A Dynamic Boosted Ensemble Learning Method Based on Random Forest

Xingzhang Ren

School of Software and Microelectronics, Peking University, Beijing, China
xzhren@pku.edu.cn

Chen Long

School of Software and Microelectronics, Peking University, Beijing, China
1401110614@pku.edu.cn

Leilei Zhang

School of Software and Microelectronics, Peking University, Beijing, China
leilei_zhang@pku.edu.cn

Ye Wei

National Engineering Research Center for Software Engineering, Peking University, China
wye@pku.edu.cn

Dongdong Du

School of Electronics Engineering and Computer Science, Peking University, Beijing, China
dudong@pku.edu.cn

Jingxi Liang

School of Software and Microelectronics, Peking University, Beijing, China
jingxil@pku.edu.cn

Shikun Zhang

National Engineering Research Center for Software Engineering, Peking University, China
zhangsk@pku.edu.cn

Weiping Li

School of Software and Microelectronics, Peking University, Beijing, China
wpli@ss.pku.edu.cn

—— **Abstract** ——

We propose a dynamic boosted ensemble learning method based on random forest (DBRF), a novel ensemble algorithm that incorporates the notion of hard example mining into Random Forest (RF) and thus combines the high accuracy of Boosting algorithm with the strong generalization of Bagging algorithm. Specifically, we propose to measure the quality of each leaf node of every decision tree in the random forest to determine hard examples. By iteratively training and then removing easy examples from training data, we evolve the random forest to focus on hard examples dynamically so as to learn decision boundaries better. Data can be cascaded through these random forests learned in each iteration in sequence to generate predictions, thus making RF deep. We also propose to use evolution mechanism and smart iteration mechanism to improve the performance of the model. DBRF outperforms RF on three UCI datasets and achieved state-of-the-art results compared to other deep models. Moreover, we show that DBRF is also a new way of sampling and can be very useful when learning from imbalanced data.

1   Introduction

Deep learning has been making significant progress in many fields in recent years especially in computer vision [14] and speech recognition [12]. However, it remains a black box as of today. The hyper-parameter tuning is a hard and tedious process, which makes it more of an artistry than hard-core science. Moreover, its lack of interpretability poses a major holdback to its adoption in many practical scenarios. Some researchers try to combine deep learning with traditional machine learning methods such as Random Forest (RF). For example, RF is used to enhance interpretability and performance with deep learning models [11, 13]. There's also advancement in representation learning [8] and trying to make RF deep [20].

RF is the paradigm of Bagging algorithm in ensemble learning. In 2001, Breiman combined decision trees [5] into a forest [4]. Features (columns) and data (rows) are randomly sampled to generate multiple decision trees and their predictions are aggregated. RF improves the prediction accuracy without significant increase in computation, and it is insensitive to multicollinearity. Its performance is robust against missing and imbalanced data, and it can well measure the roles of thousands of variables [6]. Compared with models in deep learning, it has the advantage of small computation, outstanding generalization ability and strong interpretability, which makes it ever appealing to researchers and practitioners in spite of its long existence.

However, there are also limitations of RF. First of all, the RF model can only be extended horizontally (more decision trees) but not vertically since the decision trees exist in parallel and cannot be stacked in layers in the same fashion as neurons in neural networks. Secondly, these decision trees have the same weight in voting for the final prediction despite that some of these trees may perform poorly. Lastly, all points in training data have the same weight and are treated equally in the sampling and training process, despite that some of the data are easy to classify while others are hard.

