Improving Deep Forest by Screening

Ming Pang, Kai Ming Ting, Peng Zhao, and Zhi-Hua Zhou, *Fellow, IEEE*

Abstract—Most studies about deep learning are based on neural network models, where many layers of parameterized nonlinear differentiable modules are trained by backpropagation. Recently, it has been shown that deep learning can also be realized by non-differentiable modules without backpropagation training called deep forest. We identify that deep forest has high time costs and memory requirements—this has inhibited its use on large-scale datasets. In this paper, we propose a simple and effective approach with three main strategies for efficient learning of deep forest. First, it substantially reduces the number of instances that needs to be processed through redirecting instances having high predictive confidence straight to the final level for prediction, by-passing all the intermediate levels. Second, many non-informative features are screened out, and only the informative ones are used for learning at each level. Third, an unsupervised feature transformation procedure is proposed to replace the supervised multi-grained scanning procedure. Our theoretical analysis supports the proposed approach in varying the model complexity from low to high as the number of levels increases in deep forest. Experiments show that our approach achieves highly competitive predictive performance with reduced time cost and memory requirement by one to two orders of magnitude.

Index Terms—Ensemble methods, deep forest, confidence screening, feature screening

1 Introduction

Deep learning has achieved great success in various applications, particularly with visual and speech information [1], [2], [3]. Most studies about deep learning are based on neural network models, or more accurately, many layers of parameterized nonlinear differentiable modules that can be trained by backpropagation. By recognizing that the key of deep learning may lie in the layer-by-layer processing, in-model feature transformation and sufficient model complexity, Zhou and Feng [4] proposed a deep learning method named gcForest, which is realized by non-differentiable modules without backpropagation training.

Essentially, gcForest is a novel decision tree ensemble method with predictive accuracy highly competitive to deep neural networks in a broad range of tasks. Besides, gcForest is much easier to train because it has fewer hyper-parameters. It has been shown that gcForest can achieve high predictive accuracy on datasets across different domains by using almost the same settings of hyper-parameters. Another advantage is that the model complexity of gcForest can be determined automatically in a data-dependent way. In contrast, deep neural networks needs to determine the network architecture before training, and this may make deep models more complicated than necessary [5].

A deep forest ensemble with a cascade structure enables gcForest to do representation learning. In this cascade structure, each level consists of an ensemble of decision tree forests [6], [7], i.e., an ensemble of ensembles [8]. Each level receives augmented features as input, which is the output of its preceding level. For textual or structural data, gcForest further enhances its representational learning ability by a technique called multi-grained scanning.

Although the results in [5] suggest that larger models might tend to offer better accuracy, Zhou and Feng [4] have not tried larger models with a larger number of forests and trees (in each forest). This is because deep forest is limited by the high time cost and memory requirement.

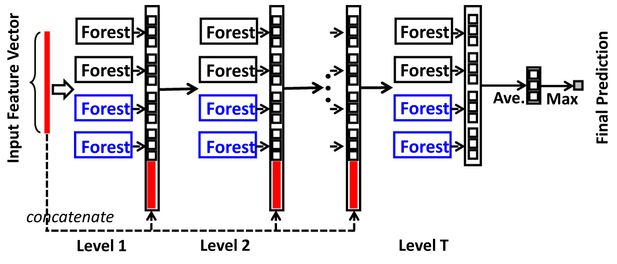
We identify that the main cause of this limitation owes much to two aspects. First, gcForest passes all instances through all levels of the cascade, and uses all features for learning, leading to a linear increase of time complexity as the number of levels increases. Second, multi-grained scanning usually converts one (original) instance into hundreds or even thousands of new instances, which significantly increases the number of training instances; and it also produces a high-dimensional input for the cascade procedure.

To address these issues, we introduce two screening mechanisms in the general framework of deep forest, with the aim to reduce time cost and memory requirement. First, *confidence screening* categorizes instances at every level of the cascade into two subsets: one is easy to predict; and the other is hard. If an instance is easy to predict, its final prediction is produced at the current level. Only if an instance is hard to predict, it needs to go through the next level (and potentially multiple levels after the next).

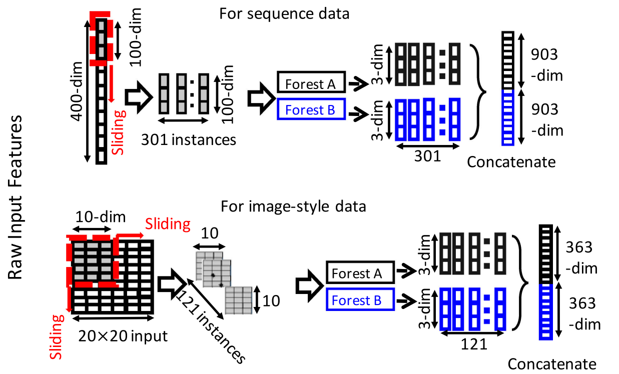
Second, *feature screening* categorizes the original features at every level of the cascade into two subsets: one is used for learning of the current level; and the other is reserved for the following levels. At each level, only the most informative features that are important for improving the prediction performance are selected for learning. Feature screening not only enhances the efficiency but also benefits the efficacy to a certain extent, in that it balances the attention to the augmented features and the original long feature vector.

• *The authors are with the National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: {pangm, tingkm, zhaop, zhouzh}@lamda.nju.edu.cn.*

Manuscript received 22 Dec. 2019; revised 28 Oct. 2020; accepted 3 Nov. 2020. Date of publication 17 Nov. 2020; date of current version 5 Aug. 2022.
(Corresponding author: Zhi-Hua Zhou.)
Recommended for acceptance by Benjamin C. M. Fung.
Digital Object Identifier no. 10.1109/TKDE.2020.3038799



(a) Cascade forest



(b) Feature re-representation using sliding window scanning

Fig. 1. gcForest: illustration of the cascade forest structure and feature re-representation using sliding window scanning [4], [5]. (a) Suppose each level of the cascade consists of two random forests (black) and two completely-random tree forests (blue). Suppose there are three classes to predict. Given an instance, each forest produces a 3-dimensional class vector. At each level (except the last one), these four class vectors are concatenated with the original feature vector to input to the next level. At the last level, all the four class vectors are averaged as the final class vector, and the class with the highest probability is the final prediction of the instance. (b) Suppose there are three classes, the raw features are 400-dim and the sliding window is 100-dim in the first row; the raw features are 20 $\times$ 20-dim and the sliding window is 10 $\times$ 10-dim in the second row.

In the multi-grained scanning procedure, subsampling the generated instances can reduce the number of training instances and the dimension of the transformed feature vector. However, a supervised feature transformation still requires a high time cost. Instead, we propose to replace the supervised multi-grained scanning procedure with an unsupervised version called *completely-random forest transformation*. It produces transformed features by constructing an unsupervised completely-random forest with the generated instances, which only has a linear time complexity.

Our theoretical analysis supports the proposed approach as the key means to vary the model complexity from low to high as the number of levels increases in deep forest. We use low model complexities at the first few levels, and increase the model complexity as the level increases. In other words, the high model complexity is only needed to produce an accurate model for hard-to-predict instances instead of all the instances. This variable model complexity mechanism further improves the efficiency of the first few levels.

In a nutshell, we propose an efficient deep forest called *gcForest*$_S$. For the cascade procedure, gcForest$_S$ adopts confidence screening and feature screening, coupled with a method to vary model complexity. For the multi-grained scanning procedure, gcForest$_S$ replaces the supervised feature transformation with an unsupervised one based on the completely-random forest. Our experiments show that gcForest$_S$ achieves predictive accuracy comparable to or better than gcForest, with one to two orders of magnitude lower memory requirement and faster runtime.

Note that we focus on deep learning with non-NN modules and aim to improve the efficiency of gcForest which is a seminal work towards non-NN deep models. Our work is a step advancement of gcForest by significantly reducing both memory requirements and time costs. This enables more widespread applications of deep forest and promotes the exploration of non-NN deep models on large-scale datasets that would otherwise be impossible.

The rest of this paper is organized as follows. Section 2 introduces deep forest and explains the reasons for its high cost of memory and time. Section 3 proposes our method gcForest$_S$. Section 4 presents the theoretical analysis of the confidence screening mechanism. Section 5 provides the discussion. Section 6 reports the empirical results. Section 7 studies the influence of confidence screening and feature screening. Section 8 concludes this paper.

2 Deep Forest

In this section we briefly introduce two key components of deep forest [4], namely, cascade forest structure and multi-grained scanning, and explain the reasons why these procedures increase the memory and time costs.

2.1 Cascade Forest Structure

Zhou and Feng [4] proposed gcForest with a cascade structure, as illustrated in Fig. 1a. In their structure, each level consists of an ensemble of decision tree forests, i.e., an ensemble of ensembles [8]. Each level of cascade receives feature information processed by its preceding level, and outputs its processing result to the next level. The processing result of one level is composed of class vectors generated by its forests. An example of the class vector generated by a forest is shown in Fig. 2. Given a test instance, each forest produces an estimate of class distribution by counting the percentage of different classes of training examples at the corresponding leaf node; and the final output is an average over the outputs from all trees in the same forest. The class vector produced by each forest can be generated by k-fold cross validation to reduce the risk of overfitting. The training procedure automatically terminates if there is no significant performance gain on the validation set.

A cascade forest can be formalized as follows. Consider the supervised learning problem of learning a mapping from the feature space $\mathcal{X}$ to the label space $\mathcal{Y}$, where $\mathcal{Y} = \{1, 2, \ldots, C\}$. Let $\mathcal{Z} = [0,1]^C$ and training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ be drawn independently and

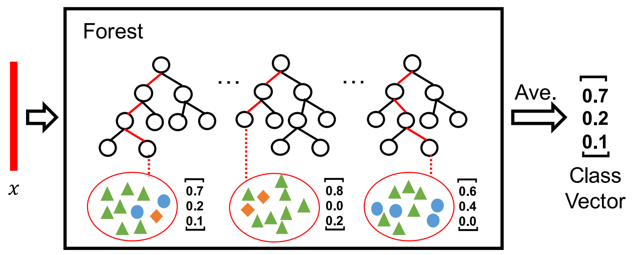


Fig. 2. Illustration of class vector generation. Each component of the final class vector is an average of the outputs of individual trees [4]. Different shapes in leaf nodes denote different classes.

identically from the underlying distribution $\mathcal{D}$. A cascade forest with T levels can be defined by a pair $(\mathbf{h}, \mathbf{f})$ where

- $\mathbf{h} = (h_1, \ldots, h_T)$, where h_t is the ensemble of forests at level t, and h_t is a member of hypothesis class H_t.
- $\mathbf{f} = (f_1, \ldots, f_T)$, where f_t is the cascade of ensembles of forests up to level t.

