. The performance, advantages, and disadvantages of nonlinear DNRFs can be a topic of future research.

Conflict of interest

None declared.

Acknowledgments

This work was supported by NSF IIS-1149299 (T.T.). We thank the editor and reviewers whose comments helped greatly improve the paper.

Appendix A. Derivation of gradient

In this section we show the derivation of Eq. (8). The gradient for the training pair $(\mathbf{x}, y)$ can be computed using the chain rule for differentiation:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial \tilde{h}} \sum_{l} \frac{\partial \tilde{h}}{\partial g_l} \frac{\partial g_l}{\partial \tilde{f}_j} \frac{\partial \tilde{f}_j}{\partial w_{kj}}. \tag{A.1}$$

Each of the derivatives on the right hand side can be computed as follows:

$$\frac{\partial E}{\partial \tilde{h}} = -2(y-\tilde{h}(\mathbf{x})),$$

$$\frac{\partial \tilde{h}}{\partial g_l} = \prod_{r \neq l} (1-g_r(\mathbf{x})),$$

$$\frac{\partial g_l}{\partial \tilde{f}_j} = \begin{cases} \prod\limits_{\substack{r \neq j \\ r \in R_l}} \tilde{f}_r(\mathbf{x}) \prod\limits_{r \in L_l} (1-\tilde{f}_r(\mathbf{x})) & \text{if } j \in R_l \\ -\prod\limits_{r \in R_l} \tilde{f}_r(\mathbf{x}) \prod\limits_{\substack{r \neq j \\ r \in L_l}} (1-\tilde{f}_r(\mathbf{x})) & \text{if } j \in L_l \\ 0 & \text{otherwise,} \end{cases}$$

$$\frac{\partial \tilde{f}_j}{\partial w_{kj}} = x_j \tilde{f}_j(\mathbf{x})(1-\tilde{f}_j(\mathbf{x})).$$

By multiplying these derivatives the gradient in Eq. (8) is obtained.

References

[1] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[2] F. Schroff, A. Criminisi, A. Zisserman, Object class segmentation using random forests, in: British Machine Vision Conference, 2008.
[3] A. Bosch, A. Zisserman, X. Muoz, Image classification using random forests and ferns, in: IEEE Eleventh International Conference on Computer Vision 2007, ICCV 2007, pp. 1–8.
[4] P. Kontschieder, S.R. Bulò, A. Criminisi, P. Kohli, M. Pelillo, H. Bischof, Context-sensitive decision forests for object detection, in: Advances in Neural Information Processing Systems vol. 25, 2012, pp. 440–448.
[5] A. Criminisi, J. Shotton, D. Robertson, E. Konukoglu, Regression forests for efficient anatomy detection and localization in ct studies, in: Medical Computer Vision. Recognition Techniques and Applications in Medical Imaging, Springer, Berlin, Heidelberg, 2011, pp. 106–117..