In this paper, we propose a dynamic boosted ensemble learning method based on random forest (DBRF) to overcome these limitations. It is a novel ensemble algorithm where such notion of hard example mining as in Boosting is incorporated into RF. Specifically, we propose to score the quality of each leaf node of every decision tree in the random forest and then vote to determine hard examples. By iteratively training and then removing easy examples from training data to train again, we evolve the model to focus on hard examples dynamically so as to learn decision boundaries better. After training, the test data can be cascaded through these learned random forests of each iteration in sequence to generate predictions, thus achieving more depth with RF. In addition, we propose to use evolution mechanism and smart iteration mechanism to enhance the performance of DBRF and conducted ablation test. We evaluate DBRF on public datasets and it outperforms RF and achieves state-of-the-art results compared to other deep models. Last but not least, we also perform visualizations to validate DBRF and analyze its effectiveness from a sampling point of view.

The contributions of this paper are listed as follows:

1. We propose a novel ensemble algorithm called DBRF to extend RF vertically by iteratively mining and training on hard examples, and thus incorporate boosting into RF training process. It outperforms RF.

2. We propose a criterion to measure the quality of a leaf node in a decision tree and a voting mechanism to determine hard examples.

3. We propose an evolution mechanism which filters out poor decision trees in a random forest, together with iteration mechanism to enhance the performance of DBRF and achieve state-of-the-art results on three UCI datasets.

4. The hard example mining process proposed with DBRF can be seen as a novel way of sampling. We demonstrate the effectiveness of using DBRF to deal with imbalanced data.

The rest of the paper is organized as follows. Section 2 explains our motives. Section 3 presents our proposed model DBRF and the enhancement mechanisms. Section 4 reports the results of multiple evaluation experiments. Finally, Section 5 contains related work and Section 6 concludes the paper.
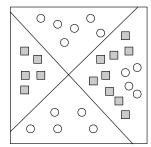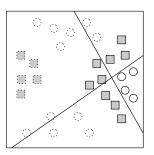
2 Motivation

In typical classification tasks, classifiers are trained to find the decision boundaries of labelled data using a set of features. The distribution of data near the decision boundaries largely affect the performance of the classifier and is generally where the overfitting and under-fitting trade-off is made. Those data whose predicted labels are not agreed upon across multiple classifiers, probably located near the decision boundaries, can be seen as hard examples. On the contrary, those data whose predicted labels are consistent and correct across all classifiers can be seen as easy examples. Naturally, the performance on hard examples determines how good a classifier is compared with others and it's ideal to focus on these hard examples during training.

Hard example mining is a common notion in many machine learning algorithms. A typical example is AdaBoost, in which wrongly classified examples are deemed as hard examples. The general idea is to assign more weight to those hard examples and train the model iteratively until convergence. This hard example mining notion has also been adopted in deep learning to successfully improve the performance of the model, especially in object recognition tasks, such as OHEM [16], S-OHEM [15] and A-Fast-RCNN [17].

Thus, we are motivated to incorporate this hard example mining notion, or boosting, into RF. It can be integrated smoothly within the RF training process since a random forest is an ensemble of decision trees trained on randomly sampled data and features, which are weak classifiers. If all decision trees make right predictions on a part of the training data, these data can be considered as easy examples. Rather than assign more weight to hard examples, we can simply remove these data from the training data to achieve the same effect. When training the RF model again on the new set of data, we essentially make the model focus on hard examples and learn the decision boundaries better without making the classifier more complex and sacrificing generalization abilities. The idea is illustrated in Figure 1.

Figure 1 Illustration of removing easy examples to improve learning. When we train a classifier as shown on the left, one of the decision regions is not ideal, but when we remove the easy examples from the good decision region and then train a classifier as shown in the right figure. At this point, the ensemble of the two classifiers allows the training data to be perfectly divided.

One point to clarify is that the aforementioned criterion to determine easy examples,

i.e. data that are classified right across multiple classifiers, is not strict enough for removal because in extreme cases where data are imbalanced, we may be left with data of only one label. We will explain our proposed criterion in detail in Section 3. But basically, with RF, we are seeking to find good rules and define easy examples as ones that fit the good rules of all classifiers. In a decision tree, each leaf node represents a rule, thus we need to come up with a criterion to determine whether a leaf node is good or not.