At level $t \in \{1, \ldots, T\}$, f_t: $\mathcal{X} \to \mathcal{Z}$ is defined as follows:

$$f_t(\mathbf{x}) = \begin{cases} h_1(\mathbf{x}) & t = 1, \\ h_t([\mathbf{x}, f_{t-1}(\mathbf{x})]) & t > 1. \end{cases} \tag{1}$$

At every level t, $h_t(\cdot)$ and $f_t(\cdot)$ output a class vector $[p_1^t, \ldots, p_C^t]$, where p_i is the prediction confidence of class i. The input of h_t is $[\mathbf{x}, f_{t-1}(\mathbf{x})]$ except that at level $t = 1$, its input is $\mathbf{x}$.

Each pair $(\mathbf{h}, \mathbf{f})$ defines a deep forest model $g : \mathcal{X} \to \mathcal{Y}$,

$$g(\mathbf{x}) = \underset{c \in \{1,\ldots,C\}}{\arg\max} [f_T(\mathbf{x})]_c, \tag{2}$$

where $[f_T(\mathbf{x})]_c$ is the cth element of the label vector $f_T(\mathbf{x})$.

Training an ensemble of forests is time-consuming and requires a lot of memory. The computational complexity of decision tree based methods is proportional to the number of samples and the dimension of features. For example, computational complexity of building a complete decision tree is $O(d \times m \log m)$; building a random forest takes $O(\# \text{ trees} \times d' \times m \log m)$ where d' (usually set as $\sqrt{d}$) is the number of features selected for each node.

Cascading many levels in gcForest is computationally expensive because every instance (in the training or testing set) is required to pass through all the levels, and all the original features are used for learning. The time complexity grows linearly with the number of levels in the cascade.

2.2 Multi-Grained Scanning

gcForest enhances cascade forest with a multi-grained scanning procedure. It is a feature representation method which transforms each instance represented with raw features to one represented with new features.

In multi-grained scanning, sliding windows are used to scan the raw features. As shown in the first example in Fig. 1b, one instance with 400 raw features is converted into 301 new instances of 100-dim by using a 100-dim sliding window. The second example demonstrates that one instance with 20×20 raw features (pixels of an image) is converted into 121 new instances of size 10×10 by using a 10×10 sliding window. This is the first part of the procedure; and it takes advantage of feature relationships (e.g., spatial or sequential feature relationships) and helps improve performance [4].

The second part of the procedure uses the converted instances to train a set of forests. For each converted instance, each forest outputs probabilistic prediction for each class. For each original instance, which has a set of converted instances, the prediction confidences of all these converted instances are concatenated to form a new instance. This new instance is the final output of multi-grained scanning; and it is used to train a cascade forest.

This aforementioned procedure increases the number of instances substantially. For example, suppose there are 10,000 images with 20×20 raw pixels. Then a 10×10 sliding window will produce a total of 1,210,000 instances of size 10×10, as the input of the new learning problem, whose scale is substantially larger than the original problem. This problem becomes even more severe when there are multiple sizes of sliding windows.

In addition, the transformed feature dimension can be much larger than the original raw input feature dimension. For example, as shown in Fig. 1b, there are 3 classes and 2 forests; then the transformed feature vector is 726-dim. If there are 10 classes and 8 forests; then the transformed feature vector is 9,680-dim. As a result, multi-grained scanning significantly increases the time cost and the memory requirement of the learning problem because of the dramatically increased numbers of instances and dimensions.

These above issues can be tackled by exploiting distributed implementation [9] or hardware facilitation. We believe, however, there is demand to tackle them via algorithmic improvement.

3 The Proposed Approach

In this section, we propose a framework for efficient learning of deep forest against the number of samples and the dimension of features, and design an efficient deep forest approach gcForest$_{\text{S}}$.

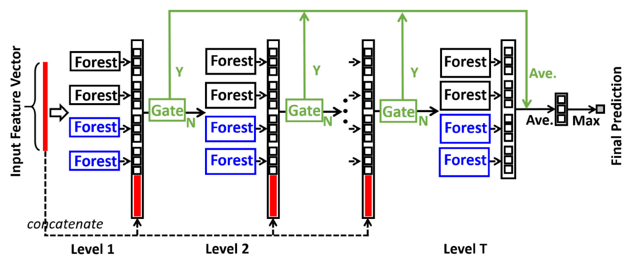
To deal with a large number of samples, we propose a *confidence screening* mechanism coupled with variable model complexity. To handle high-dimensional features, we design a *feature screening* mechanism. As for the multi-grained scanning procedure, an unsupervised *completely-random forest transformation* is proposed as a replacement in gcForest$_{\text{S}}$.

In order to study the influence of each element, we also use gcForest$_{\text{cs}}$ and gcForest$_{\text{fs}}$ for comparison, where gcForest$_{\text{cs}}$ has the confidence screening mechanism, and gcForest$_{\text{fs}}$ has the feature screening mechanism.

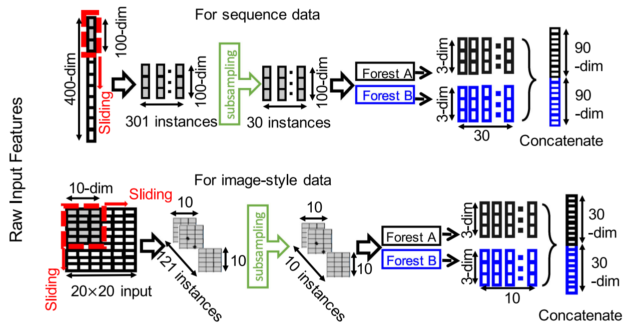
3.1 Confidence Screening

Rather than requiring all instances to go through all levels of cascade, we reduce the computation by only passing selective instances through the next level. The selection criterion is based on the prediction *confidence* which is the maximum value of the estimated class vector of one instance. For example, in a 3-class classification problem, as shown in Fig. 2, the estimated class vector of an instance is [0.7,0.2,0.1], then its prediction confidence is 0.7.

The basic idea is that an instance is pushed to the next level only if it is determined to require a higher level of



(a) Cascade forest with *confidence screening*



(b) Feature re-representation using sliding window scanning with *subsampling*

Fig. 3. gcForest$_{\text{CS}}$: illustration of the cascade forest structure with *confidence screening* and feature re-representation using sliding window scanning with *subsampling regime*. (a) Suppose there are two random forests (black) and two completely-random forests (blue) at each level of the cascade. For a three-class problem, each forest outputs a three-dimensional class vector, which is concatenated for re-representation of the original input. Instances with high prediction confidence (Y) at level i are predicted directly—they do not go through all the levels. Only instances with low prediction confidence (N) traverse to the next level. (b) Suppose there are three classes. Raw features are 400-dim, sliding window is 100-dim and subsampling size is 30 for the sequence data; raw features are 20 $\times$ 20-dim, sliding window is 10 $\times$ 10-dim and subsampling size is 10 for the image-style data.

learning; otherwise, it is predicted using the model at the current level. Based on this idea, we propose the deep forest structure with gates for confidence screening as shown in Fig. 3a.

There are two main differences in the cascade forest structure between gcForest$_{\text{CS}}$ and gcForest. First, gcForest$_{\text{CS}}$ has gates at each level to categorize instances into two subsets: one which is easy to predict; and the other is hard. As illustrated in Fig. 3a, instances with high prediction confidence (Y) at each level are predicted directly using the model at the current level; and only instances with low prediction confidence (N) are passed to the next cascade level for further improvement of the prediction. Second, as the level increases, gcForest$_{\text{CS}}$ uses more complex forests (e.g., more trees in each forest) for the remaining "hard" instances, while gcForest uses forests with the fixed setting at all levels. The effectiveness of the gcForest$_{\text{CS}}$ structure design is also verified from the theoretical view in Section 4.

A deep forest model with confidence screening can be defined by a triplet $(\mathbf{h}, \mathbf{f}, \boldsymbol{\kappa})$ where $\mathbf{h}$ and $\mathbf{f}$ are defined in the same way as in the cascade forest without confidence screening (before Eq. (1)), and $\boldsymbol{\kappa} = (\kappa_1, \ldots, \kappa_T)$ with κ_t as a screener function. Screener $\kappa_t(\mathbf{x}) = 1$ if $\mathbf{x}$ is predicted by f_t, and $\kappa_t(\mathbf{x}) = 0$ otherwise.

Screener $\kappa_t(\cdot)$ at level t is defined based on prediction confidence and confidence threshold η_t,

$$\kappa_t(\mathbf{x}) = \begin{cases} 1 & \|f_t(\mathbf{x})\|_\infty > \eta_t, \\ 0 & \text{otherwise}. \end{cases} \tag{3}$$

Let $\kappa_t^{-1}(1)$ denote the set of instances such that $\kappa_t(\mathbf{x}) = 1$, namely, the set of instances with high confidence at level t. $\kappa_t^{-1}(0)$ is similarly defined for those with low confidence.

At level t, if the prediction confidence of one instance is larger than threshold η_t, then its final prediction is produced at the current level; otherwise it needs to go through the next level (and potentially all levels in the cascade).

As aforementioned, the deep forest with confidence screening is defined by the triplet $(\mathbf{h}, \mathbf{f}, \boldsymbol{\kappa})$, and the final predictive function $g : \mathcal{X} \to \mathcal{Y}$ is defined as follows:

$$g(\mathbf{x}) = \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} \left[f_{t'}(\mathbf{x})\right]_c, \tag{4}$$

where $t' = \arg_{t \in \{1,\ldots,T\}} [\kappa_t(\mathbf{x}) = 1]$.

The key issue here is how to set the confidence threshold η_t at each level. In principle, one can define an optimization framework in which the confidence threshold at each level is set to trade off between minimizing the expected number of instances to be passed to the next level and maximizing the expected number of instances that can be corrected at the next level (which are misclassified at the current level). Unfortunately, finding such an optimum is quite difficult.

Instead, we use a simple rule to produce an effective cascade forest in a highly efficient way. The prediction confidence threshold η_t at level t is determined automatically based on the cross-validated error rate ϵ_t of all the training instances. Let hyper-parameter $a \in (0, 1)$ be a fraction of ϵ_t. All the training instances are sorted in the descending order of their prediction confidences, where c_i is the prediction confidence of $\mathbf{x}_i$. Then, the threshold η_t is set as follows:

$$\eta_t = \min\{c_k | L(\mathbf{x}_1, \ldots, \mathbf{x}_k) < a\epsilon_t,\ k \in \{1, , 2, \ldots, m\}\}, \tag{5}$$

where $L(\mathbf{x}_1, \ldots, \mathbf{x}_k) = \frac{1}{k}\sum_{i=1}^{k} \mathbb{1}[g_t(\mathbf{x}_i) \neq y_i]$ is the error rate of the k instances with the largest prediction confidences. Note that the coefficient a is the only additional hyper-parameter for confidence screening, compared with gcForest.