[6] D. Laptev, A. Vezhnevets, S. Dwivedi, J. Buhmann, Anisotropic sstem image segmentation using dense correspondence across sections, in: Medical Image Computing and Computer-Assisted Intervention MICCAI 2012, 2012, pp. 323–330.
[7] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[8] M. Kearns, L. Valiant, Cryptographic limitations on learning boolean formulae and finite automata, J. ACM 41 (1) (1994) 67–95.
[9] S. Haykin, Neural networks—a comprehensive foundation, second ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.
[10] S.E. Fahlman, C. Lebiere, The cascade-correlation learning architecture, in: Advances in Neural Information Processing Systems, Volume 2, pp. 524–532.
[11] G. Ou, Y.L. Murphey, Multi-class pattern classification using neural networks, Pattern Recognit. 40 (1) (2007) 4–18.
[12] C.-W. Hsu, C.-J. Lin, A comparison of methods for multiclass support vector machines, IEEE Trans. Neural Netw. 13 (2) (2002) 415–425.
[13] A. Torralba, K. Murphy, W. Freeman, Sharing visual features for multiclass and multiview object detection, IEEE Trans. Pattern Anal. Mach. Intell. 29 (5) (2007) 854–869. http://dx.doi.org/10.1109/TPAMI.2007.1055.
[14] A. Criminisi, J. Shotton, E. Konukoglu, Decision forests: a unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning, Found. Trends Comput. Graph. Vis. 7 (2–3) (2011) 81–227.
[15] R. Caruana, N. Karampatziakis, A. Yessenalina, An empirical evaluation of supervised learning in high dimensions, in: Proceedings of the Twenty-fifth International Conference on Machine Learning, ACM, 2008, pp. 96–103.
[16] B.H. Menze, B.M. Kelm, D.N. Splitthoff, U. Koethe, F.A. Hamprecht, On oblique random forests, in: Machine Learning and Knowledge Discovery in Databases, Springer, 2011, pp. 453–469..
[17] J.J. Rodriguez, L.I. Kuncheva, C.J. Alonso, Rotation forest: a new classifier ensemble method, IEEE Trans. Pattern Anal. Mach. Intell. 28 (10) (2006) 1619–1630.
[18] S. Bernard, L. Heutte, S. Adam, Forest-RK: a new random forest induction method, in: Advanced Intelligent Computing Theories and Applications with Aspects of Artificial Intelligence, Springer, Berlin, Heidelberg, 2008, dummy pp. 430–437..
[19] E.E. Tripoliti, D.I. Fotiadis, G. Manis, Modifications of the construction and voting mechanisms of the random forests algorithm, Data Knowl. Eng. 87 (0) (2013) 41–65.
[20] M. Hazewinkel, Encyclopaedia of Mathematics, Suppl. III, vol. 13, Springer, Netherlands, 2001.
[21] K.J. Cios, N. Liu, A machine learning method for generation of a neural network architecture: a continuous id3 algorithm, IEEE Trans. Neural Netw. 3 (2) (1992) 280–291.
[22] I. Ivanova, M. Kubat, Initialization of neural networks by means of decision trees, Knowl. Based Syst. 8 (6) (1995) 333–344.
[23] A. Banerjee, Initializing neural networks using decision trees, Computational learning theory and natural learning systems 4 (1997) 3–15.
[24] R. Setiono, W.K. Leow, On mapping decision trees and neural networks, Knowl. Based Syst. 12 (3) (1999) 95–99.
[25] C. Olaru, L. Wehenkel, A complete fuzzy decision tree technique, Fuzzy Sets Syst. 138 (2) (2003) 221–254.
[26] O. Irsoy, O.T. Yildiz, E. Alpaydin, Soft decision trees, in: IEEE Twenty-first International Conference on Pattern Recognition (ICPR), 2012, pp. 1819–1822.
[27] J. Wang, V. Saligrama, Local supervised learning through space partitioning, in: NIPS, 2012, pp. 91–99.
[28] M.J. Saberian, N. Vasconcelos, Multiclass boosting: theory and algorithms, in: Advances in Neural Information Processing Systems, 2011, pp. 2124–2132.
[29] G. Ratsch, T. Onoda, K.-R. Muller, Soft margins for adaboost, in: Machine Learning, 2000, pp. 287–320.
[30] Y. Ye, Q. Wu, J. Zhexue Huang, M.K. Ng, X. Li, Stratified sampling for feature subspace selection in random forests for high dimensional data, Pattern Recognit. 46 (3) (2013) 769–787.
[31] Q. Wu, Y. Ye, Y. Liu, M.K. Ng, Snp selection and classification of genome-wide snp data using stratified sampling random forests, IEEE Trans. NanoBiosci. 11 (3) (2012) 216–227.
[32] D. Amaratunga, J. Cabrera, Y.-S. Lee, Enriched random forests, Bioinformatics 24 (18) (2008) 2010–2014.
[33] A. Frank, A. Asuncion, UCI machine learning repository, ⟨http://archive.ics.uci.edu/ml⟩ (2010).
[34] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011) 27:1–27:27, software available at ⟨http://www.csie.ntu.edu.tw/~cjlin/libsvm⟩.
[35] R. Sznitman, C.J. Becker, F. Fleuret, P. Fua, Fast object detection with entropy-driven evaluation, in: Computer Vision and Pattern Recognition (CVPR), 2013.
[36] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer Series in Statistics, Springer, New York Inc., USA, New York, NY, 2001.

Mojtaba Seyedhosseini received the B.S. degree in Electrical Engineering from the University of Tehran in 2007, and the M.S. degree in Electrical Engineering from the Sharif University of Technology in 2009. He received his Ph.D. degree from the Scientific Computing and Imaging (SCI) Institute at the University of Utah in 2014. His research interests include machine learning, computer vision, statistical pattern recognition, and image analysis.

Tolga Tasdizen received the B.S. degree in Electrical and Electronics Engineering from Bogazici University in 1995. He received his M.S. and Ph.D. degrees in Engineering from Brown University in 1997 and 2001, respectively. After working as a postdoctoral researcher position at the Scientific Computing and Imaging (SCI) Institute at the University of Utah, he was a Research Assistant Professor in the School of Computing at the same institution. Since 2008, he has been with the Department of Electrical and Computer Engineering at the University of Utah where he is currently an Associate Professor. Dr. Tasdizen is also a Utah Science Technology and Research Initiative (USTAR) faculty member in the SCI Institute. His research interests are in image processing, computer vision and pattern recognition with a focus on applications in biological and medical image analysis. Dr. Tasdizen is a recipient of the National Science Foundation's CAREER award.