3 The Proposed Approach: DBRF

In this section, we present out proposed approach DBRF based on RF, which drives the evolution of the model by iteratively updating the training data. Compared to RF, DBRF can greatly improve the performance. We will first present the general framework and the basic algorithm of our model and then propose two mechanisms to enhance the basic model.

In the following, we consider a typical classification task where $\mathcal{X}$ and $\mathcal{Y}$ denote input and output space respectively. For a decision tree T, N denotes a decision node and L a leaf node. A RF F is a set of T trained on data $D = \{X, Y\}$ where X is points set in $\mathcal{X}$ and Y is corresponding labels set in $\mathcal{Y}$.

3.1 The General Framework

Figure 2 Illustration of DBRF with three iterations. During training (above the dotted line), firstly, a random forest is trained on the training set, and then the leaf nodes are evaluated through the HEM process (Section 3.2). The hard examples (with bold borders) can be identified accordingly (by removing easy examples) as a new training set for the next iteration. Similarly, at each iteration during test (below the dotted line), easy examples are identified (except the last iteration) and their predictions are outputted.
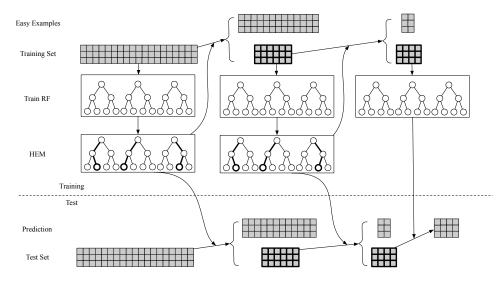

Figure 2 illustrates the basic structure of DBRF. We first split a dataset into training set and test set and then train a random forest of decision trees on the training set $D^d = \{X, Y\}$. Next, we use a criterion to measure the quality of each leaf node of each decision tree in the forest. In Figure 2, the leaf nodes circled in bold line are good ones which represent possible easy examples. By using the proposed hard example mining (HEM) method to be elaborated in Section 3.2, we can divide D^d into two parts $\{D_e^d, D_h^d\}$, where D_e^d denotes easy examples

and D_h^d hard examples. Only D_h^d are preserved for the next iteration's training. This process keeps on iterating until a predetermined n^{th} iteration is done.

At iteration i, we need to preserve the random forest model F_i trained in current iteration and the evaluation scores of all leaf nodes of all decision trees in F_i, denoted as Π_i. For predicting test set D^t, we first use F_1 to predict and then divide D^t into $\{D_e^t, D_h^t\}$, according to F_1. For D_e^t the easy data, the predictions made by F_1 are outputted as the final prediction result, while for the hard data D_h^t, we will feed them into F_2 for the next iteration's prediction. This process goes on until no longer contains data, or until the last iteration. In the last iteration n, the output of F_n will be the final predicted labels of the corresponding data.
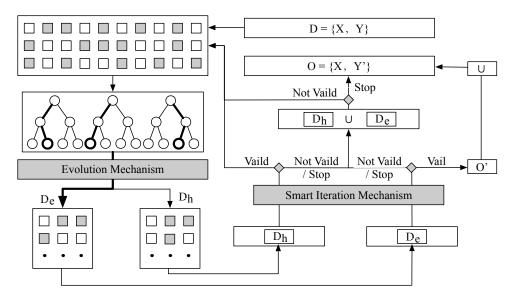
This training and test process can be further written in pseudo code shown in Algorithm 1.