Variable Model Complexity. Because the remaining instances become increasingly hard-to-predict as the level increases, we use increasingly complex forests at high levels, i.e., increasing the number of trees in each forest linearly as the number of remaining instances decreases.

The above strategy follows the result of the theoretical analysis in Section 4. It suggests that varying the model complexity from low to high as the level increases in the cascade can lead to better generalization performance. This design further reduces memory requirement and time cost; and the model complexity only increases as the level increases when it is most needed to produce an accurate model for hard-to-predict instances.

Subsampling Multi-Grained Scanning. Multi-grained scanning in gcForest [4] increases the memory consumption and runtime heavily. To address this issue, as suggested in [4], gcForest$_{\text{CS}}$ uses a *subsampling* regime. Specifically, gcForest$_{\text{CS}}$ randomly samples from the set of the converted instances produced in multi-grained scanning. As Fig. 3b illustrates, subsampling not only reduces the number of converted instances, but also reduces the number of dimensions of the transformed features by an order of magnitude from 903 to 90. At each level, subsampling multi-grained scanning generates new transformed features, and the features from the recent three levels are concatenated to classify the remaining instances which are fewer and "harder".

We design the deep forest approach gcForest$_{\text{CS}}$ which has the key confidence screening mechanism coupled with variable model complexity and subsampling multi-grained scanning.[1] Algorithm 1 summarizes gcForest$_{\text{CS}}$.

Algorithm 1. gcForest$_{\text{cs}}$

```
Input: Training set $S$, validation set $S_v$, learning algorithm $\mathcal{A}$, confidence screening parameter $a$, and the maximal number of cascade levels $T$.
Output: The gcForest$_{\text{CS}}$ model $g$.
Initialize: $S_1 = S$, $\ell_0 = 1$ and $t = 1$
Process:
1:  while $t \leq T$ do
2:      $h_t = \mathcal{A}(S_t)$
3:      Get $f_t$ according to Eq. (1)
4:      Get $\kappa_t$ according to Eq. (3)
5:      Get $g_t$ according to Eq. (4)
6:      Compute the validation error $\ell_t = L_{S_v}(g_t)$
7:      if $\ell_t > \ell_{t-1}$ then
8:          return $g$
9:      end if
10:     $g = g_t$
11:     $S_{t+1} = S_t \setminus \kappa_t^{-1}(1)$
12:     $t = t + 1$
13: end while
14: return $g$
```

3.2 Feature Screening

In the cascade structure, each level of the cascade concatenates the original features with transformed features of its preceding level. Rather than using all the original features for learning at each level, we reduce the computation by selecting features that are important for improving the performance of the cascade.

The cascade forest with feature screening and confidence screening in gcForest$_{\text{S}}$ can be defined by a triplet $(\mathbf{h}, \mathbf{f}, \boldsymbol{\kappa})$ in a similar way to the cascade forest of gcForest$_{\text{CS}}$. The key difference is that gcForest$_{\text{S}}$ uses selective features instead of all at each level. Specifically, in gcForest$_{\text{S}}$, at level $t \in \{1, \ldots, T\}$, the predictive function f_t is defined as follows:

$$f_t(\mathbf{x}) = \begin{cases} h_1(\mathbf{z}_1) & t = 1, \\ h_t\big([\mathbf{z}_t, f_{t-1}(\mathbf{x})]\big) & t > 1, \end{cases} \tag{6}$$

where $\mathbf{z}_t$ is a subset of the original features $\mathbf{x}$ that is used by the ensemble of forests at level t.

Feature screening benefits deep forest in three aspects. First, a lower dimension of features reduces the time cost and the memory requirement. Second, screening irrelevant features increases the chance that relevant features are selected at each split of each decision tree. It can enhance the performance of each forest in case the fraction of relevant features is small [10]. Third, even if the high-dimensional original features are all relevant, a small number of augmented features are very likely to be drowned out in the original features [5]. Feature screening can balance between the augmented information and the original information, and may further improve the performance of deep forest.

At each level, the main goal of feature screening is to select features that are important for improving the performance. Note that tree-based methods are naturally suitable for calculating the relative importance of each feature [10]. Based on the importance ranking of an extra forest, a direct method is to select the top features, which is named *FS-rank*. At each level, FS-rank recalculates feature importances, and selects features with importances greater than a threshold $\tau \geq 0$, i.e.,

$$S = \{j \,|\, \mathcal{I}^j > \tau, j = 1, \ldots, d\},$$

where $\mathcal{I}^j$ is the importance of feature X_j.

In detail, for a single decision tree DT_n, the importance of feature X_j is calculated by

$$\mathcal{I}_n^j = \sum_{l=1}^{L} \hat{\iota}_l \mathbb{1}(v_l = j),$$

where L is the number of internal nodes of the tree. At node $l \in \{1, 2, \ldots, L\}$, feature X_{v_l} is used to split this internal node, and $\hat{\iota}_l$ is the estimated improvement of purity corresponding to the splitting. The importance of feature X_j is the sum of such improvements over all the internal nodes in which feature X_j is the splitting feature.

For a forest, the importance of feature X_j is simply averaged over trees

$$\mathcal{I}^j = \frac{1}{N} \sum_{n=1}^{N} \mathcal{I}_n^j.$$

Different kinds of forests are different in measuring estimated improvement $\hat{\iota}$ which is depends on their purity measurements. For classification trees, $\hat{\iota}$ is measured by Gini index; for regression trees, $\hat{\iota}$ is measured by squared error.

Although feature importances measure the prediction strength of each feature, FS-rank has two main issues that affect the efficiency and efficacy of feature screening. First, it takes a high cost of computation to determine the threshold τ. For each candidate value of τ, one needs to remove features with smaller importances, and refit the classifier to assess its performance. Second, the greedy selection strategy does not consider the redundancy of features. The remaining features with large importances may contain redundancy [6]. Given a selection size budget, selecting redundant features means informative features with smaller

1. Note that subsampling multi-grained scanning can be used to provide a new representation at each level.

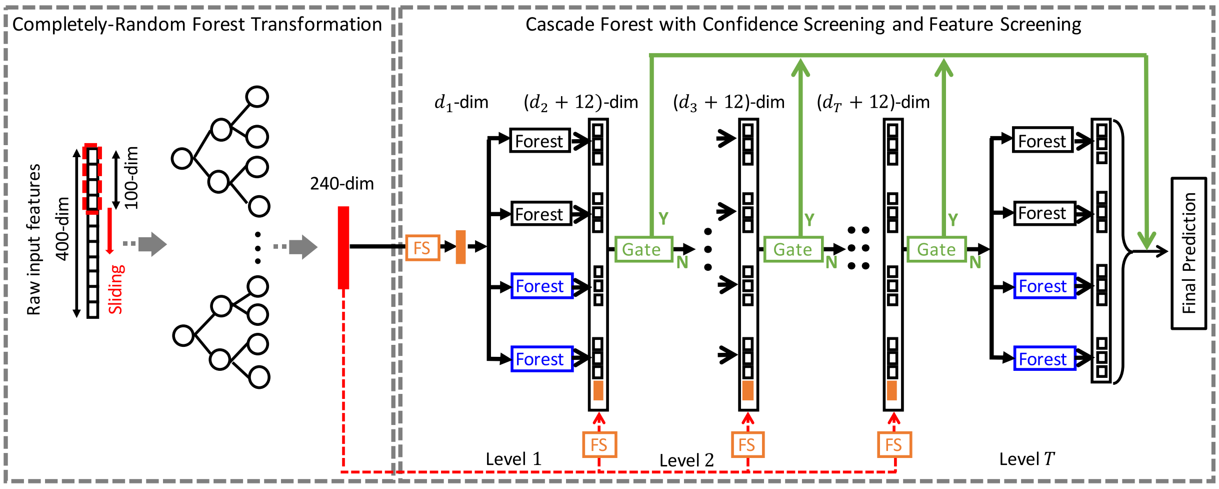


Fig. 4. gcForest$_S$: illustration of the overall procedure. Suppose there are three classes to predict, raw features are 400-dim, sliding window is 100-dim, the re-represented features of completely-random forest transformation are 240-dim. Feature screening is abbreviated as FS and d_i is the selection size of the features at level i.

importances are discarded which may degrade the prediction performance.

In this paper, we take both feature redundancy and nonlinear feature importances into consideration, and propose a feature screening method named *FS-reg*. FS-reg can be formulated as follows.

$$\hat{\beta} = \arg\min_{\beta} \left\| y - \sum_{j=1}^{d} \mathbf{x}_j \beta_j \right\|_2^2 + \lambda \sum_{j=1}^{d} (\mathcal{I}^{\max} - \mathcal{I}^j)|\beta_j|, \tag{7}$$

where $\mathcal{I}^{\max}$ is the maximum value of $\{\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^d\}$ and $\lambda \geq 0$. The objective function aims to minimize the squared loss with a weighted l_1 regularization.

We solve this optimization problem via the least angle regression which can compute the solutions for all the values of λ extremely efficiently [11]. The computation complexity is in the same order as a single least squares. FS-reg selects the best solution among all the solutions with $\|\hat{\beta}\|_0$ less than the given selection size budget.

At each level of the cascade, gcForest$_S$ automatically selects features with non-zero coefficients of $\hat{\beta}$. Class vectors generated by FS-reg can also be used as augmented features. Feature importance values are updated at each level. At the first level, feature importance is calculated by classification trees where the learning target is the labels; at each of the following levels, feature importance is calculated by regression trees based on the remaining samples of this level where the learning target is the residual of the last level. Thus, Eq. (7) tends to select features that are important for improving the performance of deep forest. Besides, we name the deep forest with only feature screening as gcForest$_{\text{fs}}$.

3.3 Completely-Random Forest Transformation

In order to handle feature relationships, gcForest adds a plug-in module, multi-grained scanning, which results in high memory requirements and time costs. To address this issue, we replace the supervised multi-grained scanning procedure with an unsupervised one, named completely-random forest (CRF) transformation, which has linear time complexity with low memory requirements.

CRF transformation combines completely-random forest with a variant of random subspace for feature transformation. As shown in Fig. 5, sliding windows scan the raw features, and each instance $\mathbf{x}_i$ can generate r new instances $\{\mathbf{z}_i^1, \mathbf{z}_i^2, \ldots, \mathbf{z}_i^r\}$. A completely-random forest is constructed based on a dataset subsampling from these new instances. Because a completely-random tree generates a node partition by randomly selecting an attribute and then randomly selecting a split value between the maximum and minimum values of the selected attribute, the growth of completely-random forest does not require labels, and label information is only used to label leaf nodes for classification.

After getting u leaf nodes $\{l_1, l_2, \ldots, l_u\}$, we can use transformation function $q(\mathbf{z}^j, l_k)$ to get the transformed feature $D_{j,k} \in \mathbb{R}$ for each pair of instance $\mathbf{z}^j$ and leaf node l_k. Thus, for each original instance $\mathbf{x}$, we can get a real value matrix $D \in \mathbb{R}^{r \times u}$. In this work, we use the mean of instances in leaf node l_k, i.e., $\mathbf{c}_k$, to represent l_k; transformation function $q(\mathbf{z}^j, l_k) = \|\mathbf{z}^j - \mathbf{c}_k\|_2$.

Consider that the transformed feature vector with dimension $r \times u$ might be too long, we group the converted instances $\{\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^r\}$ into r_g groups $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{r_g}\}$, where each group contains r/r_g consecutive instances. Thus, we can get a smaller transformed feature $D' \in \mathbb{R}^{r_g \times u}$ with $D'_{j',k} = \sum_{j \in \mathcal{G}_{j'}} \|\mathbf{z}^j - \mathbf{c}_k\|_2$.

In a similar way, data with spatial relationships (e.g., image data), can also be processed by CRF transformation.

Fig. 4 summarizes the overall procedure of gcForest$_S$.[2] There are two main differences between gcForest$_S$ and gcForest$_{CS}$. First, for the cascade procedure, gcForest$_S$ combines feature screening with confidence screening where FS-reg selects a subset of features at each level. Second, to handle feature relationships, an unsupervised CRF transformation replaces the supervised multi-grained scanning.

2. Note that multiple window sizes can be used, and one window size is used here as a simplified example.

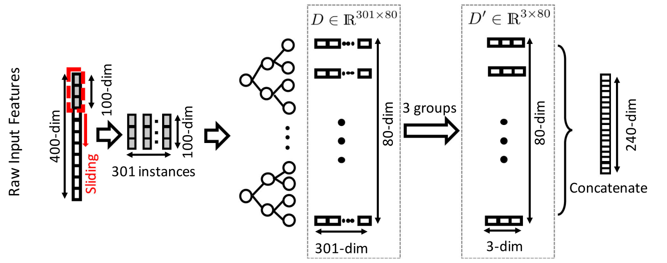


Fig. 5. Illustration of completely-random forest transformation. Suppose raw input features are 400-dim, sliding window is 100-dim, the forest has 20 trees, each tree has 4 leaves, and group size is 3.

4 Theoretical Analysis

In this section, we provide theoretical justifications for both confidence screening and variable model complexity. In particular, we are interested in the effect of splitting all instances into two parts according to prediction confidence, and thus, we ignore the concatenation of previous label predictions in the analysis. Specifically, we perform the generalization analysis, based on the notion of *Rademacher Complexity* [12].

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be the training sample set of size m drawn independently according to underlying distribution $\mathcal{D}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is a training instance and $y_i \in \mathcal{Y} = \{1, 2, \ldots, C\}$ is the associated class label. Besides, denote by $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ the loss function. For any hypothesis h, the *expected risk* is defined over the underlying distribution $\mathcal{D}$,

$$R(h) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(h(\mathbf{x}), y)],$$

whose empirical version called *empirical risk* is defined over the training sample set S,

$$\hat{R}_S(h) = \frac{1}{m}\sum_{i=1}^m \ell(h(\mathbf{x}_i), y_i).$$

In the following, we will utilize the notion of Rademacher complexity to measure the hypothesis complexity and then use it to establish the generalization error bounds.

Definition 1 (Rademacher Complexity [12]). *Let $\mathcal{G}$ be a family of functions and a fixed sample of size m as $S = \{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$, where $\mathbf{z}_i = (\mathbf{x}_i, y_i)$. Then, the* empirical Rademacher complexity *of $\mathcal{G}$ with respect to the sample S is defined as*

$$\hat{\mathfrak{R}}_S(\mathcal{G}) = \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^m \sigma_i g(\mathbf{z}_i)\right].$$

where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)^\top$, with σ_is independent uniform random variables taking values in $\{-1, +1\}$.

Besides, the Rademacher complexity *of $\mathcal{G}$ is the expectation of the empirical Rademacher complexity over all samples of size m drawn according to $\mathcal{D}$*

$$\mathfrak{R}_m(\mathcal{G}) = \mathbb{E}_{S\sim\mathcal{D}^m}[\hat{\mathfrak{R}}_S(\mathcal{G})]. \tag{8}$$

The illustration of cascade structure is plotted in Fig. 6. Suppose the cascade level is T, and denote the corresponding classifier as $\mathbf{h} = (h_1, \ldots, h_T)$ with $h_t : \mathcal{X} \to [-1, +1]$ (or $\{-1, +1\}$) the classifier at level t, which belongs to the hypothesis set $\mathcal{H}_t$, where $t = 1, \ldots, T$. We denote by Cascade_T the set of all the feasible classifier $\mathbf{h}$. Note that we only consider the binary scenario here for simplicity.

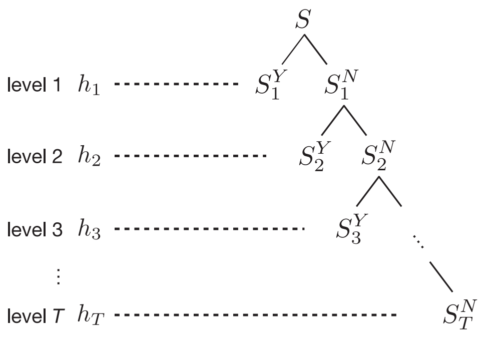


Fig. 6. The structure of the cascade forest with confidence screening. There is a total of T levels, where the instances are split into two parts at each level: the left part contains instances with high confidence, and the right part contains those with low confidence and thus needs to be further processed at higher levels.

Furthermore, we denote by

$$\mathscr{H}_t = \{\mathbf{x} \to s_t(\mathbf{x})h_t(\mathbf{x}) : s_t \in \mathcal{S}_t, h_t \in \mathcal{H}_t\},$$

the family of the products of the screener function and the classifier at level t. Then we have the following results regarding the generalization bound of the learned model.

Theorem 1. *Fix $\rho > 0$. Assume that the function in $\mathcal{H}_t$ takes values in $[-1, +1]$ for all the level $t \in \{1, \ldots, T\}$, and the training sample set S is of size m drawn independently and identically from the underlying distribution $\mathcal{D}$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, the following holds for all $h \in \text{Cascade}_T$:*

$$R(h) \le \hat{R}_S(h) + \sum_{t=1}^T \min\left\{4\hat{\mathfrak{R}}_S(\mathscr{H}_t), \frac{m_t}{m}\right\} + C(m,\rho) + \min_{\mathcal{I}\subseteq\mathcal{T}, |\mathcal{I}|\ge|\mathcal{T}|-1/\rho} \sum_{k\in\mathcal{I}}\left(\frac{m_t}{m} - 4\hat{\mathfrak{R}}_S(\mathscr{H}_t)\right) + \sqrt{\frac{\log(4/\delta)}{2m}},$$

where

$$C(m,\rho) = \frac{2}{\rho}\sqrt{\frac{\log T}{m}} + \sqrt{\frac{\log T}{\rho^2 m}\log\left(\frac{\rho^2 m}{\log T}\right)}, \tag{9}$$

and m_t is the number of screened instances at level t, i.e., $m_t = |S_t^Y|$. Moreover, $\mathcal{T}$ is the set of level t whose screening ratio is greater than $4\hat{\mathfrak{R}}_S(\mathscr{H}_t)$, namely, $\mathcal{T} = \{k : m_t/m > 4\hat{\mathfrak{R}}_S(\mathscr{H}_t)\}$.

To simplify the presentation, we ignore the non-leading terms and only keep the terms regarding cascade level T, instance number m and Rademacher complexity terms, and provide the following simpler form of the result:

$$R(h) \le \hat{R}_S(h) + \sum_{t=1}^T \min\left\{4\hat{\mathfrak{R}}_S(\mathscr{H}_t), \frac{m_t}{m}\right\} + \widetilde{O}\left(T\sqrt{\frac{\log T}{m}}\right).$$

Proof. The proof mainly has two steps, first introducing the convex ensembles with multiple hypothesis set, and then relating the deep cascade structure to the convex ensembles.

First, we introduce the convex ensembles with multiple hypothesis set $g_{\boldsymbol{\alpha}}$. For any $\boldsymbol{\alpha} \in \Delta_T$, denote $g_{\boldsymbol{\alpha}}$ as follows,

$$g_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t s_t(\mathbf{x}) h_t(\mathbf{x}),$$

where Δ_T is the simplex in $\mathbb{R}^T$. Since $g_{\boldsymbol{\alpha}}$ is a convex combination of the mappings $\mathbf{x} \rightarrow s_t(\mathbf{x})h_t(\mathbf{x})$, we can apply the result of Theorem 1 in [13] and obtain that

$$\begin{aligned} R(h) \quad & \leq \inf_{\boldsymbol{\alpha}\in\Delta_T} \left[\hat{R}_{S,\rho}(g_{\boldsymbol{\alpha}}) + \tfrac{4}{\rho} \textstyle\sum_{t=1}^{T} \alpha_t \hat{\mathfrak{R}}_S(\mathscr{H}_t) \right] \\ & \quad + C(m,p) + \sqrt{\tfrac{\log(4/\delta)}{2m}}, \end{aligned} \tag{10}$$

where $C(m,\rho)$ is defined in Eq. (9) and

$$\hat{R}_{S,\rho}(g_{\boldsymbol{\alpha}}) = \frac{1}{m} \sum_{t=1}^{T} \sum_{s(\mathbf{x}_i,k)=1} \mathbb{I}[y_i \alpha_t h_t(\mathbf{x}_i) \ < \ \rho].$$

Here, $\mathbb{I}[\cdot]$ is the indicator function.