Algorithm 1 Dynamic Boosted Random Forest

Input: Training set: D^d, Test set: D^t, Iterations n
Output: Prediction of test set: O
 1: // Training Procedure
 2: $D \leftarrow D^d$
 3: **for** $i = 1 \rightarrow n$ **do**
 4:     Train RF as F_i on dataset D
 5:     Get scores of leaf nodes Π_i by HEM (Section 3.2)
 6:     Split D into easy data D_e and hard data D_h according to F_i and Π_i
 7:     $D \leftarrow D_h$
 8:     add F_i to F-list
 9:     add π_i to Π-list
10: **end for**
11:
12: // Test Procedure
13: $D \leftarrow D^t$
14: $O \leftarrow \varnothing$
15: **for** $i = 1 \rightarrow n$ **do**
16:     $F_i \leftarrow F$-list at i
17:     $\Pi_i \leftarrow \Pi$-list at i
18:     Split D into easy data D_e and hard data D_h according to F_i and Π_i
19:     Predict D_e as O' by F_i
20:     $D \leftarrow D_h$
21:     $O = O \cup O'$
22: **end for**
23: Predict D_h as O' by F_n
24: $O = O \cup O'$
25: **return** O

We propose to use two mechanisms to enhance the basic model of DBRF. To prevent the negative influence of poor decision trees in a random forest on the HEM process, we propose to use an evolution mechanism to eliminate them from the random forest, which will be further elaborated in Section 3.3. Furthermore, we propose a smart iteration mechanism to better guide the HEM process, as elaborated in Section 3.3. The model of DBRF with two mechanisms is illustrated in Figure 3.

■ **Figure 3** Illustration of the mechanisms of DBRF. The evolution mechanism is used to eliminate poor decision trees from the random forest and the smart iteration mechanism is used to better guide the HEM process.

3.2 Hard Example Mining (HEM)

A random forest consists of multiple decision trees. We use D_F^e and D_F^h to denote easy examples and hard examples of a random forest, and D_T^e and D_T^h to denote easy examples and hard examples of each decision tree. We propose that D_F^e and D_F^h can be generated as

$$D_F^e = \bigcap_{T \in F} D_T^e, \ D_F^h = D - D_F^e$$

meaning that the intersection of easy examples of all the decision trees are considered as easy examples.

A leaf node L in a decision tree corresponds to a rule R_L. Suppose the path to reach the leaf node L from the root in a decision tree F is $W_L = \{n_1, n_2, n_3, ..., n_i\}$, where n_1 to n_i denote the decision nodes along the path. The probability distribution in $\mathcal{Y}$ in node L is π_L, c is any label in $\mathcal{Y}$, then we can define the rule R_L corresponding to the leaf node L as

$$R_L : x \mid x \in \bigcap_{n_i \in W_L} r(n_i) \Rightarrow y = arg \ \max_c \pi_L(c)$$

where $r(\cdot)$ is a function that represents the data that satisfy the rule of a decision node n. Then the easy examples of a decision tree can be defined as

$$D_T^e = \{x \mid x \in \bigcap_{L \in T, T \in F} (score(L) > \sigma)\}$$

where $score(\cdot)$ is a leaf node evaluation metric and σ is the threshold, which is implemented as the average score of leaf nodes in all decision trees in our model.

Next, we propose several leaf node evaluation metrics $score(\cdot)$. We may use support and confidence, as were used to evaluate association rules [1] to score the leaf node. For the rule R_L of a leaf node L, all data that satisfy the preconditions of the rule, i.e. $x \in_{n_i \in W_L} r(n_i)$,

compose candidate set C. Then

$$score_{supp}(L) = \frac{|C|}{|X|}$$

$$score_{conf}(L) = \frac{|\{y = c,\ x \in C\}|}{|C|}$$

Since both support and confidence derive from association rules, we can merge them to be f1 score.

$$score_{f1}(L) = 2 \cdot \frac{score_{supp}(L) \cdot score_{conf}(L)}{score_{supp}(L) + score_{conf}(L)}$$

Also, since Gini impurity (gini) and information gain (entropy) are the partitioning criteria used in decision tree, we can define gini score and entropy score of a leaf node as

$$score_{gini}(L) = -Gini(L) = \sum_{j=1}^{c} p_j^2 - 1$$

$$score_{entropy}(L) = -Entropy(L) = \sum_{j=1}^{c} p_j^2\ log\ p_j$$

where c_j is j^{th} label in $\mathcal{Y}$ and p_j is the probability of c_j .