The second step is to provide the upper bound for the first term in r.h.s. of Eq. (10). Following the analysis in [14], it can be bounded by,

$$\begin{aligned} & \inf_{\boldsymbol{\alpha}\in\Delta_T} \left[\hat{R}_{S,\rho}(g_{\boldsymbol{\alpha}}) + \tfrac{4}{\rho} \textstyle\sum_{t=1}^{T} \alpha_t \hat{\mathfrak{R}}_S(\mathscr{H}_t) \right] \\ & \leq \hat{R}_S(h) + \textstyle\sum_{t=1}^{T} \min\left\{ 4\hat{\mathfrak{R}}_S(\mathscr{H}_t), \tfrac{m_t}{m} \right\} \\ & + \min_{\mathcal{I}\subseteq\mathcal{T}, |\mathcal{I}|\geq|\mathcal{T}|-1/\rho} \textstyle\sum_{t\in\mathcal{I}} \left(\tfrac{m_t}{m} - 4\hat{\mathfrak{R}}_S(\mathscr{H}_t) \right). \end{aligned} \tag{11}$$

Hence, we complete the proof of the statement by combining (10) and (11). □

Remark 1. Our goal is to design efficient deep forests with better generalization ability. To deal with a large number of instances efficiently, we design a confidence screening mechanism. Because most of the instances will be screened at the first few levels, the screening ratio m_t/m is large when t is small. Note that the second term of the generalization error bound in Theorem 1 is the minimization of screening ratio m_t/m and complexity term $4\hat{\mathfrak{R}}_S(\mathscr{H}_t)$. Instead of using models with the same complexity at each level, this term indicates that one should reduce the corresponding complexity term at the first few levels in order to make the generalization error bound tighter. This is the theoretical basis in which we vary the model complexity in the cascade from low to high (by increasing the ensemble size) as the level t increases.

5 Discussion

The cascade procedure with confidence screening has some connections with two lines of research. First, confidence screening is related to boosted cascade [15] which aims to reject many negative instances and has achieved success in visual object detection problems. Although the cascade structure appears to be similar on the surface, the boosted cascade procedure is not suitable for classification tasks because the nature of the object detection tasks is different.

Second, there are some studies which add the output of one classifier as an additional input for another classifier in a series of multiple classifiers to improve the accuracy of a single classifier [16], [17]. Like gcForest, these methods pass all instances through all the classifiers which are inefficient.

Feature screening reduces the number of features used at each level, and more features are used for the low-confidence instances. It is related to the time-efficient feature extraction approaches [18], [19], [20], [21]. For test instances, these approaches only extract cheap and sufficient features, and when the classification confidence is high enough, the test instance will be classified. There are two main differences between these works and ours. First, their goal is to reduce the test time cost, while our goal is to reduce both the training and test time cost of deep forest. Second, the main concern is the time cost of feature extraction in their setting, while all the features are provided in deep forest and the main problem is to improve the efficiency of the learning process.

Recently, Extra-PCTs [22] was proposed and extended extremely randomized trees to structured output prediction (SOP) problems. Consider that Extra-PCTs not only has the best prediction performance, it can also learn good feature rankings which makes feature screening feasible. It is possible to use Extra-PCTs as building blocks in our proposal to construct deep forest models for SOP problems.

There have been many sparse-learning based methods proposed for feature selection [23], [24], [25], [26]. The representative work is least angle regression [11], where the sparse regularization forces many feature coefficients to become smaller. In the cascade forest structure, we need to select informative features for improving the performance. Different from the related works, our feature screening method readjusts importance weights of features based on the feature importances (acquired from random forest).

The multi-grained scanning procedure, which uses multiple sliding windows to scan the raw features, is related to multi-resolution examination procedures [27]. One sliding window producing a set of instances from one training example is related to bag generators [28] of multi-instance learning [29], [30], [31]. For example, if applied to images, it can be regarded as the SB image bag generator [32].

The subsampling strategy is related to sampling strategies for bag-of-features image classification [33], [34]. By treating an image as a collection of independent patches, random sampling gives equal or better representative selection than the sophisticated multiscale interest operators. In this paper, we use a simple random sampling strategy for gcForest$_{\text{CS}}$ and show that it works in the context of deep forest. It is interesting to explore other sampling strategies [35], [36].

Completely-random forest transformation is related to random trees embedding [37], [38], while the latter transforms the original features into a sparse representation by using a one-hot encoding of all the leaves.

Stochastic discrimination (SD) [39] is a general method to combine components with weak discriminative power to construct strong classifiers. The stochastic discrimination theory studied the combination of different ways to partition feature spaces. In this perspective, completely-random forest transformation is related to SD, which increases its discriminative ability by combining various completely-random trees. Extreme learning machine [40] randomly assigns input weights for hidden nodes, and the input feature vectors can be viewed as being randomly projected onto different nodes; this is a random initialization of neural networks, like a random initialization of RBF neural networks, very different from decision tree growth in completely-random trees.

A comprehensive review of the relationship between deep forest and existing ensemble methods [8] is provided by Zhou and Feng [4], e.g., Boosting [41], Bagging as base learners for Boosting [42]; stacking [43], [44], [45], [46].

The cascade forest (casForest) structure is crucial for the layer-by-layer processing. Recently, Lyu *et al.* [47] gave a theoretical explanation about the success of casForest from the perspective of margin theory.

There have been quite a few works applying gcForest to different applications. Utkin and Ryabinin [48] modified gcForest and proposed a Siamese deep forest as an alternative to the Siamese neural networks to solve the metric learning tasks. Yang *et al.* [49] proposed MLDF which extended gcForest into multi-label learning by replacing the forest block with random forest of predictive clustering trees [50]. gcForest also achieved superior performance in many other tasks, e.g., software defect prediction [51], hyperspectral image classification [52], self-interacting proteins prediction [53] and so on. Because our target is to improve the efficiency of deep forest, our method can help to improve the efficiency of Siamese deep forest, MLDF and other modified applications of gcForest as well.

6 Experiments

Our work focuses on efficient learning of deep forest. The goal is to validate that gcForest$_{\text{S}}$ can achieve predictive accuracy comparable to or better than gcForest with much less memory and time cost. gcForest$_{\text{S}}$ contains three main elements: confidence screening, feature screening and completely-random forest transformation. To analyze the effect of each element, we also design gcForest$_{\text{CS}}$ and gcForest$_{\text{fS}}$ for comparison, which are based on confidence screening and feature screening, respectively.

The experiments are divided into two categories: with and without multi-grained scanning to examine the effects in these two scenarios.

6.1 Experimental Setup

Parameter Settings. In all experiments, gcForest, gcForest$_{\text{CS}}$, gcForest$_{\text{fS}}$ and gcForest$_{\text{S}}$ all use the *same* cascade structure. Every level consists of v random forests and v completely-random forests [7] in the experiments, where $v = 4$ with multi-grained scanning and $v = 1$ without. The class vector of each forest is generated by three-fold cross validation.

For gcForest, every forest has 500 trees which is its recommended setting [4]. For both gcForest$_{\text{CS}}$ and gcForest$_{\text{S}}$, each forest at the first level has w trees and the number of trees is increased linearly as the number of instances at the subsequent levels decreases, in the accordance to the result of the theoretical analysis. In the comparisons, $w = 100$ are used for both methods. Furthermore, in the experiments without multi-grained scanning, $w = 20, 50, 100$ to examine their effects on the efficacy of gcForest$_{\text{CS}}$. For gcForest$_{\text{fS}}$, all the settings are the same as gcForest$_{\text{S}}$, except that gcForest$_{\text{fS}}$ does not use confidence screening.

The number of cascade levels stops increasing when the current level does not improve the accuracy of the previous level for all these deep forests.

For gcForest$_{\text{CS}}$, the prediction confidence threshold η at each level is determined automatically according to Eq. (5). Hyper-parameter a is set according to a simple rule as follows. If experiments with multi-grained scanning, $a = 1/20$. Otherwise, a is set according to the training accuracy of the first level ϵ_1. If $\epsilon_1 > 90\%$, $a = 1/10$; otherwise, $a = 1/3$. For gcForest$_{\text{S}}$, we set the feature selection budget as $\lfloor d/2 \rfloor$. In other words, at most half of the original features are selected at each level. Feature importance is calculated by an extra random forest with 100 trees at each level. Consider that least angle regression produces a full solution path of Eq. (7) which contains the results for all the values of λ. It produces the estimated class vectors via a 3-fold cross validation. The other settings are the same as gcForest$_{\text{CS}}$.

In (subsampling) multi-grained scanning, gcForest uses three window sizes with sizes of $\lfloor d/16 \rfloor$, $\lfloor d/8 \rfloor$, $\lfloor d/4 \rfloor$; gcForest$_{\text{CS}}$ and gcForest$_{\text{S}}$ use one window size with size $\lfloor d/16 \rfloor$ for d raw features. For CRF transformation, the maximum depth of completely-random trees is set as 6; leaf nodes which contain more than $1/4$ of the instances or less than $1/400$ are discarded, since they are unrepresentative; the group size is set as 3 for sequential datasets and 4 for image datasets. By using both CRF transformation and multi-grained scanning (with restricted tree depths for efficiency), gcForest$_{\text{S}}$ can further improve the prediction performance on complex image datasets. Note that gcForest$_{\text{CS}}$ and gcForest$_{\text{S}}$ can surely adopt multiple window sizes, which might offer a better accuracy as suggested by Zhou and Feng [4]. Nevertheless, it is sufficient to use only one window size for gcForest$_{\text{CS}}$ and gcForest$_{\text{S}}$ to achieve a satisfying performance with even less time cost and memory requirement.

Datasets. Experiments are performed on *all* the datasets[3] used by gcForest, i.e., LETTER and ADULT from UCI repository, IMDB [54], MNIST [55], sEMG [56], CIFAR10 [57]. In addition, we also employ two high-dimensional datasets EPS20K and EPSILON.[4] The data characteristics of these datasets are summarized in Table 1.

Evaluation Metrics and Methodology. We adopt the predictive accuracy as the classification performance measurement which is suitable for these balanced datasets. Logloss is also used as a measurement criterion. Note that if deep forests with and without confidence screening have the same predicted label for an instance, the former replaces its predicted probabilities with the latter's so that logloss can better measure the influence of confidence screening. Training time, test time and memory usage are used to evaluate the efficiency. Experiments on the datasets sEMG, MNIST, CIFAR10 and EPSILON are evaluated on the given training

3. Three small datasets, i.e., ORL, GTZAN and YEAST are excluded which have little to do with our purpose of improving the efficiency.

4. https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html. EPS20K contains 20,000 samples randomly selected from EPSILON.

TABLE 1
Description of Datasets

Datasets	#train inst.	#test inst.	#dim	#label
sEMG	1,260	540	3,000	6
LETTER	14,000	6,000	16	26
EPS20K	14,000	6,000	2,000	2
ADULT	34,189	14,653	113	2
IMDB	35,000	15,000	5,000	2
CIFAR10	50,000	10,000	1,024	10
MNIST	60,000	10,000	784	10
EPSILON	400,000	100,000	2,000	2

and testing sets. On the other datasets, we conduct 10 independent runs.

Hardware. In the experiments without multi-grained scanning, we use a machine with 4×2.10 GHz CPUs and 32GB main memory. In the experiments with multi-grained scanning, we use a machine with 28×2.40 GHz CPUs and 756GB main memory. This is because 32 GB main memory is not enough for the multi-grained scanning procedure of gcForest (although gcForest$_{\text{CS}}$ and gcForest$_{\text{S}}$ has no barriers). An exception is the EPSILON dataset which is conducted on the latter device, since gcForest cannot get its results in 24 hours on the former device.

6.2 Results With Multi-Grained Scanning

The datasets sEMG, MNIST and CIFAR10 are used here because they hold spatial or sequential relationships among the raw features; and the other datasets do not. Deep forest methods are compared, and comparisons with state-of-the-art neural networks are included in the supplementary, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2020.3038799.

Table 2 shows that gcForest$_{\text{S}}$ improves the predictive accuracy with one to two orders of magnitude less memory and faster runtime. Notably, on the sEMG dataset, the accuracy is improved by more than 6 percent, and the runtime speedup is more than 500 times; on the CIFAR10 dataset, the accuracy is improved by more than 4 percent, and the runtime speedup is more than 20 times.

By using subsampling multi-grained scanning, gcForest$_{\text{CS}}$ also reduces the memory usage by an order of magnitude. However, the supervised feature transformation still requires a high time cost. By replacing the supervised procedure with CRF transformation, gcForest$_{\text{S}}$ substantially speeds up the training and testing of deep forest, while the predictive accuracies are also improved.

Interestingly, if gcForest adopts subsampling multi-grained scanning at each level (the same as gcForest$_{\text{CS}}$) instead of multi-grained scanning, its accuracy will degrade heavily, e.g., it achieves 67.78 percent on sEMG which is much lower than the original gcForest and gcForest$_{\text{CS}}$. Thus we report the results of the original gcForest only. This outcome further verifies the effectiveness of confidence screening.

We also compare eight combinations of the three major components, i.e., completely-random forest transformation, confidence screening and feature screening. The experiments are categorized into experiments with and without

TABLE 2
Comparison Results (With Multi-Grained Scanning) of gcForest, gcForest$_{\text{CS}}$ and gcForest$_{\text{S}}$ on Accuracy (in Percents), Training Time (in CPU Seconds), Test Time (in CPU Seconds) and Memory Usage (in Megabytes)

Datasets	Method	Accuracy (%)	Training time (s)	Test time (s)	Memory (MB)
sEMG	gcForest	71.30	34323.78	2288.29	41,789
	gcForest$_{\text{CS}}$	72.59	1547.62	77.48	4,348
	gcForest$_{\text{S}}$	77.41	61.61	3.85	3,239
MNIST	gcForest	99.26	27840.39	464.27	50,518
	gcForest$_{\text{CS}}$	99.26	1060.65	9.64	4,997
	gcForest$_{\text{S}}$	99.32	951.49	9.52	4,269
CIFAR10	gcForest	61.78	63068.32	2102.71	73,826
	gcForest$_{\text{CS}}$	62.62	13341.68	667.08	6,875
	gcForest$_{\text{S}}$	66.21	2051.38	91.85	6,458

TABLE 3
Accuracies (in Percents) of gcForest, gcForest$_{\text{CS}}$, gcForest$_{\text{fS}}$ and gcForest$_{\text{S}}$ With and Without Completely-Random Forest Transformation (CRF-trans)

Accuracy (%)	Method	sEMG	MNIST	CIFAR10
With CRF-trans	gcForest	75.74	98.89	65.03
	gcForest$_{\text{CS}}$	75.74	99.20	65.01
	gcForest$_{\text{fS}}$	76.67	98.96	65.31
	gcForest$_{\text{S}}$	77.41	99.32	66.21
Without CRF-trans	gcForest	48.15	98.02	52.03
	gcForest$_{\text{CS}}$	47.78	98.30	52.47
	gcForest$_{\text{fS}}$	50.93	98.25	53.41
	gcForest$_{\text{S}}$	51.67	98.29	54.12

If CRF-trans is used, all methods use the same transformed features; otherwise, all methods use original features.

completely-random forest transformation (CRF-trans). In the experiments with CRF-trans, all methods use transformed features as input features. In the experiments without CRF-trans, all methods use original features as input features. The results in Table 3 validate that all the three major components and their combinations help improve performance.

The CRF transformation enables gcForest and gcForest$_{\text{CS}}$ to achieve much better accuracy than the original gcForest and gcForest$_{\text{CS}}$ using the supervised procedure on both sEMG (e.g., from 71.30 to 75.74 percent in gcForest) and CIFAR10 (e.g., from 61.78 to 65.03 percent in gcForest). This is despite the fact that the accuracy degrades slightly on MNIST (e.g., from 99.26 to 98.89 percent in gcForest). These outcomes show that the CRF transformation not only improves the efficiency, it may also enhances the feature re-representation process.

6.3 Results Without Multi-Grained Scanning

In this part, we conduct experiments without multi-grained scanning on the datasets that do not hold spatial or sequential relationships among the raw features. Deep forest methods are compared, and comparisons with other classical methods are included in the supplementary, available online.

Table 4 shows that gcForest$_{\text{CS}}$(100) achieves accuracies and logloss comparable to or better than gcForest on all five

TABLE 4
Comparison Results (Without Multi-Grained Scanning) of gcForest, gcForest$_{CS}$, gcForest$_{fS}$ and gcForest$_{S}$ on Accuracy (in Percents), Logloss, Training Time (in CPU Seconds), Test Time (in CPU Seconds) and Memory Usage (in megabytes)

Datasets	Method	Accuracy (%)	Logloss	Training time (s)	Test time (s)	Memory (MB)
LETTER	gcForest	97.08 ± 0.25	0.124 ± 0.006	86.42	3.17	4526
	gcForest$_{CS}$(20)	96.42 ± 0.35	0.125 ± 0.006	13.39	0.41	206
	gcForest$_{CS}$(50)	96.93 ± 0.36	0.120 ± 0.006	17.16	1.25	472
	gcForest$_{CS}$(100)	97.08 ± 0.32	0.117 ± 0.006	75.23	2.23	915
	gcForest$_{fS}$	97.04 ± 0.29	0.131 ± 0.007	66.89	2.81	2834
	gcForest$_{S}$	97.09 ± 0.41	0.130 ± 0.008	18.27	2.79	542
EPS20K	gcForest	83.17 ± 0.52	0.475 ± 0.004	4291.10	1538.60	2103
	gcForest$_{CS}$(20)	76.34 ± 1.35	0.498 ± 0.003	112.28	18.85	1587
	gcForest$_{CS}$(50)	83.31 ± 0.52	0.470 ± 0.003	266.82	59.43	1634
	gcForest$_{CS}$(100)	83.52 ± 0.53	0.470 ± 0.004	576.76	179.10	1684
	gcForest$_{fS}$	86.11 ± 0.24	0.334 ± 0.009	219.37	43.42	1572
	gcForest$_{S}$	86.52 ± 0.30	0.338 ± 0.010	191.99	52.19	1421
ADULT	gcForest	86.06 ± 0.17	0.312 ± 0.002	198.85	12.24	3002
	gcForest$_{CS}$(20)	86.04 ± 0.08	0.317 ± 0.008	27.47	2.18	173
	gcForest$_{CS}$(50)	86.04 ± 0.20	0.322 ± 0.006	45.19	4.12	351
	gcForest$_{CS}$(100)	86.11 ± 0.14	0.316 ± 0.003	95.48	6.86	648
	gcForest$_{fS}$	86.24 ± 0.13	0.302 ± 0.006	53.31	5.45	1322
	gcForest$_{S}$	86.30 ± 0.15	0.312 ± 0.005	51.72	2.71	605
IMDB	gcForest	89.20 ± 0.29	0.305 ± 0.005	11633.65	152.2	3750
	gcForest$_{CS}$(20)	89.19 ± 0.23	0.304 ± 0.004	590.79	16.63	1518
	gcForest$_{CS}$(50)	89.40 ± 0.23	0.302 ± 0.003	974.07	23.19	1802
	gcForest$_{CS}$(100)	89.57 ± 0.21	0.301 ± 0.004	1623.11	32.05	1992
	gcForest$_{fS}$	89.49 ± 0.26	0.281 ± 0.005	864.01	13.92	1755
	gcForest$_{S}$	89.56 ± 0.15	0.281 ± 0.004	582.92	15.85	1627
EPSILON	gcForest	86.45	0.382	50398.47	1161.60	27932
	gcForest$_{CS}$(20)	85.45	0.385	2244.89	118.01	7735
	gcForest$_{CS}$(50)	85.67	0.386	4401.69	134.44	9112
	gcForest$_{CS}$(100)	86.52	0.381	7836.05	150.57	11006
	gcForest$_{fS}$	88.77	0.277	3746.32	71.14	16427
	gcForest$_{S}$	89.17	0.276	2600.84	36.65	9929

EPSILON has the provided training and testing sets. On the other datasets, 10 test runs are conducted and the average accuracies as well as the standard deviations are presented.

datasets, while the memory usage and the runtime are both substantially reduced. The runtime speedup is about 6 times on EPS20K, IMDB and EPSILON. The memory improvement rate is about 5 times on LETTER and ADULT. This verifies the effectiveness of confidence screening.

Compared with gcForest, gcForest$_{fS}$ attains substantial improvement of efficiency. Especially on the high-dimensional datasets, i.e., EPS20K, IMDB and EPSILON, gcForest$_{fS}$ achieves better prediction performance with an order of magnitude smaller time costs. This verifies the effectiveness of feature screening.

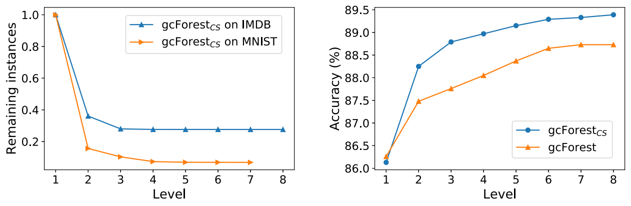
By combining confidence screening and feature screening, gcForest$_{S}$ further improves the prediction performance while reducing the memory usage and time cost. Notably, on EPS20K and EPSILON, the accuracy is improved by about 3 percent, while the test time speedup is about 30 times.

Since there are different requirements for the runtime and the memory usage on different applications and different devices, we need to adjust the number of trees of each forest to meet these requirements. The results in [5] show that the larger models with more trees tend to offer better performance. In other words, decreasing the number of trees of each forest may degrade the performance of deep forest. In our proposed method, the variable model complexity strategy increases the number of trees linearly as the number of remaining instances decreases, which means the computational complexity of each level is no more than the first level. To study the effect of variable model complexity against different requirements, we conduct experiments on gcForest$_{CS}$ with different numbers of trees of each forest at the first level, i.e., 20, 50 and 100.