All in all, we propose three leaf node evaluation metrics, namely *score-gini*, *score-entropy*, and *score-f1*. Their effects are explored in experiments, as discussed in Section 4.1.

3.3 Mechanisms

3.3.1 Evolution Mechanism

In RF, data and features are randomly sampled to generate multiple decision trees. Chances are that some of them are of poor quality, which have negative effect on the voting process in our proposed HEM method. To overcome this problem, we draw on the idea of genetic programming [2] and propose to use a fitness formula to eliminate those decision trees with lower scores before determining easy and hard examples from the training data. In our implementation, we use the average evaluation metric score across all leaf nodes as the fitness score of each decision tree. The specific procedure is as follows.

1. At each iteration, calculate the fitness score of each decision tree in a random forest after training.
2. Set an elimination ratio (20% by default) and calculate the threshold, and then eliminate those decision trees whose score is lower than the threshold.
3. Determine the easy and hard examples by voting among the rest of the decision trees and remove them to generate a new training set for the next iteration.

3.3.2 Smart Iteration Mechanism

At each iteration during training, the division of D^d as $\{D_e^d, D_h^d\}$, might not be ideal in terms of validation accuracy. We propose to use the prediction accuracy on the training data D^d as the validation accuracy (which can be generated using k-fold cross-validation as mentioned in Stacking [18, 3]), to judge the quality of the division and annul the division or terminate training if rules are triggered. We propose two smart iteration rules as follows.

1. If the validation accuracy decreases for N consecutive times (five by default), apply early termination.

2. If the validation accuracy of easy examples D^e is lower than that of the whole training data D^d, render this division invalid and continue to train the model on D^d in the next iteration.

4   Experiments

To evaluate DBRF, we compared its performance on public datasets against several popular or related methods, and performed ablation test to determine the effects of the three proposed mechanisms for enhancement. We further applied visualization to demonstrate the effectiveness of DBRF.

In these experiments, we used the default parameters defined in our model and did not fine-tune them. The default number of decision trees in a random forest is 200; the default splitting criterion in a decision tree is Gini impurity; the default number of iterations is 10; the default quality evaluation criterion for leaf nodes is *score-f1*; the other default parameters have already been mentioned in the Section 3.

4.1   Comparison with Other Approaches

We used three datasets from the UCI Machine Learning Repository[1] in the comparison experiments, namely Adult, Letter and Yeast. We evaluated three versions of DBRF, namely DBRF-g, DBRF-e and DBRF-f, which uses *score-gini*, *score-entropy*, and *score-f1* as the quality evaluation criterion of leaf nodes respectively. For comparison, we chose two ensemble algorithms, GBDT [10] and RF [4], two related state-of-the-art approaches, gcForest [20] and sNDF [13], and multiplayer perceptron (MLP).

For fairer comparison, we compiled the open-source code of gcForest[2] published by Professor Zhou's team and used exactly the same method to split a dataset into training set and test set. The evaluation metric of the experiments is accuracy, as shown in Table 1. Numbers in italics are directly cited from either the gcForest paper [20] or the sNDF paper [13]. We used PyTorch to reproduce sNDF and recorded the results on both Adult and Yeast datasets. We used scikit-learn[3] library to evaluate GBDT.

Table 1 Comparison of test accuracy on UCI datasets.