As shown in Table 4, gcForest$_{CS}$ has close enough accuracies on these three settings; and the maximal gap among them is about 1 percent (except on EPS20K). In contrast, gcForest has a large accuracy gap. For example, gcForest(500) achieves 83.17, 89.20 and 86.45 percent on EPS20K, MNIST and EPSILON, respectively; while gcForest(20) achieves respective accuracies 73.79, 88.21 and 82.63 percent which has much larger accuracy gaps than gcForest$_{CS}$. Thus, when the runtime and memory usage are limited, variable model complexity combined with confidence screening leads to more robust performances than gcForest. These results validate the effectiveness of the variable model complexity strategy, which are also consistent with the theoretical analysis.

7 Influence of Screening Mechanisms

The effectiveness of confidence screening and feature screening have been validated in the last section. We further



(a) Ratio of remaining instances (b) Accuracy at each level

Fig. 7. The examples of gcForest$_{CS}$ and gcForest. (a) Ratio of remaining instances at each level of gcForest$_{CS}$ on IMDB and MNIST; (b) Test accuracy at each level of gcForest$_{CS}$ and gcForest on IMDB.

investigate the influence of these two screening mechanisms at each level of deep forest.

7.1 Influence of Confidence Screening

In this part, we aim to study the influence of confidence screening that links the decreasing number of instances with the improvement at each level in terms of accuracy, memory and runtime. The examples are the results of one test run based on IMDB and MNIST, and the results are similar on other datasets.

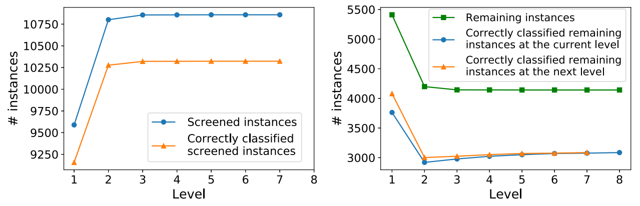
The number of instances to be screened is adaptive to the problem at hand. As shown in Fig. 7a, gcForest$_{CS}$ screens 64 and 84 percent of the test instances of IMDB and MNIST, respectively, at the first level.[5] At the last level, only 28 and 7 percent test instances of IMDB and MNIST, respectively, are passed through all the levels. In contrast, all the test instances are passed through all the levels in gcForest. The results are similar on training. This leads directly to the high improvement rates in terms of memory and runtime.

We demonstrate the test accuracy at each level of gcForest$_{CS}$ and gcForest on IMDB in Fig. 7b. Both gcForest$_{CS}$ and gcForest have 8 levels. At the first level, gcForest is slightly better than gcForest$_{CS}$ because gcForest uses more complex forests with more trees. After the first level, gcForest$_{CS}$ gets better accuracy.

We further demonstrate how confidence screening influences the classification result. As Fig. 8a shows, the accumulated number of screened instances increases as the number of levels increases. Because instances are screened if their predictions have high confidence, the accuracy of the screened instances is about 95 percent which is much higher than the overall accuracy ($<90\%$). Interestingly, gcForest$_{CS}$ and gcForest have the same predictions on the screened instances except two of them.

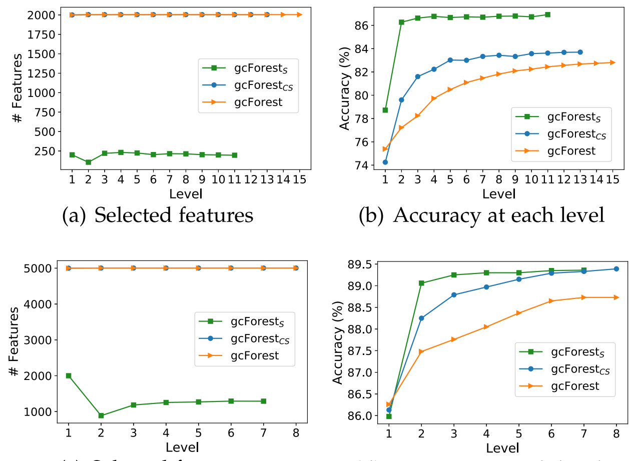
The result of the remaining instances is shown in Fig. 8b: the blue line plots the number of remaining instances which are correctly classified at the current level; and the orange line represents the number of those correctly classified at the next level. Fig. 8b shows that the accuracy of remaining instances increases when they are passed from the current level to the next level—due to the model used in the next level. At the final level, the number of correctly classified instances of gcForest$_{CS}$ is 80 instances more than that of gcForest. This outcome indicates that confidence screening

5. If none of the test instances have high enough confidences, gcForest$_{CS}$ lets all the instances pass to the next level.



(a) Screened instances (b) Remaining instances

Fig. 8. IMDB: Screened instances and remaining instances at each level of gcForest$_{CS}$. (a) The accumulated numbers of screened instances and correctly classified screened instances up to a certain level of gcForest$_{CS}$; (b) The number of remaining instances at each level of gcForest$_{CS}$. At each level, the remaining instances are passed to the next level. The number of correctly classified remaining instances increases from the current level to the next level.



(a) Selected features (b) Accuracy at each level

(c) Selected features (d) Accuracy at each level

Fig. 9. The examples of gcForest$_S$, gcForest$_{CS}$ and gcForest. (a, c) Number of features used at each level; (b, d) Test accuracy at each level of gcForest$_S$, gcForest$_{CS}$ and gcForest. The first row and the second row are based on EPS20K and IMDB, respectively.

encourages models at each level to better focus on the hard-to-predict instances that leads directly to their better predictions.

7.2 Influence of Feature Screening

In this part, we analyze the influence of feature screening that selects variational number of features at each level. The examples are the results of one test run based on EPS20K and IMDB, and the results are similar on other datasets.

The number of selected features is adaptive to each level of gcForest$_S$. At each level, gcForest$_S$ selects a subset of features that can bring in the most improvement. As shown in Fig. 9, gcForest$_S$ screens about 90 (70 percent) of the features at each level on EPS20K (IMDB), while gcForest and gcForest$_{CS}$ use all the features.

Feature screening brings two main benefits to deep forest. First, feature screening improves the efficiency of deep forest as fewer features are used. Second, feature screening emphasizes the importance of transformed features, especially when the original feature vector is much longer than the transformed feature vector. As shown in Fig. 9, by using fewer original features, it balances between the original features and the transformed features, which can make better use of the cascade structure and lead to better predictions.

TABLE 5
Accuracies of gcForest$_S$ With Different Strategies of Feature Screening Including FS-Rank, FS-Sparse and FS-Reg

gcForest$_S$ (%)	EPS20K	IMDB	EPSILON
FS-rank	85.35	89.00	87.89
FS-sparse	86.37	89.28	89.01
FS-reg	86.52	89.56	89.17

Experiments are conducted on three high-dimensional datasets, i.e., EPS20K, IMDB and EPSILON.

As discussed in Section 3.2, feature screening can be done in different strategies. We conduct experiments on three high-dimensional datasets to compare three strategies for feature screening. The first method, FS-rank, selects features with the highest feature importance directly. The second method, FS-sparse, selects features by a sparse regularization where each feature is treated equally. The third method, FS-reg, selects features by a reweighted sparse regularization where each feature is reweighted according to its feature importance. In this paper, we adopt FS-reg for feature screening, which considers both feature redundancy and non-linear feature importances. As shown in Table 5, gcForest$_S$ with FS-reg achieves the best prediction performance.

8 Conclusion

In this paper, we focus on improving the efficiency of deep forest, which extends our preliminary research [58]. We first identify two deficiencies of gcForest and then introduce two screening mechanisms and the unsupervised CRF transformation to address these issues. The confidence screening splits the instances into easy-to-predict and hard-to-predict subsets at each level, which substantially reduces the number of instances passed to the next level. The feature screening selects informative features, rather than using all features at each level, which considerably reduces the number of dimensions used for learning. The CRF transformation constructs an unsupervised completely-random forest, which has linear time complexity only. All three strategies significantly improve the efficiency of deep forest.

Our theoretical analysis supports the proposed approach in varying the model complexity from low to high as the number of levels increases in deep forest. Thus, the screening process is coupled with this variable model complexity mechanism, which substantially reduces the memory requirements and time costs of deep forest at the first few levels.

The effectiveness of the approach is validated in our evaluation that it achieves more with less, i.e., gcForest$_S$ has predictive accuracy comparable to or better than gcForest—this is achieved with one to two orders of magnitude smaller time costs and memory requirements.

gcForest$_S$ is a step advancement of gcForest that is instrumental in more widespread applications of deep forest and further explorations of non-NN deep models. An interesting future work is to incorporate gcForest$_S$ into the recently proposed *abductive learning* [59] paradigm to enable a more efficient connection between machine learning and logical reasoning since trees are more amenable to symbolic representation and with fewer theoretical mysteries [60] than neural networks.

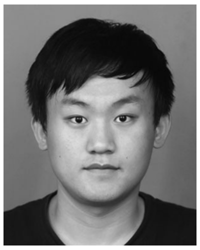
Acknowledgments

This research was supported by the National Science Foundation of China (61751306, 61921006) and the Collaborative Innovation Center of Novel Software Technology and Industrialization. Ming Pang was supported by the Outstanding PhD Candidate Program of Nanjing University.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
[2] G. Hinton *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[4] Z.-H. Zhou and J. Feng, “Deep forest: Towards an alternative to deep neural networks,” in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 3553–3559.
[5] Z.-H. Zhou and J. Feng, “Deep forest,” *Nat. Sci. Rev.*, vol. 6, no. 1, pp. 74–86, 2018.
[6] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[7] F. T. Liu, K. M. Ting, Y. Yu, and Z.-H. Zhou, “Spectrum of variable-random trees,” *J. Artif. Intell. Res.*, vol. 32, pp. 355–384, 2008.
[8] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Boca Raton, FL, USA: CRC Press, 2012.
[9] Y.-L. Zhang *et al.*, “Distributed deep forest and its application to automatic detection of cash-out fraud,” *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 5, 2019, Art. no. 55.
[10] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Berlin, Germany: Springer, 2009.
[11] B. Efron *et al.*, “Least angle regression,” *Ann. Statist.*, vol. 32, no. 2, pp. 407–499, 2004.
[12] P. L. Bartlett and S. Mendelson, “Rademacher and Gaussian complexities: Risk bounds and structural results,” *J. Mach. Learn. Res.*, vol. 3, pp. 463–482, 2002.
[13] C. Cortes, M. Mohri, and U. Syed, “Deep boosting,” in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 1179–1187.
[14] G. DeSalvo, M. Mohri, and U. Syed, “Learning with deep cascades,” in *Proc. Int. Conf. Algorithmic Learn. Theory*, 2015, pp. 254–269.
[15] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2001, pp. 511–518.
[16] J. Gama and P. Brazdil, “Cascade generalization,” *Mach. Learn.*, vol. 41, no. 3, pp. 315–343, 2000.
[17] H. Zhao and S. Ram, “Constrained cascade generalization of decision trees,” *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 6, pp. 727–739, Jun. 2004.
[18] L.-P. Liu, Y. Yu, Y. Jiang, and Z.-H. Zhou, “TEFE: A time-efficient approach to feature extraction,” in *Proc. 8th IEEE Int. Conf. Data Mining*, 2008, pp. 423–432.
[19] Z. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle, “Classifier cascades and trees for minimizing feature evaluation cost,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 2113–2144, 2014.
[20] F. Nan and V. Saligrama, “Adaptive classification for prediction under a budget,” in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4727–4737.
[21] J. Janisch, T. Pevn ỳ, and V. Lis ỳ, “Classification with costly features using deep reinforcement learning,” in *Proc. 33rd AAAI Conf. Artif. Intell.*, 2019, pp. 3959–3966.
[22] D. Kocev, M. Ceci, and T. Stepišnik, “Ensembles of extremely randomized predictive clustering trees for predicting structured outputs,” *Mach. Learn.*, vol. 109, pp. 2213–2241, 2020.
[23] J. Li *et al.*, “Feature selection: A data perspective,” *ACM Comput. Surv.*, vol. 50, no. 6, 2018, Art. no. 94.
[24] S. Hara and T. Maehara, “Enumerate lasso solutions for feature selection,” in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1985–1991.

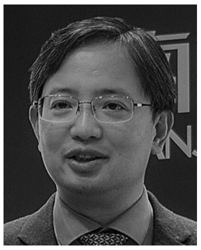
[25] J. Wang *et al.*, "Online feature selection with group structure analysis," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 11, pp. 3029–3041, Nov. 2015.
[26] S. Li, L. Li, J. Yan, and H. He, "SDE: A novel clustering framework based on sparsity-density entropy," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 8, pp. 1575–1587, Aug. 2018.
[27] S. Mallat, *A Wavelet Tour of Signal Processing*. Cambridge, MA, USA: Academic Press, 1999.
[28] X.-S. Wei and Z.-H. Zhou, "An empirical study on image bag generators for multi-instance learning," *Mach. Learn.*, vol. 105, no. 2, pp. 155–198, 2016.
[29] T. G. Dietterich, R. H. Lathrop, and T. Lozano-P érez, "Solving the multiple instance problem with axis-parallel rectangles," *Artif. Intell.*, vol. 89, no. 1/2, pp. 31–71, 1997.
[30] X.-S. Wei, H.-J. Ye, X. Mu, J. Wu, C. Shen, and Z.-H. Zhou, "Multiple instance learning with emerging novel class," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2019.2952588.
[31] J. Wu, S. Pan, X. Zhu, C. Zhang, and X. Wu, "Multi-instance learning with discriminative bag mapping," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 6, pp. 1065–1080, Jun. 2018.
[32] O. Maron and T. Lozano-P érez, "A framework for multiple-instance learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1998, pp. 570–576.
[33] E. Nowak, F. Jurie, and B. Triggs, "Sampling strategies for bag-of-features image classification," in *Proc. Eur. Conf. Comput. Vis.*, 2006, pp. 490–503.
[34] F. Shi, E. Petriu, and R. Laganiere, "Sampling strategies for real-time action recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 2595–2602.
[35] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 1113–1120.
[36] A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan, "The big data bootstrap," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1787–1794.
[37] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2962–2970.
[38] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.
[39] E. Kleinberg, "Stochastic discrimination," *Ann. Math. Artif. Intell.*, vol. 1, no. 1/4, pp. 207–239, 1990.
[40] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1/3, pp. 489–501, 2006.
[41] Y. Park and J. Ho, "Tackling overfitting in boosting for noisy healthcare data," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2019.2959988.
[42] G. I. Webb, "MultiBoosting: A technique for combining boosting and wagging," *Mach. Learn.*, vol. 40, no. 2, pp. 159–196, 2000.
[43] D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992.
[44] L. Breiman, "Stacked regressions," *Mach. Learn.*, vol. 24, no. 1, pp. 49–64, 1996.
[45] K. M. Ting and I. H. Witten, "Issues in stacked generalization," *J. Artif. Intell. Res.*, vol. 10, pp. 271–289, 1999.
[46] Q. Yao *et al.*, "Privacy-preserving stacking with application to cross-organizational diabetes prediction," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 4114–4120.
[47] S.-H. Lyu, L. Yang, and Z.-H. Zhou, "A refined margin distribution analysis for forest representation learning," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 5531–5541.
[48] L. V. Utkin and M. A. Ryabinin, "A siamese deep forest," *Knowl.-Based Syst.*, vol. 139, pp. 13–22, 2018.
[49] L. Yang, X.-Z. Wu, Y. Jiang, and Z.-H. Zhou, "Multi-label learning with deep forest," in *Proc. 24th Eur. Conf. Artif. Intell.*, 2020, pp. 1634–1641.
[50] D. Kocev, C. Vens, J. Struyf, and S. Džeroski, "Tree ensembles for predicting structured outputs," *Pattern Recognit.*, vol. 46, no. 3, pp. 817–833, 2013.
[51] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," *Inf. Softw. Technol.*, vol. 114, pp. 204–216, 2019.
[52] X. Liu, R. Wang, Z. Cai, Y. Cai, and X. Yin, "Deep multigrained cascade forest for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 10, pp. 8169–8183, Oct. 2019.
[53] Z.-H. Chen, L.-P. Li, Z. He, J.-R. Zhou, Y. Li, and L. Wong, "An improved deep forest model for predicting self-interacting proteins from protein sequence using wavelet transformation," *Front. Genetics*, vol. 10, 2019, Art. no. 90.
[54] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics*, 2011, pp. 142–150.
[55] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[56] C. Sapsanis, G. Georgoulas, A. Tzes, and D. Lymberopoulos, "Improving EMG based classification of basic hand movements using EMD," in *Proc. 35th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, 2013, pp. 5754–5757.
[57] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Univ. Toronto, 2009.
[58] M. Pang, K.-M. Ting, P. Zhao, and Z.-H. Zhou, "Improving deep forest by confidence screening," in *Proc. IEEE Int. Conf. Data Mining*, 2018, pp. 1194–1199.
[59] Z.-H. Zhou, "Abductive learning: Towards bridging machine learning and logical reasoning," *Sci. China Inf. Sci.*, vol. 62, no. 7, 2019, Art. no. 76101.
[60] Z.-H. Zhou, "Why over-parameterization of deep neural networks does not overfit?," *Sci. China Inf. Sci.*, vol. 64, no. 1, 2021, Art. no. 116101.



Ming Pang received the BSc degree in computer science and technology from Nanjing University, Nanjing, China, in 2014. Currently, he is currently working toward the PhD degree in the National Key Lab for Novel Software Technology, Nanjing University, China, supervised by prof. Zhi-Hua Zhou. His research interest is mainly on machine learning and data mining. He is currently working on ensemble learning.

Kai Ming Ting received the PhD degree from the University of Sydney, Australia. He worked at the University of Waikato, New Zealand, Deakin University, Australia, Monash University, Australia and Federation University, Australia. He joined Nanjing University, China, in 2020. His current research interests are in the areas of isolation kernel, isolation distributional kernel, ensemble approaches, data mining and machine learning. He co-chaired the Pacific-Asia Conference on Knowledge Discovery and Data Mining PAKDD 2008. He has served as a senior member of program committee for AAAI Conference for AI and PAKDD; a member of program committees for a number of international conferences including ACM SIGKDD, IEEE ICDM, ICML and ECML. Research grants received include those from National Natural Science Foundation of China, US Air Force of Scientific Research (AFOSR/AOARD), Australian Research Council, Toyota Info-Technology Center and Australian Institute of Sport. awards received include the Runner-up Best Paper Award, in 2008 IEEE ICDM, and the Best Paper Award, in 2006 PAKDD.

Peng Zhao received the BSc degree from Tongji University, Shanghai, China, in 2016. Currently, he is working toward the PhD degree in the National Key Lab for Novel Software Technology, Nanjing University, China, supervised by Prof. Zhi-Hua Zhou. His research interest is mainly on machine learning and data mining. He is currently working on robust learning in non-stationary environments.



Zhi-Hua Zhou (Fellow, IEEE) received the BSc, MSc and PhD degrees in computer science from Nanjing University, China, in 1996, 1998 and 2000, respectively, all with the highest honors. He joined the Department of Computer Science & Technology, Nanjing University, China as an assistant professor, in 2001, and is currently professor, head of the Department of Computer Science and Technology, and dean of the School of Artificial Intelligence; he is also the founding director of the LAMDA group. His research interests are mainly in artificial intelligence, machine learning and data mining. He has authored the books *Ensemble Methods: Foundations and Algorithms*, *Evolutionary Learning: Advances in Theories and Algorithms*, *Machine Learning* (in Chinese), and published more than 150 papers in top-tier international journals or conference proceedings. He has received various awards/honors including the National Natural Science Award of China, the IEEE Computer Society Edward J. McCluskey Technical Achievement Award, the CCF-ACM Artificial Intelligence Award, the ACML Distinguished Contribution Award, the PAKDD Distinguished Contribution Award, the IEEE ICDM Outstanding Service Award, the Microsoft Professorship Award, etc. He also holds 24 patents. He is the editor-in-chief of the *Frontiers of Computer Science*, associate editor-in-chief of the *Science China Information Sciences*, action or associate editor of the *Machine Learning*, the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, the *ACM Transactions on Knowledge Discovery from Data*, etc. He served as associate editor-in-chief for the *Chinese Science Bulletin* (2008–2014), associate editor for the *IEEE Transactions on Knowledge and Data Engineering* (2008–2012), the *IEEE Transactions on Neural Networks and Learning Systems* (2014–2017), the *ACM Transactions on Intelligent Systems and Technology* (2009–2017), the *Neural Networks* (2014–2016), etc. He founded ACML (Asian Conference on Machine Learning), served as Advisory Committee member for IJCAI (2015–2016), Steering Committee member for ICDM, PAKDD and PRICAI, and chair of various conferences such as general co-chair of ICDM 2016 and PAKDD 2014, program co-chair of AAAI 2019 and SDM 2013, and area chair of NeurIPS, ICML, AAAI, IJCAI, KDD, etc. He was the chair of the IEEE CIS Data Mining Technical Committee (2015–2016), the chair of the CCF-AI (2012–2019), and the chair of the CAAI Machine Learning Technical Committee (2006–2015). He is a foreign member of the Academy of Europe, and a fellow of the ACM, AAAI, AAAS, IAPR, IET/IEE, CCF, and CAAI.