	Adult	Letter	Yeast
sNDF	85.58%	*97.08%*	60.31%
gcForest	*86.40%*	97.12%	*63.45%*
MLP	*85.25%*	*95.70%*	*55.60%*
RF	*85.49%*	*96.50%*	*61.66%*
GBDT	86.34%	96.32%	60.98%
DBRF-g	86.57%	97.02%	63.68%
DBRF-e	86.56%	97.18%	**64.13%**
DBRF-f	**86.62%**	**97.25%**	63.90%

---

[1] `http://archive.ics.uci.edu/ml/index.php`
[2] `https://github.com/kingfengji/gcForest`
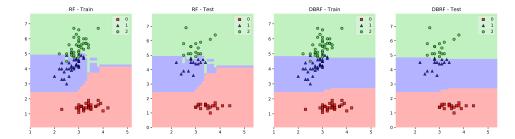[3] `http://scikit-learn.org/stable/`

As Table 1 shows, DBRF performs better than RF in all three datasets, which indicates that DBRF is indeed superior to RF. All DBRF versions achieve state-of-the-art results, proving evidence to the effectiveness of DBRF. Last but not least, the results in the last three rows indicate that *score-f1* may be the best fit for the quality evaluation criterion of leaf nodes, though not by a large margin over the other two.

4.2 Decision Regions Visualization

We visualized the decision boundaries learned by DBRF to verify that our proposed hard example mining method indeed achieves our intended purpose.

To facilitate visualization, we selected the second and third column of the UCI Iris dataset, and randomly splitted the dataset into 67% for training and 33% for testing. We conducted a comparison experiment between RF and DBRF using default parameters (for RF we used scikit-learn). RF achieves 98% accuracy on the training set but only 94% accuracy on the test set, which indicates overfitting. In contrast, though DBRF achieves 94% accuracy on the training set, it achieves a surprising 98% accuracy on the test set, suggesting that it has learned better decision boundaries. The decision boundaries learned by RF and DBRF are shown in Figure 4, which corroborates the above findings.

Figure 4 Visualization of decision boundaries. The two pictures on the left represent the RF's decision boundaries of the training set and test set, while the two pictures on the right represent the DBRF's decision boundary of the training set and test set, respectively.
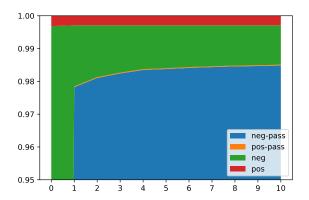

4.3 Imbalanced Data

We can better understand how our proposed hard example mining method makes DBRF more effective from a sampling point of view. Suppose we have a large dataset and the decision boundaries are determined by only a small proportion of the dataset. These are important data that ideally need to be preserved for training. However, traditional sampling methods such as random sampling used in RF have no clue whether a data point is important or not. Since they account for only a small amount, chances are that after sampling there are simply not enough important data left for a classifier to learn from. This is how a model may fail to learn the boundaries in spite of seemingly abundant data and features. Removing easy examples makes the proportion of important data higher, and thus makes it more likely that there will be sufficient amount of important data left after sampling. To the best of our knowledge, our proposed hard example mining method is a new way of sampling.

This is extremely useful when dealing with imbalanced datasets. We applied DBRF to Kaggle's classic imbalanced datasets, Credit Card Fraud Dataset[4] to demonstrate this point.

---

[4] `https://www.kaggle.com/mlg-ulb/creditcardfraud`

We randomly selected 65% of the dataset as training data and 35% as test data. Figure 5 shows how the proportion of positive and negative examples in the training data shifted after each iteration. We can see that in the initial training set (on the left of the x-axis), the data are extremely imbalanced with a negative-to-positive ratio of 565.38:1. As the iteration goes on, more data are removed from the training set and at the end (on the right of the x-axis), data are more balanced in the remaining hard examples for training with a negative-to-positive ratio of 3.62:1.

Figure 5 Stacked training data distribution in each iteration. The blue area (neg-pass) and the yellow area (pos-pass) at the bottom represent the proportions of negative and positive examples that were removed after each iteration. The red area (pos) and green area (neg) at the top represent the proportions of positive and negative examples that were kept as hard examples after each iteration.

The whole process achieves the purpose of hard example mining and sampling at the same time. Essentially, it uses data's importance rather than its label as a guideline for sampling. Without much fine-tuning, DBRF achieves an AUC 97.55% on the test set, a very competitive score.

4.4   Running time

The three UCI datasets we used for evaluation differ in data size (1.5k, 20k, 50k), ranging from small to medium. To further consolidate our claim that the proposed model DBRF is superior to RF and can achieve state-of-the-art result while still being computational efficient, we conducted a series of more experiments on Kaggle's Credit Card Fraud dataset mentioned in section 4.3. It has a considerably larger data size of 280k, and is extremely imbalanced since only about 400 data have positive labels. Thus, we have to use AUC instead of accuracy as the evaluation metric. We also reported standard deviation and computation time where necessary.

We compared our DBRF model against several closely related methods in terms of AUC and training time. As the original data are given in time order, we performed shuffling before randomly splitting the data into training set and test set with a ratio of 2:1. To be more credible, we ran the experiments on three different data splits, and for each split we ran each method five times and recorded means and standard deviations etc.

For RF-based models (gcForest, RF and DBRF), we grew 200 trees in a random forest per iteration, considering there are 30 features in the dataset. For iterative models (sNDF, gcForest, DBRF and GBDT), we set iteration times as 10 except for GBDT, for we found that the validation AUC of the first three models didn't change much after roughly 10 iterations. And since after 10 iterations, GBDT still performed not well enough so we increased it to

200, when the validation AUC stabilized. All the training time reported were recorded at the end of maximum iterations. For MLP or deep neural network configurations, we used ReLU for activation function, cross-entropy for loss function, Adam for optimization. Via three-fold cross-validation, we set the MLP with 3 hidden layers, 200 neurons in the first layer, 100 in the second and 10 in the third. The other hyper-parameters were set as their default value.

Table 2 Comparison of AUC and running time on Credit Card Fraud dataset on three data splits.

| | AUC(Test) | | Time(s) | |
	Avg.	Std.$(\times 10^{-2})$	Avg.	Std.
sNDF	0.9348/0.9290/0.9264	1.2849/1.3254/1.1094	712.4/717/724.4	20.9/34.0/24.5
gcForest	0.9580/0.9702/0.9548	0.2208/0.3660/0.1673	161/152.2/148.4	8.06/8.11/4.50
MLP	0.9172/0.8423/0.9112	1.9070/4.4489/1.9399	51.7/53.8/62.2	9.98/8.73/8.89
RF	0.9508/0.9598/0.9478	0.3050/0.5299/0.2218	61.2/61.6/61.54	1.42/1.84/1.43
GBDT	0.9434/0.8829/0.7259	0.0009/0.0032/0.0063	147.4/141.6/135.2	8.57/6.71/3.06
DBRF	**0.9762/0.9848/0.9730**	0.0293/0.0242/0.0117	97.4/81/77.80	5.73/2.83/0.75

From Table 2 one can see that DBRF performs consistently better than all other methods in terms of AUC on test set on all three data splits. Besides, the training time of DBRF is quite competitive. In particular, it is more computationally efficient than sNDF and gcForest.

We also want to know how the computation time and the size of training data change over each iteration. We recorded computation time and the proportion of easy examples that are removed at each iteration during both training and testing process on the first data split. The results are shown in Table 3. For comparison, RF trains in 58.2s and costs 0.33s in testing on average.

Table 3 Computation time and the proportion of easy examples of the first three iteration of DBRF on the first data split.

		Iter.1	Iter.2	Iter.3	All
Time	Train	59.18s	1.05s	0.91s	61.14s
	Test	2.86s	0.16s	0.16s	3.18s
Pass rate	Train	98.08%	0.16%	0.12%	98.36%
	Test	98.05%	0.15%	0.11%	98.31%

Table 4 shows that after the first iteration of DBRF, if we train other methods on the remaining training data, their overall test AUC all increased, which indicates the effectiveness of DBRF. Table 5 records the test AUC of RF and DBRF after three iterations when the number of trees in a random forest is fixed. It shows that just like neural nets, depth is also beneficial to RF.

Table 4 Comparison of AUC of other methods trained alone and on top of the first iteration of DBRF.

	MLP	RF	GDBT	sNDF	gcForest
alone	91.72%	95.08%	94.34%	93.48%	95.80%
+DBRF	95.98%	97.40%	97.38%	96.69%	97.37%

■ **Table 5** Comparison of AUC between RF and DBRF after three iterations when number of trees per forest is fixed.

	17	50	150	450	1350	4050
RF	-	93.86%	94.58%	96.59%	96.60%	96.67%
DBRF	97.21%	97.25%	97.57%	97.62%	97.83%	-

5 Related Work

Ensemble learning [19] is a powerful machine learning technique in which multiple learners are trained to solve the same problem. RF [4], GBDT [10], XGBoost [7] are paradigms of ensemble learning, which are all tree-based learning algorithms. DBRF is a novel ensemble algorithm in that it incorporates boosting into the training process of RF. AdaBoost [9] iteratively adds weight to the wrongly classified examples so as to focus training on these hard examples. In contrast, DBRF removes easy examples in each iteration to train on hard examples, which can effectively avoid the effects of unconcerned data while achieving similar boosting effect.

Deep learning research community is also resorting to the strength of tree models. Kontschieder et al. proposed deep neural decision forests [13], which is a novel approach that unifies decision trees with the representation learning known from deep convolutional networks, by training them in an end-to-end manner. In contrary, we attempt to make RF deep without training through back propagation and hyper-parameter tuning.

DBRF uses RF as the base learner. The capacity of RF is extended vertically by iteratively mining and training on hard examples. Zhou proposed gcForest [20, 8] has a cascade procedure similar to DBRF, but the specific approach is different. GcForest cascades the base learner by passing the output of one level of learners as input to the next level, which is similar to Stacking [18, 3]. DBRF cascades the base learner by using the quality of leaf nodes in one level as a criterion to dynamically evolve the training data to train the next level (model of the next iteration).

6 Conclusion

In this paper, to overcome some of the limitations of Random Forest (RF), we incorporate boosting into the training process of RF and propose a novel dynamic boosted ensemble learning method based on random forest (DBRF). Specifically, we propose a criterion to measure the quality of a leaf node in a decision tree and then vote to remove easy examples. By iteratively mining and training on hard examples, we evolve the model to learn decision boundaries better and thus extend RF vertically. We also propose evolution mechanism and smart iteration mechanism to enhance DBRF. Experiments show that DBRF outperforms RF and achieves state-of-the-art results. We also provide an explanation of its effectiveness from a sampling point of view.

We believe that DBRF is a very practical approach and is particularly useful in learning from imbalanced data. In future work, we plan to extend DBRF to learn from raw feature data such as images and sequences.

References

1 Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

2 W Banzhaf, P Nordin, RE Keller, and FD Francone. Genetic programming: An introduction: On the automatic evolution of computer programs and its applications. dpunkt–verlag fur digitale technologie gbmh and morgan kaufmann publishers. *Inc., Heidelberg and San Francisco CA, resp*, 1998.

3 Leo Breiman. Stacked regressions. *Machine learning*, 24(1):49–64, 1996.

4 Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

5 Leo Breiman. *Classification and regression trees*. Routledge, 2017.

6 Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

7 Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

8 Ji Feng and Zhi-Hua Zhou. Autoencoder by forest. *arXiv preprint arXiv:1709.09018*, 2017.

9 Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

10 Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

11 Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

12 Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

13 Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 1467–1475. IEEE, 2015.

14 Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

15 Minne Li, Zhaoning Zhang, Hao Yu, Xinyuan Chen, and Dongsheng Li. S-ohem: Stratified online hard example mining for object detection. In *CCF Chinese Conference on Computer Vision*, pages 166–177. Springer, 2017.

16 Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 761–769, 2016.

17 Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. *arXiv preprint arXiv:1704.03414*, 2, 2017.

18 David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

19 Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

20 Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. *arXiv preprint arXiv:1702.08835*, 2017.